

Db2 12 for z/OS

*Guía SQL y de programación de
aplicaciones*
Última actualización: 2025-04-18



Notas

Antes de utilizar esta información y el producto al que da soporte, consulte la información general que se encuentra en "Avisos" al final de esta información.

Subsequent editions of this PDF will not be delivered in IBM Publications Center. Always download the latest edition from [IBM Documentation](#).

2025-04-18 edición

Esta edición se aplica a Db2 12 for z/OS (número de producto 5650-DB2), Db2 12 for z/OS Value Unit Edition (número de producto 5770-AF3) y a cualquier release posterior hasta que se indique lo contrario en nuevas ediciones. Compruebe que esté utilizando la edición correcta para el nivel de producto.

Los cambios específicos se indican por medio de una barra vertical a la izquierda del cambio. Una barra vertical a la izquierda del pie de una figura indica que la figura ha cambiado. Los cambios en la edición que no tienen importancia técnica no se han resaltado.

Contenido

Acerca de esta información.....	xi
Quién debe leer esta información.....	xii
Db2 Utilities Suite for z/OS.....	xii
Terminología y referencias.....	xii
Características de accesibilidad para Db2 for z/OS.....	xiii
Cómo enviar los comentarios.....	xiii
Cómo leer los diagramas de sintaxis.....	xiv
Capítulo 1. Planificación y diseño de aplicaciones de Db2.....	1
Incompatibilidades de release SQL y aplicaciones.....	1
La función incorporada SUBSTR siempre devuelve un mensaje de error para una entrada no válida.....	2
Las sentencias CREATE TABLESPACE y CREATE INDEX sin ninguna cláusula USING de nivel de espacio fallan si el grupo de almacenamiento especificado cuando se ha creado la base de datos que lo contiene no existe.....	2
Nuevo número máximo de marcadores de parámetro o variables de host en una única sentencia SQL.....	3
Cambio de resultado para la sentencia SQL EXPLAIN PACKAGE.....	3
Cambio de resultado para la sentencia SQL EXPLAIN STABILIZED DYNAMIC QUERY.....	4
Cambios en la columna DSVOLSER de la tabla de catálogo SYSCOPY.....	4
Los niveles de compatibilidad de aplicación se aplican a las sentencias de definición y control de datos.....	4
Reenlace automático de planes y paquetes creados antes de DB2 10.....	5
Soporte de opción de enlace KEEPNAMIC(YES) para ROLLBACK.....	5
Las alteraciones en la compresión de índices son cambios pendientes para los espacios de tablas universales.....	6
Tipos de datos de argumentos de salida de una llamada de procedimiento almacenado en una aplicación Java.....	6
Sentencias SELECT INTO con UNION o UNION ALL.....	7
Cambiar en SQLCODE cuando el resultado de la función incorporada POWER está fuera de rango.....	8
Funciones CHAR9 y VARCHAR9 para la compatibilidad con el formato de serie anterior a DB2 10 de datos decimales.....	9
Parámetro de subsistema BIF_COMPATIBILITY y esquemas SQL para la compatibilidad con el formato de serie anterior a DB2 10 de datos decimales.....	9
La entrada de serie combinada EBCDIC para las funciones incorporadas RTRIM, TRIM, LTRIM y STRIP debe ser válida.....	10
El número máximo de funciones escalares externas definidas por el usuario que se ejecutan en un hilo de ejecución de aplicaciones (Db2) ya no es ilimitado (APAR PH44833).....	11
Palabras reservadas de SQL.....	11
Funciones incorporadas y funciones definidas por el usuario existentes.....	12
Cambios de SQLCODE.....	15
Determinar el valor de cualquier opción de procesamiento SQL que afecte al diseño de su programa.....	15
Cambios que invalidan paquetes.....	16
Identificación de paquetes invalidados.....	20
Cambios que podrían requerir reempaquetados.....	21
Determinar el valor de cualquier opción vinculante que afecte al diseño de su programa.....	22
Programación de aplicaciones para mejorar el rendimiento.....	22
Diseño de su solicitud de recuperación.....	23

Unidad de trabajo en TSO.....	24
Unidad de trabajo en CICS.....	25
Planificación de la recuperación de programas en IMS programas.....	26
Deshacer los cambios seleccionados en una unidad de trabajo utilizando puntos de salvaguarda.....	33
Planificación para la recuperación de espacios de mesa que no están registrados.....	35
Diseño de la aplicación para acceder a datos distribuidos.....	36
Servidores remotos y datos distribuidos.....	37
Preparación para actualizaciones coordinadas de dos o más fuentes de datos.....	37
Forzar reglas de sistema restringidas en su programa.....	38
Capítulo 2. Conectarse a Db2 desde su programa de aplicación.....	39
Invocación del recurso de conexión de llamada.....	40
Recurso de conexión de llamada.....	42
Disponibilidad de la interfaz de lenguaje CAF (DSNALI).....	45
Requisitos para programas que utilizan CAF.....	47
Cómo modifica CAF el contenido de los registros.....	48
Conexiones implícitas a CAF.....	48
Lista de parámetros de la sentencia CALL DSNALI.....	49
Resumen del comportamiento de CAF.....	51
Funciones de conexión CAF.....	52
Activar un seguimiento de CAF.....	63
Códigos de retorno y códigos de razón de CAF.....	64
Ejemplos de escenarios de CAF.....	65
Ejemplos de invocación de CAF.....	66
Invocación del recurso de conexión de servicios de recuperación de recursos.....	71
Recurso de conexión de Resource Recovery Services.....	73
Poner a disposición la interfaz de lenguaje RRSAF (DSNRLI).....	77
Requisitos para programas que utilizan RRSAF.....	79
Cómo RRSAF modifica el contenido de los registros.....	79
Conexiones implícitas a RRSAF.....	79
Lista de parámetros de la sentencia CALL DSNRLI.....	80
Resumen del comportamiento de RRSAF.....	81
Funciones de conexión RRSAF.....	83
Códigos de retorno y códigos de razón de RRSAF.....	117
Ejemplos de escenarios RRSAF.....	117
Ejemplos de programas para RRSAF.....	119
Interfaz de lenguaje universal (DSNULI).....	121
Edición de enlaces de una aplicación con DSNULI.....	123
Controlar la CICS función de adjuntar desde una aplicación.....	124
Detectar si el CICS funcionamiento del servicio de adjuntos.....	124
Mejorar la reutilización de hilo en CICS aplicaciones.....	125
Capítulo 3. Db2 Programación de SQL.....	127
Creación y modificación de objetos de interfaz de usuario (Db2) desde programas de aplicación.....	127
Creación de tablas a partir de programas de aplicación.....	127
Suministro de clave única para una tabla.....	150
Fijación de tablas con definiciones incompletas.....	150
RENOMBRAR TABLA en un escenario de mantenimiento de tablas.....	151
Descarte de tablas.....	152
Definición de una vista.....	152
Dejar una opinión.....	154
Creación de una expresión de tabla común.....	154
Creación de un desencadenante.....	160
Objetos de secuencia.....	179
Db2 objeto extensiones relacionales.....	181
Creación de un tipo diferenciado.....	181

Creación de una función definida por el usuario.....	189
Creación de procedimientos almacenados.....	226
Adición y modificación de datos en tablas de programas de aplicación.....	352
Inserción de datos en tablas.....	352
Añadir datos al final de una tabla.....	367
Almacenamiento de datos que no tienen un formato tabular.....	367
Actualización de datos de tabla.....	367
Supresión de datos de tablas.....	370
Acceso a datos en tablas desde programas de aplicación.....	373
Determinar a qué mesas tiene acceso.....	373
Mostrar información sobre las columnas de una tabla determinada.....	373
Recuperación de datos utilizando la sentencia SELECT.....	374
Recuperación de un conjunto de filas utilizando un cursor.....	423
Especificación del acceso a filas directo utilizando los ID de fila.....	455
Formas de manipular datos LOB.....	457
Referenciar un objeto de secuencia.....	471
Recuperación de miles de filas.....	471
Determinar cuándo se cambió una fila.....	471
Comprobación de si una columna XML contiene un valor determinado.....	472
Acceder a datos de Db2 que no están en una tabla.....	473
Garantizar que las consultas se realicen de manera satisfactoria.....	473
Elementos que deben incluirse en un programa DL/I por lotes.....	474
Invocación de una función definida por el usuario.....	476
Cómo determina Db2 la autorización para invocar funciones definidas por el usuario.....	477
Asegurarse de que Db2 ejecute la función definida por el usuario prevista.....	478
Cómo resuelve Db2 las funciones.....	479
Comprobación de cómo resuelve Db2 las funciones utilizando DSN_FUNCTION_TABLE.....	482
Restricciones al pasar argumentos con tipos distintos a funciones.....	482
Casos en los que Db2 convierte argumentos para una función definida por el usuario.....	484

Capítulo 4. Programación de SQL incorporado..... 487

Descripción general de las aplicaciones de programación que acceden a datos de Db2 for z/OS.....	487
Declaración de definiciones de vista y tabla.....	489
DCLGEN (generador de declaraciones).....	490
Generación de declaraciones de tablas y vistas con DCLGEN.....	490
Incluir declaraciones de DCLGEN en su programa.....	498
Ejemplo: Añadir declaraciones de DCLGEN a una biblioteca.....	498
Definición de elementos que su programa puede utilizar para comprobar si una sentencia SQL se ha ejecutado correctamente.....	501
Definición de áreas de descriptor de SQL (SQLDA).....	502
Declaración de variables host y variables de indicación.....	503
Variables de host.....	503
Matrices de variables de host.....	504
Estructuras host.....	506
Variables de indicación, matrices y estructuras.....	506
Establecimiento del CCSID para variables host.....	508
Determinar qué causó un error al recuperar datos en una variable de host.....	509
Acceder a un módulo predeterminado de la aplicación.....	510
Compatibilidad de SQL y tipos de datos de lenguaje.....	511
Uso de variables host en sentencias SQL.....	514
Recuperación de una única fila de datos en variables host.....	514
Determinación de si un valor recuperado en una variable host es nulo o está truncado.....	517
Actualización de datos utilizando variables de host.....	518
Inserción de una sola fila utilizando una variable host.....	519
Utilización de matrices de variables de host en sentencias SQL.....	520
Recuperación de varias filas de datos en matrices de variables de host.....	520
Inserción de varias filas de datos desde matrices de variables de host.....	521

Inserción de valores nulos en columnas utilizando las matrices o variables indicadoras.....	522
Recuperación de una única fila de datos en una estructura de host.....	523
Inclusión de SQL dinámico en el programa.....	524
Diferencias entre SQL estático y dinámico.....	525
Posibles idiomas de host para aplicaciones SQL dinámicas.....	528
IIInclusión de SQL dinámico para sentencias no SELECT en el programa.....	529
Incluir SQL dinámico para instrucciones SELECT de lista fija en su programa.....	530
Inclusión de SQL dinámico para sentencias SELECT de lista variable en el programa.....	532
Ejecución dinámica de una sentencia SQL utilizando EXECUTE IMMEDIATE.....	548
Ejecución dinámica de una sentencia SQL utilizando PREPARE y EXECUTE.....	549
Ejecución dinámica de una sentencia de cambio de datos.....	552
Ejecución dinámica de una sentencia con marcadores de parámetros mediante el uso de SQLDA.....	555
Comprobación de la ejecución de sentencias SQL.....	556
Comprobación de la ejecución de sentencias SQL utilizando SQLCA.....	557
Comprobación de la ejecución de sentencias SQL utilizando SQLCODE y SQLSTATE.....	561
Comprobación de la ejecución de sentencias SQL mediante la sentencia WHENEVER.....	562
Comprobación de la ejecución de sentencias SQL utilizando la instrucción GET DIAGNOSTICS... ..	563
Tipos de datos para los elementos GET DIAGNOSTICS.....	565
Manejo de códigos de error de SQL.....	569
Errores aritméticos y de conversión.....	570
Escritura de aplicaciones que permiten al usuario crear y modificar tablas.....	570
Guardar sentencias SQL que se convierten en solicitudes de usuario.....	571
Datos XML en aplicaciones SQL incorporadas.....	571
Tipos de datos de variable host para datos XML en aplicaciones de SQL incorporado.....	572
Actualizaciones de columnas XML en aplicaciones SQL incrustadas.....	577
Recuperación de datos XML en aplicaciones SQL incrustadas.....	579
Programas de ejemplo que llaman a procedimientos almacenados.....	582
Aplicaciones ensambladoras que emiten sentencias SQL.....	582
Ejemplos de programación de ensamblador.....	585
Definición del área de comunicación SQL, SQLSTATE y SQLCODE en el ensamblador.....	585
Definición de áreas de descriptor de SQL (SQLDA) en ensamblador.....	586
Declaración de variables de host y variables de indicador en ensamblador.....	587
Tipos de datos SQL y ensamblador equivalentes.....	594
Macros para aplicaciones ensambladoras.....	601
Gestión de códigos de error de SQL en aplicaciones de ensamblador.....	602
Aplicaciones C y C++ que emiten sentencias SQL.....	603
Ejemplos de programación en C y C++.....	606
Definición del área de comunicaciones SQL, SQLSTATE y SQLCODE en C y C++.....	614
Definición de áreas de descriptor de SQL (SQLDA) en C y C++.....	615
Declaración de variables de host y variables de indicador en C y C++.....	616
SQL equivalente y tipos de datos C.....	644
Gestión de códigos de error de SQL en aplicaciones C y C++.....	652
Aplicaciones COBOL que emiten sentencias SQL.....	654
Ejemplos de programación de COBOL.....	659
Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en COBOL.....	684
Definición de áreas de descriptor de SQL (SQLDA) en COBOL.....	685
Declaración de variables host y variables de indicador en COBOL.....	686
SQL equivalente y tipos de datos COBOL.....	714
Extensiones orientadas a objetos en COBOL.....	720
Gestión de códigos de error de SQL en aplicaciones Cobol.....	721
Aplicaciones Fortran que emiten sentencias SQL.....	722
Definición del área de comunicación SQL, SQLSTATE y SQLCODE en Fortran.....	724
Definición de áreas de descriptor de SQL en (SQLDA) Fortran.....	725
Declaración de variables de host y variables de indicador en Fortran.....	726
SQL equivalente y tipos de datos Fortran.....	731
Aplicaciones PL/I que emiten sentencias SQL.....	734
Ejemplos de programación de PL/I.....	738

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en PL/I.....	743
Definición de áreas de descriptor de SQL (SQLDA) en PL/I.....	744
Declaración de variables host y variables de indicador en PL/I.....	745
SQL equivalente y tipos de datos PL/I.....	760
Aplicaciones REXX que emiten sentencias SQL.....	766
Ejemplos de programación de REXX.....	769
Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en REXX.....	784
Definición de áreas de descriptor de SQL (SQLDA) en REXX.....	785
SQL equivalente y tipos de datos REXX.....	785
Acceso a la aplicación de programación de interfaces de lenguaje REXX de Db2	787
Asegurarse de que Db2 interprete correctamente los datos de entrada de caracteres en los programas REXX.....	789
Pasar el tipo de datos de un tipo de datos de entrada a Db2 para programas REXX.....	790
Configuración del nivel de aislamiento de las sentencias SQL en un programa REXX.....	790
Recuperación de datos de tablas de Db2 es en programas REXX.....	791
Cursos y nombres de sentencias en REXX.....	792
Gestión de códigos de error de SQL en aplicaciones REXX.....	793

Capítulo 5. Invocación de un procedimiento almacenado desde una aplicación... 795

Pase de parámetros de salida de gran tamaño a procedimientos almacenados mediante variables de indicador.....	800
Tipos de datos para llamar a procedimientos almacenados.....	801
Llamar a un procedimiento almacenado desde un procedimiento REXX.....	801
Preparación de un programa cliente que invoca un procedimiento almacenado remoto.....	805
Cómo determina Tiffany & Co. (Db2) qué procedimiento almacenado ejecutar.....	805
Llamar a diferentes versiones de un procedimiento almacenado desde una sola aplicación.....	806
Invocar varias instancias de un procedimiento almacenado.....	807
Designación de la versión activa de un procedimiento de SQL nativo.....	808
Omisión temporal de la versión activa de un procedimiento de SQL nativo.....	809
Especificación del número de procedimientos almacenados que pueden ejecutarse de forma simultánea.....	809
Recuperación del estado de procedimiento.....	810
Escribir un programa para recibir los conjuntos de resultados de un procedimiento almacenado.....	811

Capítulo 6. Métodos de codificación de datos distribuidos..... 817

Acceso a datos distribuidos utilizando nombres de tablas de tres partes.....	817
Acceso a mesas temporales declaradas remotas mediante el uso de nombres de mesa de tres partes.....	819
Acceso a datos distribuidos utilizando sentencias CONNECT explícitas.....	820
Especificación de un nombre de alias de ubicación para varios sitios.....	821
Liberar conexiones.....	821
Transmisión de datos mixtos.....	822
Identificación del servidor en tiempo de ejecución.....	822
Limitaciones de SQL en servidores diferentes.....	822
Soporte para la ejecución de sentencias de SQL largas en un entorno distribuido.....	823
Consultas distribuidas en tablas ASCII o Unicode.....	823
Restricciones al utilizar cursos desplazables para acceder a datos distribuidos.....	824
Restricciones al utilizar cursos posicionados en filas para acceder a datos distribuidos.....	824
IBM MQ con Db2.....	824
Mensajes de IBM MQ.....	824
Funciones de MQ de Db2 y procedimientos almacenados XML y MQ de Db2.....	827
Generación de documentos XML a partir de tablas existentes y posterior envío a una cola de mensajes de MQ.....	829
Fragmentación de documentos XML desde una cola de mensajes de MQ.....	830
Tablas MQ de Db2.....	830
Mensajes básicos con IBM MQ.....	842
Envío de mensajes con IBM MQ.....	843

Recuperación de mensajes con IBM MQ.....	844
Conectividad de aplicación a aplicación con IBM MQ.....	845
Mensajería asíncrona en Db2 for z/OS.....	846
Capítulo 7. Db2 como consumidor y proveedor de servicios web.....	861
En desuso: Funciones definidas por el usuario SOAPHTTPV y SOAPHTTPC.....	861
Funciones definidas por el usuario SOAPHTTPNV y SOAPHTTPNC.....	863
SQLSTATEs para Db2 como consumidor de servicios web.....	863
Capítulo 8. Niveles de compatibilidad de aplicaciones en Db2 12.....	867
Niveles de compatibilidad de aplicaciones V12R1Mnnn.....	869
Establecimiento de niveles de compatibilidad de aplicaciones para clientes y controladores de servidores de datos.....	870
DSNTIJLC.....	872
DSNTIJLR.....	875
Usar tablas de perfiles para controlar qué niveles de compatibilidad de aplicaciones de Db2 for z/OS se usar para aplicaciones específicas de cliente de servidor de datos.....	876
Nivel de compatibilidad de aplicaciones V11R1.....	878
Nivel de compatibilidad de aplicaciones V10R1.....	884
Gestión de incompatibilidades de aplicaciones.....	888
Habilitar la compatibilidad de aplicaciones predeterminada con nivel de función 500 o superior.....	889
Capítulo 9. Preparación de una aplicación para su ejecución en Db2 for z/OS.....	891
Configuración de los valores predeterminados de DB2I.....	894
Procesamiento de instrucciones SQL para la preparación de programas.....	896
Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor.....	897
Procesamiento de sentencias SQL mediante el uso del Db2 precompilador.....	901
Diferencias entre el coprocesador Db2 y el Db2 precompilador.....	910
Traducción de instrucciones de nivel de comando en un CICS programa.....	911
Opciones para el proceso de sentencias de SQL.....	913
Compilación y edición de enlaces de una aplicación.....	926
Enlace de planes y paquetes de aplicación.....	927
Crear una versión de paquete.....	929
Vinculación de un DBRM que está en un archivo HFS a un paquete o colección.....	930
Enlace de un plan de aplicación.....	933
Proceso de enlace para acceso remoto.....	937
Enlazar un programa por lotes.....	941
Conversión de DBRM vinculados a un plan en DBRM vinculados a un paquete.....	942
Convertir un plan existente en paquetes para ejecutarlo de forma remota.....	943
Establecimiento del nivel de programa.....	944
Opciones de reglas dinámicas para sentencias de SQL dinámico.....	944
Selección dinámica de planes.....	946
Revinculación de aplicaciones.....	947
Revinculación de un paquete.....	948
Introducción gradual de los reenvíos de paquetes.....	950
Volver a encuadrinar un plan.....	951
Reencuadrinación de listas de planes y paquetes.....	952
Generación de listas de comandos REBIND.....	952
Revinculación automática.....	957
Especificación de las reglas que se aplican a un comportamiento SQL en tiempo de ejecución.....	959
Conjuntos de datos de entrada y salida para trabajos por lotes DL/I.....	960
Procedimientos JCL proporcionados por Db2 para la preparación de una aplicación.....	962
JCL para incluir el código de interfaz adecuado al utilizar los procedimientos JCL suministrados por Db2.....	963
Adaptación de los procedimientos JCL suministrados por Db2, para la preparación de CICS programas.....	964
Paneles de DB2I que se utilizan para la preparación del programa.....	965

Panel Preparación del programa de Db2.....	967
Panel de valores predeterminados 1 de DB2I.....	971
Panel 2 de valores predeterminados de DB2I.....	973
Panel Precompilar.....	974
Panel Enlazar paquete.....	976
Panel Enlazar plan.....	979
Paneles Valores predeterminados para BIND PACKAGE y Valores predeterminados REBIND PACKAGE.....	982
Paneles Valores predeterminados para BIND PLAN y Valores predeterminados REBIND PLAN.....	984
Panel Tipos de conexión del sistema.....	986
Paneles para entrar listas de valores.....	988
Preparación del programa: Panel Compilar, Enlazar y Ejecutar.....	989
Paneles de DB2I que se utilizan para volver a enlazar y liberar planes y paquetes.....	990
Panel Enlazar/Volver a enlazar/liberar selección.....	991
Volver a vincular panel de paquete.....	992
Reenlazar Panel de activación del paquete.....	994
Volver a vincular el panel del plan.....	997
Panel de paquete gratuito.....	998
Panel Liberar plan.....	999

Capítulo 10. Ejecutar una aplicación en Db2 for z/OS..... 1001

Procesador de mandatos de DSN.....	1001
Panel Ejecutar de DB2I.....	1002
Ejecución de un programa en primer plano en TSO.....	1003
Ejecutar una aplicación REXX de e Db2	1004
Invocar programas a través del Interactive System Productivity Facility.....	1004
ISPF.....	1004
Invocación de un único programa SQL a través de ISPF y DSN.....	1006
Invocación de múltiples programas SQL a través de ISPF y DSN.....	1006
Carga y ejecución de un programa por lotes.....	1007
Autorización para ejecutar un programa de DL/I por lotes.....	1008
Reiniciar un programa por lotes.....	1009
Ejecución de procedimientos almacenados desde el Db2 command line processor.....	1011
Db2 command line processor Sentencia CALL.....	1011
Ejemplo de ejecución de una aplicación de batch Db2 en TSO.....	1012
Ejemplo de invocación de aplicaciones en un procedimiento de mandatos.....	1013

Capítulo 11. Prueba y depuración de un programa de aplicación en Db2 for z/OS1015

Diseño de una estructura de datos de prueba.....	1015
Analizar las necesidades de datos de la aplicación.....	1015
Autorización para tablas de prueba y aplicaciones.....	1017
Ejemplo de instrucciones SQL para crear una estructura de prueba completa.....	1017
Introducción de datos en las tablas de prueba.....	1018
Métodos para probar sentencias SQL.....	1018
Ejecución de SQL utilizando SPUFI.....	1019
Contenido de un conjunto de datos de entrada SPUFI.....	1023
El panel SPUFI.....	1023
Cambio de los valores predeterminados de SPUFI.....	1025
Establecimiento del carácter terminador de SQL en un conjunto de datos de entrada SPUFI.....	1030
Control de la tolerancia de avisos en SPUFI.....	1031
Salida de SPUFI.....	1032
Probar una función externa definida por el usuario.....	1033
Probar una función definida por el usuario mediante z/OS Debugger.....	1033
Probar una función definida por el usuario enviando los mensajes de depuración a SYSPRINT..	1035
Probar una función definida por el usuario mediante aplicaciones de controlador.....	1035
Probar una función definida por el usuario mediante sentencias SQL INSERT.....	1035
Depuración de procedimientos almacenados.....	1036

Depuración de procedimientos almacenados mediante el uso del Unified Debugger.....	1037
Depuración de procedimientos almacenados con z/OS Debugger.....	1038
Grabar mensajes de depuración de procedimientos almacenados en un archivo.....	1039
Aplicaciones de controlador para procedimientos de depuración.....	1040
Db2 tablas que contienen información de depuración.....	1040
Depuración de un programa de aplicación.....	1040
Localización del problema en una aplicación.....	1041
Técnicas para depurar programas en TSO.....	1045
Técnicas para depurar programas en IMS.....	1046
Técnicas para depurar programas en CICS.....	1047
Encontrar una referencia violada o una restricción de verificación.....	1050
Capítulo 12. Datos de muestra y aplicaciones suministrados con Db2 for z/OS... 1053	
Tablas de ejemplo de Db2.....	1053
Tabla de actividades (DSN8C10.ACT).....	1053
Tabla de departamentos (DSN8C10.DEPT).....	1054
Tabla de empleados (DSN8C10.EMP).....	1056
Tabla de fotografías y currículums de empleados (DSN8C10.EMP_PHOTO_RESUME).....	1060
Tabla de proyectos (DSN8C10.PROJ).....	1061
Tabla de actividades de proyectos (DSN8C10.PROJECT).....	1062
Tabla de empleados de actividades de proyectos (DSN8C10.EMPPROJECT).....	1063
Tabla de ejemplo Unicode (DSN8C10.DEMO_UNICODE).....	1064
Relaciones entre las tablas de ejemplo.....	1065
Vistas en las tablas de ejemplo.....	1066
Almacenamiento de tablas de aplicaciones de ejemplo.....	1070
Tablas SYSDUMMYx.....	1073
Db2 programas de muestra de ayuda a la productividad.....	1074
Programa de ejemplo DSNTIAUL.....	1076
Programa de ejemplo DSNTIAD.....	1081
Programas de ejemplo DSNTEP2 y DSNTEP4.....	1084
Aplicaciones de ejemplo suministradas con Db2 for z/OS.....	1091
Tipos de aplicaciones de muestra.....	1092
Entornos y lenguajes de aplicación para las aplicaciones de ejemplo.....	1094
Ejemplos de aplicaciones en TSO.....	1095
Ejemplos de aplicaciones en IMS.....	1415
Ejemplos de aplicaciones en CICS.....	1461
Recursos de información para Db2 for z/OS y productos relacionados..... 1521	
Avisos..... 1523	
Información de la interfaz de programación.....	1524
Marcas comerciales.....	1525
Términos y condiciones de la documentación de producto.....	1525
Consideraciones sobre la política de privacidad.....	1525
Glosario..... 1527	
Índice..... 1529	

Acerca de esta información

Esta información trata sobre cómo diseñar y escribir programas de aplicación que accedan a Db2 for z/OS (Db2), un sistema de gestión de bases de datos relacionales (DBMS) altamente flexible.

A lo largo de este contenido, "Db2" significa "Db2 12 for z/OS." Las referencias a otros productos de Db2 utilizan nombres completos o abreviaturas más específicas.

Importante: Para encontrar el contenido más actualizado sobre Db2 12 for z/OS, utilice siempre Documentación de IBM® o descargue el archivo PDF más reciente desde [Manuales en formato PDF para Db2 12 for z/OS \(Db2 for z/OS en IBM Documentation\)](#).

Esta documentación del producto Db2 12 for z/OS asume generalmente que el nivel de función más alto disponible está activado y que sus aplicaciones se ejecutan con el nivel de compatibilidad de aplicaciones más alto disponible, con las siguientes excepciones:

- Los apartados siguientes referentes a la documentación describen el proceso de migración de Db2 12 y cómo activar nuevas funciones en los niveles de función:
 - [Migración a Db2 12 \(Db2 Instalación y migración\)](#)
 - [¿Qué hay de nuevo en Db2 12 \(Db2 for z/OS ¿Qué hay de nuevo?\)](#)
 - [Adoptar nuevas capacidades en Db2 12 entrega continua \(Db2 for z/OS ¿Qué hay de nuevo?\)](#)
- **FL 501** Una etiqueta como esta suele marcar la documentación cambiada para el nivel de función 500 o superior, con un enlace a la descripción del nivel de función que introduce el cambio en Db2 12. Para obtener más información, consulte [Cómo se documentan los niveles de función de Db2 \(Db2 para z/OS: Novedades\)](#).

Entrega continua en Db2 12

La disponibilidad de la mayoría de las capacidades de las aplicaciones con nuevas funciones en Db2 12 depende del tipo de mejora, del nivel de función activado y de los niveles de compatibilidad de cada aplicación. Para obtener una lista de todos los niveles de función disponibles actualmente en Db2 12, consulte [Db2 12Niveles de función \(¿Qué hay de nuevo Db2 for z/OS?\)](#).

Muchas mejoras de nuevas funciones surten efecto en cualquier nivel de función cuando se aplica un PTF en cada subsistema Db2 o miembro de partición de datos. Para obtener más información, consulte [APAR de nueva función para Db2 12 \(Db2 for z/OS ¿Qué hay de nuevo?\)](#).

Mejoras del almacenamiento virtual

Las mejoras de almacenamiento virtual pasan a estar disponibles cuando se activa el nivel de función en el que se han incluido y otro superior. La activación de nivel de función 100 introduce todas las mejoras de almacenamiento virtual en el release inicial de Db2 12. Es decir, la activación de nivel de función 500 no introduce ninguna mejora del almacenamiento virtual.

Parámetros de subsistema

Los nuevos valores de parámetro de subsistema solo entran en vigor cuando se activa el nivel de función que los ha introducido o un nivel de función superior. Muchos de los cambios de parámetros de subsistema en el release inicial de Db2 12 se aplican en nivel de función 500. Para obtener más información sobre los cambios de parámetros del subsistema en Db2 12, consulte [Cambios en los parámetros del subsistema en Db2 12 \(Db2 for z/OS ¿Qué hay de nuevo?\)](#).

Mejoras de optimización

Las mejoras de optimización están disponibles después de la activación del nivel de función que las introduce o un nivel de función superior, y de la preparación completa de las sentencias SQL. El momento en el que se produce una preparación completa depende del tipo de sentencia:

- Para las sentencias SQL estáticas, después de enlazar o volver a enlazar el paquete
- Para las sentencias SQL dinámicas no estabilizadas, inmediatamente, a menos que la sentencia esté en la memoria caché de sentencias dinámicas

- Para las sentencias SQL dinámicas estabilizadas, después de la invalidación, con un nivel de compatibilidad de aplicaciones libre o modificado

La activación de nivel de función 100 introduce todas las mejoras de optimización en el release inicial de Db2 12. Es decir, nivel de función 500 no introduce ninguna optimización de mejora.

Prestaciones de SQL

Las nuevas funciones de SQL están disponibles después de la activación del nivel de función que los introduce o un nivel superior, para las aplicaciones que se ejecutan en el nivel de compatibilidad de aplicación equivalente o superior. Las nuevas funciones de SQL en el release inicial de Db2 12 están disponibles en nivel de función 500 para las aplicaciones que se ejecutan en el nivel de compatibilidad de aplicación equivalente o superior. Puede continuar ejecutando la compatibilidad de sentencias SQL con niveles de función más bajos o releases anteriores de Db2 11, incluidos Db2 y DB2 10. Para obtener información detallada, consulte [Capítulo 8, “Niveles de compatibilidad de aplicaciones en Db2 12”, en la página 867.](#)

Quién debe leer esta información

Esta información está dirigida a desarrolladores de aplicaciones de Db2 , que estén familiarizados con el Lenguaje de Consulta Estructurado (SQL) y que conozcan uno o más lenguajes de programación compatibles con Db2 .

Db2 Utilities Suite for z/OS

Importante: Db2 Utilities Suite for z/OS está disponible como un producto opcional. Debe solicitar y adquirir por separado una licencia para dichos programas de utilidad, y cuando en esta publicación se habla de las funciones de estos programas de utilidad no implica que tenga una licencia sobre los mismos.

Los programas de utilidad de Db2 12 pueden utilizar el programa DFSORT independientemente de si ha adquirido una licencia para DFSORT en el sistema. Para obtener más información sobre DFSORT, consulte <https://www.ibm.com/support/pages/dfsort>.

Los programas de utilidad de Db2 pueden utilizar IBM Db2 Sort for z/OS como alternativa a DFSORT para las funciones SORT y MERGE de programa de utilidad. El uso de Db2 Sort for z/OS requiere la adquisición de una licencia de Db2 Sort for z/OS. Para obtener más información sobre Db2 Sort for z/OS, consulte la documentación de Db2 Sort for z/OS.

Conceptos relacionados

[Paquetes de utilidades de Db2 \(Db2 Utilities\)](#)

Terminología y referencias

Cuando se hace referencia a un producto de Db2 distinto de Db2 for z/OS, en esta información se utiliza el nombre completo del producto para evitar ambigüedades.

Los términos siguientes se utilizan de la forma indicada:

Db2

Representa el programa bajo licencia Db2 o un subsistema concreto de Db2.

IBM rebranded DB2 to Db2, and Db2 for z/OS is the new name of the offering that was previously known as "DB2 for z/OS". As a result, you might sometimes still see references to the original names, such as "DB2 for z/OS" and "DB2", in different IBM web pages and documents. If the PID, Entitlement Entity, version, modification, and release information match, assume that they refer to the same product.

IBM OMEGAMON for Db2 Performance Expert on z/OS

Consulte cualquiera de los productos siguientes:

- IBM IBM OMEGAMON for Db2 Performance Expert on z/OS
- IBM Db2 Performance Monitor on z/OS

- IBM Db2 Performance Expert for Multiplatforms and Workgroups
- IBM Db2 Buffer Pool Analyzer for z/OS

C, C++ y lenguaje C

Representa el lenguaje de programación C o C++.

CICS

Representa CICS Transaction Server for z/OS.

IMS

Representa IMS Database Manager o IMS Transaction Manager.

MVS

Representa el elemento MVS del sistema operativo z/OS, que es equivalente al componente Programa de control base (BCP) del sistema operativo z/OS.

RACF

Representa las funciones que proporciona el componente RACF de z/OS Security Server.

Características de accesibilidad para Db2 for z/OS

Las características de accesibilidad ayudan al usuario con incapacidades físicas, como por ejemplo movilidad restringida o visión limitada, a utilizar satisfactoriamente productos de tecnología de la información.

Funciones de accesibilidad

La lista siguiente incluye las principales características de accesibilidad de los productos z/OS, incluido Db2 for z/OS. Estas características dan soporte a:

- Operaciones de sólo teclado
- Interfaces utilizadas habitualmente por lectores de pantalla y amplificadores de pantalla.
- Personalización de atributos de visualización como color, contraste y tamaño de font

Consejo: Documentación de IBM (que incluye información para Db2 for z/OS) y sus publicaciones relacionadas están habilitadas para la accesibilidad para IBM Home Page Reader. Puede utilizar todas las características mediante el teclado en lugar del ratón.

Navegación con el teclado

Para obtener información sobre cómo navegar por los paneles de ISPF de Db2 for z/OS utilizando TSO/E o ISPF, consulte *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide* y *z/OS ISPF User's Guide*. Estas guías describen cómo navegar por cada una de estas interfaces, incluyendo la utilización de atajos de teclado o teclas de función (teclas PF). Esta guía incluye los valores por omisión para las teclas PF y explica cómo modificar estas funciones.

Información relacionada sobre la accesibilidad

IBM y la accesibilidad

Para obtener más información sobre el compromiso que IBM tiene con la accesibilidad, consulte *el Centro de accesibilidad de IBM* en <http://www.ibm.com/able>.

Cómo enviar sus comentarios sobre la documentación de Db2 for z/OS

Sus comentarios ayudan a IBM a proporcionar documentación de calidad.

Envíe sus comentarios sobre Db2 for z/OS y la documentación del producto relacionada por correo electrónico a db2zinfo@us.ibm.com.

Para ayudarnos a responder a su comentario, incluya la siguiente información en su correo electrónico:

- El nombre y la versión del producto
- La dirección (URL) de la página, para comentarios sobre la documentación en línea
- El nombre del manual y la fecha de publicación, para comentarios sobre manuales en PDF
- El tema o el título de la sección
- El texto específico que está comentando y su comentario

Conceptos relacionados

[Acerca de la documentación de productos de Db2 12 for z/OS \(Db2 for z/OS en IBM Documentation \)](#)

Referencia relacionada

[Manuales en formato PDF para Db2 12 for z/OS \(Db2 for z/OS en IBM Documentation \)](#)

Cómo leer los diagramas de sintaxis

Existen ciertos convenios para los diagramas de sintaxis que se utilizan en la documentación de IBM.

Aplique las siguientes reglas cuando lea diagramas de sintaxis que se utilizan en la documentación de Db2 for z/OS:

- Lea los diagramas de sintaxis de izquierda a derecha, de arriba a abajo, siguiendo la vía de acceso de la línea.

El símbolo  indica el principio de una sentencia.

El símbolo  indica que la sintaxis de la sentencia continúa en la línea siguiente.

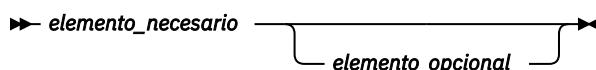
El símbolo  indica que una sentencia continúa de la línea anterior.

El símbolo  indica el final de una sentencia.

- Los elementos necesarios aparecen en la línea horizontal (en la vía de acceso principal).

 *elemento_necesario* 

- Los elementos opcionales aparecen bajo la vía de acceso principal.



Si un elemento opcional aparece sobre la vía de acceso principal, ese elemento no tiene ningún efecto sobre la ejecución de la sentencia y se utiliza sólo por razones de legibilidad.

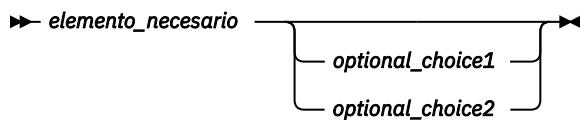


- Si puede elegir entre dos o más elementos, éstos aparecerán verticalmente, en una pila.

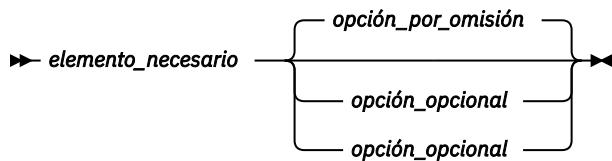
Si *debe* elegir uno de los elementos, un elemento de la pila aparece en la vía de acceso principal.



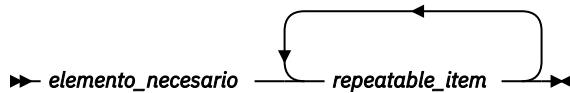
Si la elección de uno de los elementos es opcional, la pila entera aparece bajo la vía de acceso principal.



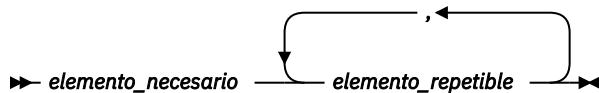
Si uno de los elementos es el valor por omisión, aparece sobre la vía de acceso principal y las opciones restantes se muestran a continuación.



- Una flecha que vuelve a la izquierda, sobre la línea principal, indica un elemento que se puede repetir.

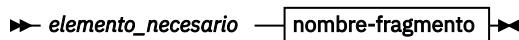


Si la flecha de repetición contiene una coma, debe separar los elementos repetidos con una coma.

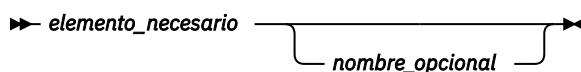


Una flecha de repetición sobre una pila indica que puede repetir los elementos de la pila.

- A veces un diagrama se debe dividir en fragmentos. El fragmento de sintaxis se muestra de forma independiente del diagrama de sintaxis principal, pero el contenido del fragmento se debe leer como si estuviera en la vía de acceso principal del diagrama.



nombre-fragmento



- Para algunas referencias de diagramas de sintaxis, debe seguir las reglas descritas en la descripción para dicho diagrama y también las reglas que se describen en otros diagramas de sintaxis. Por ejemplo:
 - Para *expresión*, también debe seguir las reglas descritas en [Expresiones \(Db2 SQL\)](#).
 - Para obtener referencias a *fullselect*, también debe seguir las reglas descritas en [fullselect \(Db2 SQL\)](#).
 - Para obtener referencias a *search-condition*, también debe seguir las reglas descritas en [Condiciones de búsqueda \(Db2 SQL\)](#).
- Con la excepción de las palabras clave XPath, las palabras clave aparecen en mayúsculas (por ejemplo, FROM). Las palabras clave deben escribirse exactamente como se muestran.
- Las palabras clave XPath se definen como nombres en minúsculas y deben escribirse exactamente como se muestran.
- Las variables aparecen completamente en minúsculas (por ejemplo, *nombre-columna*). Representan nombres o valores proporcionados por el usuario.
- Si se muestran signos de puntuación, paréntesis, operadores aritméticos u otros símbolos de este tipo, debe especificarlos como parte de la sintaxis.

Conceptos relacionados

[Acerca de los comandos en Db2 for z/OS \(Db2 Comandos\)](#)

[Utilidades en línea de Db2 \(Db2 Utilities\)](#)

[Utilidades autónomos de Db2 \(Db2 Utilities\)](#)

Capítulo 1. Planificación y diseño de aplicaciones de Db2

Es necesario tomar decisiones acerca de la planificación y el diseño antes de escribir o ejecutar el programa. Es necesario tomar estas decisiones tanto si escribe una nueva aplicación de Db2 como si migra una aplicación existente de un release anterior de Db2.

Acerca de esta tarea

Si está migrando una aplicación existente desde una versión anterior de Db2, lea las incompatibilidades de la aplicación y la versión de SQL y realice los cambios necesarios en la aplicación.

Si está escribiendo una nueva aplicación de Db2 , primero determine los siguientes elementos:

- el valor de algunas de las opciones de procesamiento SQL
- el método de encuadernación
- el valor de algunas de las opciones de encuadernación

A continuación, asegúrese de que su programa implementa las recomendaciones adecuadas para que promueva la concurrencia, pueda manejar situaciones de recuperación y reinicio, y pueda acceder de manera eficiente a los datos distribuidos.

Conceptos relacionados

[Herramientas e IDE para desarrollar aplicaciones de Db2 \(Introducción a Db2 para z/OS\)](#)

Tareas relacionadas

[Programación de aplicaciones para mejorar el rendimiento \(Db2 Performance\)](#)

[Programación para mejorar la simultaneidad \(Db2 Performance\)](#)

[Escritura eficiente de consultas SQL \(Db2 Performance\)](#)

[Mejora del rendimiento para las aplicaciones que acceden a datos distribuidos \(Db2 Performance\)](#)

Referencia relacionada

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2 \)](#)

Incompatibilidades de release SQL y aplicaciones

Cuando migre o aplique el mantenimiento en Db2 12, tenga en cuenta y planifique las incompatibilidades de aplicaciones y SQL entre releases que puedan afectar al entorno de Db2.

GUPI

Los siguientes cambios incompatibles se aplican a cualquier nivel de función de Db2 12, incluyendo la primera vez que ha migrado a Db2 12. Para cambios incompatibles quede pueden repercutir en el entorno de Db2 12 al activar los niveles de función 501 y superiores, consulte [Resumen de cambios incompatibles para los niveles de función 501 y superiores \(Novedades de DB2 para z/OS\)](#).

Prestaciones de SQL

Las nuevas funciones de SQL están disponibles después de la activación del nivel de función que los introduce o un nivel superior, para las aplicaciones que se ejecutan en el nivel de compatibilidad de aplicación equivalente o superior. Las nuevas funciones de SQL en el release inicial de Db2 12 están disponibles en nivel de función 500 para las aplicaciones que se ejecutan en el nivel de compatibilidad de aplicación equivalente o superior. Puede continuar ejecutando la compatibilidad de sentencias SQL con niveles de función más bajos o releases anteriores de Db2 11, incluidos Db2 y DB2 10. Para obtener información detallada, consulte [Capítulo 8, “Niveles de compatibilidad de aplicaciones en Db2 12”, en la página 867](#).

Las incompatibilidades de versión que se modificaron o añadieron desde la primera edición de esta publicación de Db2 12 se indican por medio de la barra vertical del margen izquierdo. En otras áreas de

esta publicación, la barra vertical del margen indica un cambio o adición que se ha producido desde el release Db2 11 de esta publicación.

La función incorporada SUBSTR siempre devuelve un mensaje de error para una entrada no válida

Anteriormente, durante la ejecución de la función incorporada SUBSTR, Db2 a veces devolvía incorrectamente un resultado para una entrada no válida en lugar de emitir un mensaje de error apropiado. Después de que se aplique el PTF para el APAR PH36071 y se active el nivel de función 500 o superior de Db2 12, el parámetro de subsistema SUBSTR_COMPATIBILITY se establece en PREVIOUS de forma predeterminada y Db2 sigue comportándose como antes de aplicar el PTF. Si el parámetro de subsistema SUBSTR_COMPATIBILITY se establece en CURRENT, Db2 siempre impone las reglas para la función incorporada SUBSTR que se documentan en *Referencia SQL* y devuelve un código de error de SQL si no se cumplen las reglas.

Por ejemplo, si el parámetro de subsistema SUBSTR_COMPATIBILITY se establece en CURRENT, la consulta siguiente devuelve un código de error de SQL:

```
SELECT SUBSTR('ABCD', 2+1, 3) FROM SYSIBM.SYSDUMMY1;
```

Anteriormente, esta consulta ha devuelto incorrectamente el resultado 'CD '.

Antes de establecer el parámetro de subsistema SUBSTR_COMPATIBILITY en CURRENT, es posible que tenga que modificar algunas de las aplicaciones para manejar este cambio.

Acciones a emprender

En Db2 12, antes de establecer el parámetro de subsistema SUBSTR_COMPATIBILITY en CURRENT, identifique las aplicaciones que son incompatibles con este cambio iniciando un rastreo para IFCID 0376 y, a continuación, ejecutando las aplicaciones. Revise la salida de rastreo para los cambios incompatibles con el identificador 14. Corrija las aplicaciones afectadas para que sean compatibles si el parámetro de subsistema SUBSTR_COMPATIBILITY se establece en CURRENT en el futuro.

Referencia relacionada

[SUBSTR COMPATIBILITY \(parámetro del subsistemaSUBSTR_COMPATIBILITY \) \(Db2 Instalación y migración\)](#)

Las sentencias CREATE TABLESPACE y CREATE INDEX sin ninguna cláusula USING de nivel de espacio fallan si el grupo de almacenamiento especificado cuando se ha creado la base de datos que lo contiene no existe

A partir de la versión 500 de MySQL (Db2 12), Db2 registra el grupo de almacenamiento predeterminado para el espacio de tabla o índice en el catálogo de Db2 . Sin embargo, Db2 tampoco valida la existencia de un grupo de almacenamiento especificado en la cláusula STOGROUP de una sentencia CREATE DATABASE. Como resultado, una sentencia CREATE TABLESPACE o CREATE INDEX que omite la cláusula USING en el nivel de espacio de tabla o índice ahora falla con SQLCODE -204, si el grupo de almacenamiento especificado cuando se ha creado la base de datos que lo contiene no existe.

Por ejemplo, supongamos que se creó una base de datos con la siguiente declaración y no existe ningún grupo de almacenamiento con el nombre NONESUCH.

```
CREATE DATABASE MYDB0 STOGROUP NONESUCH;
```

Si emite una instrucción CREATE TABLE SPACE como el siguiente ejemplo, o una instrucción CREATE INDEX similar, que no especifica un grupo de almacenamiento existente, falla con SQLCODE -204 porque el grupo de almacenamiento NONESUCH no existe.

```
CREATE TABLESPACE TSPACE0 IN MYDB0;
```

Es decir, el resultado es similar al siguiente:

```
DSNT408I SQLCODE = -204, ERROR: NONESUCH IS AN UNDEFINED NAME  
DSNT418I SQLSTATE   = 42704 SQLSTATE RETURN CODE
```

Acciones a emprender

Si el grupo de almacenamiento especificado en una instrucción CREATE DATABASE no existe, realice una o ambas de las siguientes acciones:

- Especifique un grupo de almacenamiento existente en una cláusula USING STOGROUP de nivel de índice o espacio de tabla (una cláusula USING de nivel de partición no es suficiente para evitar el error) en cualquier instrucción CREATE TABLESPACE o CREATE INDEX que cree un espacio de tabla o índice en esa base de datos. Por ejemplo, la siguiente instrucción CREATE TABLESPACE tiene éxito si la emite en lugar del ejemplo anterior, suponiendo que existe el grupo de almacenamiento SYSDEFLT:

```
CREATE TABLESPACE TSPACE0 IN MYDB0 USING STOGROUP SYSDEFLT;
```

- Emitir una instrucción ALTER DATABASE y especificar la cláusula STOGROUP que identifica un grupo de almacenamiento. Por ejemplo, puede emitir la siguiente declaración y, a continuación, volver a intentar las declaraciones CREATE TABLESPACE o CREATE INDEX.

```
ALTER DATABASE MYDB0 STOGROUP SYSDEFLT;
```

Referencia relacionada

[CREATE TABLESPACE declaración \(Db2 SQL\)](#)

[CREATE INDEX declaración \(Db2 SQL\)](#)

[CREATE DATABASE declaración \(Db2 SQL\)](#)

[ALTER DATABASE declaración \(Db2 SQL\)](#)

Nuevo número máximo de marcadores de parámetro o variables de host en una única sentencia SQL

Db2 12 impone el número máximo de marcadores de parámetro o variables de host en una sola sentencia SQL. A partir del nivel de función 100, Db2 12 emite SQLCODE -101 para cualquier sentencia SQL que contenga más de 16.000 marcadores de parámetro o variables de host.

Acciones a emprender

Identifique y modifique cualquier sentencia SQL existente que contenga más de 16.000 marcadores de parámetro o variables de host.

Referencia relacionada

[Límites en Db2 for z/OS \(Db2 SQL\)](#)

Cambio de resultado para la sentencia SQL EXPLAIN PACKAGE

Cuando Db2 procesa la sentencia SQL EXPLAIN PACKAGE, la columna HINT_USED de PLAN_TABLE se llena con EXPLAIN PACKAGE: *copy*. El campo *copy* de la columna HINT_USED será uno de los valores siguientes:

- "CURRENT": la copia actual
- "PREVIOUS": la copia anterior
- "ORIGINAL": la copia original

Este cambio da soporte a la nueva capacidad de incorporación gradual del reenlace que ha introducido el nivel de función 505. Sin embargo, el cambio entra en vigor inmediatamente cuando se migra a Db2 12.

Acciones a emprender

Cambie la salida esperada para las consultas que hacen referencia a esta columna.

Cambio de resultado para la sentencia SQL EXPLAIN STABILIZED DYNAMIC QUERY

Cuando Db2 procesa la sentencia SQL EXPLAIN STABILIZED DYNAMIC QUERY, la columna HINT_USED de PLAN_TABLE se llena con EXPLAIN PACKAGE: *copy*. El campo *copy* de la columna HINT_USED será uno de los valores siguientes:

- "CURRENT": la copia actual
- "INVALID": la copia no válida

Este cambio da soporte a la nueva capacidad de incorporación gradual del reenlace que ha introducido el nivel de función 505. Sin embargo, el cambio entra en vigor inmediatamente cuando se migra a Db2 12.

Acciones a emprender

Cambie la salida esperada para las consultas que hacen referencia a esta columna.

Cambios en la columna DSVOLSER de la tabla de catálogo SYSCOPY

Db2 12 introduce una nueva capacidad para suprimir sólo las copias de imagen de FlashCopy si existen copias de imagen secuenciales equivalentes, para un procedimiento de copia de seguridad eficaz que utilice el espacio de disco mínimo. Como soporte de esta prestación, los valores posibles para la columna DSVOLSER en la tabla de catálogo SYSIBM.SYSCOPY han cambiado. Anteriormente, el valor de la columna DSVOLSER era una serie vacía para copias de imagen completas catalogadas, secuenciales. Algunas aplicaciones pueden suponer que si el atributo de longitud del valor de DSVOLSER es cero, la copia de imagen está catalogada. En Db2 12, esa suposición ya no es correcta. Para las copias de imagen completas catalogadas y secuenciales que se crean a partir de una copia de imagen de FlashCopy con coherencia y que también tenían unidades de trabajo no confirmadas restituidas, la columna DSVOLSER ahora contiene información de punto de comprobación de Db2.

Acciones a emprender

Modifique las aplicaciones que utilizan la columna DSVOLSER en la tabla de catálogo SYSCOPY para tolerar la información de punto de comprobación para copias de imagen completas catalogadas y secuenciales. Para obtener detalles, consulte la descripción de DSVOLSER en la [Tabla de catálogo SYSCOPY](#).

Los niveles de compatibilidad de aplicación se aplican a las sentencias de definición y control de datos

Después de la activación de nivel de función 500 o superior en Db2 12, los niveles de compatibilidad de aplicación también controlan la sintaxis, la semántica, los valores predeterminados y la validación de opciones para la mayoría de las sentencias de definición de datos y las sentencias de control de datos. Las sentencias de definición de datos (a veces abreviadas como DDL) incluyen varias sentencias CREATE y ALTER. Las sentencias de control de datos (a veces abreviadas como DCL) incluyen varias sentencias GRANT y REVOKE. Sólo los niveles de compatibilidad de aplicación V12R1M509 y superior controlan el comportamiento de cualquier definición de datos o sentencias de control de datos.

La opción de enlace APPLCOMPAT para un paquete se aplica a la mayoría de las sentencias de definición y control de datos de SQL estático. El registro especial CURRENT APPLICATION COMPATIBILITY se aplica a la mayoría de sentencias de definición de datos y control de datos de SQL dinámico.

Para la regeneración implícita de un objeto, se utiliza el nivel de compatibilidad de aplicación que estaba en vigor para la sentencia CREATE o ALTER anterior para ese objeto.

Para la materialización de los cambios de definición de datos pendientes, se utiliza el nivel de compatibilidad de aplicación de la sentencia ALTER pendiente.

Puede especificar la cláusula USING APPLICATION COMPATIBILITY de determinadas sentencias ALTER para volver a generar un objeto con un nivel de compatibilidad de aplicación específico.

Conceptos relacionados

[Niveles de función y niveles relacionados en Db2 12 \(Db2 for z/OS ¿Qué hay de nuevo?\)](#)

[Niveles de compatibilidad de aplicaciones en Db2 12](#)

El nivel de compatibilidad de sus aplicaciones controla la adopción y el uso de nuevas capacidades y mejoras, y a veces reduce el impacto de los cambios incompatibles. La ventaja es que puede completar el proceso de migración de Db2 12 sin necesidad de actualizar sus aplicaciones inmediatamente.

Referencia relacionada

[APPLCOMPAT opción bind \(comandos Db2 \)](#)

[COMPATIBILIDAD CON LA APLICACIÓN ACTUAL registro especial \(Db2 SQL\)](#)

Reenlace automático de planes y paquetes creados antes de DB2 10

Los *enlaces automáticos* (también llamados "autoenlaces") relacionados con la migración se produce en Db2 12 porque no puede utilizar estructuras de tiempo de ejecución de un plan o paquete que se enlazó por última vez en un release anterior a DB2 10. Los planes y paquetes que estaban enlazados en Db2 11 pueden ejecutarse en Db2 12, sin el riesgo de autoenlaces relacionados con la migración. Sin embargo, los planes y paquetes que están enlazados en Db2 12 no se pueden ejecutar en miembros de Db2 11 sin un autoenlace en Db2 11.

Si especifica YES o COEXIST para el parámetro de subsistema ABIND, Db2 12 vuelve a enlazar automáticamente los planes y paquetes que estaban enlazados antes de DB2 10 cuando Db2 ejecuta los paquetes. El resultado del enlace automático crea un nuevo paquete y descarta la copia actual. Db2 no mueve la copia actual a la copia anterior u original porque Db2 12 no puede utilizarla. Si se produce una regresión, REBIND SWITCH PREVIOUS y REBIND SWITCH ORIGINAL no están disponibles.

Si especifica NO para el parámetro de subsistema ABIND, se devuelven los SQLCODE negativos para cada intento de ejecutar un paquete o plan que estaba enlazado antes de DB2 10. SQLCODE -908, SQLSTATE 23510 se devuelve para los paquetes y SQLCODE -923, SQLSTATE 57015 se devuelve para los planes hasta que se reenlanzan en Db2 12.

Acciones a emprender

Al prepararse para la migración a Db2 12 en Db2 11, puede reducir el cambio y el riesgo de los paquetes sujetos a enlaces automáticos en Db2 12. Para ello, vuelva a enlazar todos los paquetes que se enlazaron por última vez antes de DB2 10 en Db2 11, antes de migrar a Db2 12. Para obtener más información sobre los impactos que pueden tener los reenlaces automáticos relacionados con la migración en el entorno Db2 y las acciones que puede realizar para evitarlos, consulte [Vuelva a vincular los planes y paquetes antiguos en Db2 11 para evitar vinculaciones automáticas perjudiciales en Db2 12 \(Instalación y migración Db2 \)](#).

Referencia relacionada

[AUTO BIND \(parámetro del subsistemaABIND \) \(Db2 Instalación y migración\)](#)

Información relacionada

[-908 \(Db2 Codes\)](#)

[-923 \(Db2 Codes\)](#)

Soporte de opción de enlace KEEPNAMIC(YES) para ROLLBACK

En Db2 12, cuando el valor de APPLCOMPAT es V12R1M500, la opción de enlace KEEPDYNAMIC(YES) afecta a ambas sentencias, COMMIT y ROLLBACK. Con KEEPDYNAMIC(YES), las sentencias de SQL dinámico del paquete se retienen después de COMMIT o ROLLBACK, y dichas sentencias se pueden volver a ejecutar sin otra PREPARE.

Antes de Db2 12, la opción de enlace KEEPDYNAMIC(YES) sólo se aplica a sentencias COMMIT. Después de una sentencia ROLLBACK, se necesitaba otra PREPARE para que se pudieran ejecutar las sentencias de SQL dinámico. Esta situación también es verdadera en Db2 12 si la compatibilidad de la aplicación se establece en V11R1 o anterior.

En Db2 12, cuando el valor de APPLCOMPAT es V12R1M500 o superior, después de emitir una sentencia ROLLBACK, el comportamiento es distinto que en las versiones anteriores:

- Una sentencia OPEN sin una sentencia PREPARE anterior no recibe un SQLCODE -514.
- Una sentencia EXECUTE sin una sentencia PREPARE anterior no recibe un SQLCODE -518.

Una aplicación que se ha escrito en Db2 11 y que estaba enlazada con KEEPNAMIC(YES) era necesaria para preparar de nuevo la sentencia de SQL dinámico después de que se haya emitido ROLLBACK. En Db2 12 cuando la compatibilidad de la aplicación se establece en V12R1M500 o superior, las sentencias PREPARE adicionales no son necesarias.

Acciones a emprender

Al migrar a Db2 12, revise los paquetes que utilizan la opción de enlace KEEPDYNAMIC(YES). Puede hacer que los programas de SQL dinámico enlazados con KEEPNAMIC(YES) se ejecuten de forma más eficaz eliminando sentencias PREPARE que vuelven a preparar sentencias SQL, tras la ejecución de sentencias ROLLBACK. No realice esta acción hasta que esté seguro de que ya no necesita ejecutar los programas en Db2 11 o anteriores. Después de migrar a Db2 12, si realiza esta acción (para eliminar sentencias PREPARE después de ROLLBACK), los programas no funcionarán correctamente si posteriormente establece la compatibilidad de la aplicación en V11R1 o anterior.

Referencia relacionada

[KEEPDYNAMIC opción bind \(comandos Db2 \)](#)

Las alteraciones en la compresión de índices son cambios pendientes para los espacios de tablas universales

Cuando el nivel de compatibilidad de aplicación es V12R1M500 o superior, la modificación para utilizar la compresión de índices para índices en espacios de tablas universales es un cambio pendiente que coloca el índice en un estado pendiente de aviso de REORG (AREOR). Los programas de utilidad LOAD REPLACE y REBUILD INDEX ya no materializan el cambio. Debe utilizar un REORG en línea para materializar el nuevo valor del atributo COMPRESS en la sentencia ALTER INDEX.

En releases anteriores a Db2 12, cualquier alteración para utilizar la compresión de índice colocaba el índice en el estado pendiente de REBUILD (RBDP). Es necesario utilizar el programa de utilidad REBUILD INDEX para volver a crear el índice o utilizar el programa de utilidad REORG para reorganizar el espacio de tabla que corresponde al índice.

Acciones a emprender

Para índices en espacios de tabla universales, utilice una REORG en línea para materializar el nuevo valor para el atributo COMPRESS en la sentencia ALTER INDEX.

Tareas relacionadas

[Compresión de índices \(Db2 Performance\)](#)

Referencia relacionada

[ALTER INDEX declaración \(Db2 SQL\)](#)

Tipos de datos de argumentos de salida de una llamada de procedimiento almacenado en una aplicación Java

En nivel de función 500 o superior con la compatibilidad de aplicaciones establecida en V11R1, cuando una aplicación Java™ que utiliza IBM Data Server Driver for JDBC and SQLJ llama a un procedimiento

almacenado, los tipos de datos de los argumentos de salida de procedimiento almacenado coinciden con los tipos de datos de los parámetros de la definición de procedimiento almacenado.

Explicación

Antes de DB2 10, si un cliente Java llamaba a un procedimiento almacenado Db2 for z/OS, los tipos de datos de los argumentos de salida coincidían con los tipos de datos de los argumentos de sentencia CALL correspondientes. A partir de DB2 10, los tipos de datos de los argumentos de salida coinciden con los tipos de datos de los parámetros de la definición de procedimiento almacenado.

En Db2 12, cuando la compatibilidad de aplicaciones se establece en V10R1, puede establecer el parámetro de subsistema DDF_COMPATIBILITY en SP_PARMS_JV para mantener el comportamiento que existía antes de DB2 10. Sin embargo, cuando la compatibilidad de la aplicación se establece en V11R1 o V12R1M100, o en V12R1M500 o superior, SP_PARMS_JV ya no está soportado.

En Db2 12 con la compatibilidad de aplicaciones establecida en V11R1 o V12R1M100, o en V12R1M500 o superior, si la versión de IBM Data Server Driver for JDBC and SQLJ es inferior a 3.63 o 4.13, se puede generar una `java.lang.ClassCastException` cuando se recupera un valor de argumento de salida.

Acciones a emprender

Efectúe una de las acciones siguientes:

- Actualice IBM Data Server Driver for JDBC and SQLJ a la versión 3.63 o 4.13 o posterior.
- Modifique los tipos de datos en las llamadas de método `CallableStatement.registerOutParameter` para que coincidan con los tipos de datos de parámetros en las definiciones de procedimiento almacenado. Puede establecer la compatibilidad de aplicaciones en V10R1 y ejecutar un rastreo para el IFCID 0376 para identificar las aplicaciones afectadas. Los registros de rastreo para esas aplicaciones tienen un valor de campo QW0376FN de 8.

Conceptos relacionados

[Niveles de compatibilidad de aplicaciones en Db2 12](#)

El nivel de compatibilidad de sus aplicaciones controla la adopción y el uso de nuevas capacidades y mejoras, y a veces reduce el impacto de los cambios incompatibles. La ventaja es que puede completar el proceso de migración de Db2 12 sin necesidad de actualizar sus aplicaciones inmediatamente.

Referencia relacionada

[Parámetros de subsistema que no están en los paneles de instalación \(Db2 Installation and Migration\)](#)

Sentencias SELECT INTO con UNION o UNION ALL

Una UNION o UNION ALL no está permitida en la cláusula from más externa de una sentencia SELECT INTO. Sin embargo, los releases anteriores a Db2 12 toleran inadvertidamente las sentencias SQL que contienen esta sintaxis no válida.

El comportamiento está controlado por el parámetro de subsistema DISALLOW_SEL_INTO_UNION. En Db2 11, el valor predeterminado tolera la sintaxis no válida. En Db2 12, el valor predeterminado no permite la sintaxis no válida.

Una aplicación que utiliza la sintaxis SQL no válida falla en BIND o REBIND con SQLCODE -109.

Acciones a emprender

Identifique los paquetes que utilicen UNION o UNION ALL en la cláusula from de una sentencia SELECT INTO y corríjalos según sea necesario. Puede especificar temporalmente que Db2 continúe tolerando la sintaxis no válida NO para el parámetro de subsistema DISALLOW_SEL_INTO_UNION. Sin embargo, este parámetro de subsistema está en desuso y se espera que se elimine en el futuro.

Puede identificar los paquetes afectados mientras DISALLOW_SEL_INTO_UNION está establecido en NO mediante el enlace de paquetes sospechosos en un ID de colección ficticia con EXPLAIN (ONLY) y

supervisando los registros de IFCID 0376. Los registros de rastreo para las aplicaciones afectadas tienen un valor de campo QW0376FN de 11.

Utilice el procedimiento siguiente:

1. Emite la siguiente sentencia SQL para generar una lista de mandatos BIND.

```
SELECT 'BIND PACKAGE(DUMMYCOL) COPY(' ||  
       COLLID || '.' || NAME || ') ' ||  
       CASE WHEN(VERSION <> '')  
              THEN 'COPYVER(' || VERSION || ') '  
              ELSE '' END ||  
       'EXPLAIN(ONLY)'  
  FROM SYSIBM.SYSPACKSTMT  
 WHERE STATEMENT LIKE '%SELECT%INTO%UNION%'  
   OR STATEMENT LIKE '%SELECT%UNION%INTO%';
```

La sentencia genera una salida similar al submandato BIND siguiente:

```
BIND PACKAGE(DUMMYCOL) COPY(DSN_DEFAULT_COLLID_PLAY01.PLAY01) EXPLAIN(ONLY)
```

2. Copie los resultados de la sentencia SELECT en un trabajo de enlace. Si los submandatos BIND tienen más de 72 bytes, es necesario dar formato.
3. Inicie y recopile un rastreo para el IFCID 0376.
4. Ejecute el trabajo de enlace que ha creado.
5. Detenga el rastreo de IFCID 0376 y analice la salida.

Referencia relacionada

[DISALLOW_SEL_INTO_UNION en macro DSN6SPRM \(Db2 Instalación y migración\)](#)

Información relacionada

[-109 \(Db2 Codes\)](#)

Cambiar en SQLCODE cuando el resultado de la función incorporada POWER está fuera de rango

Después de la activación de nivel de función 500 o superior en Db2 12, el SQLCODE que se devuelve cuando el resultado de la función incorporada POWER está fuera de rango se cambia en algunos casos.

Anteriormente, cuando Db2 ejecutaba la función incorporada POWER y el resultado era un tipo de datos DOUBLE que estaba fuera de rango, Db2 devolvía SQLCODE -802. En Db2 12 con nivel de función 500 o superior activado, se devuelve SQLCODE +802.

Por ejemplo, la consulta siguiente devuelve SQLCODE +802:

```
SELECT POWER(DOUBLE(2.0E38), DOUBLE(2.0))  
  FROM SYSIBM.SYSDUMMY1;
```

Las invocaciones de la función POWER que tienen argumentos DOUBLE y devuelven resultados fuera de rango devuelven SQLCODE +802 en lugar de SQLCODE -802.

Acciones a emprender

En Db2 12, antes de que se active nivel de función 500 o superior, identifique las aplicaciones con esta incompatibilidad iniciando un rastreo para el IFCID 0376 y, a continuación, ejecutando las aplicaciones. Revise la salida de rastreo para los cambios incompatibles con el identificador 1201. Ajuste el proceso de errores para tener en cuenta el cambio en el SQLCODE devuelto de error a aviso.

Tareas relacionadas

[Gestión de incompatibilidades de aplicaciones](#)

Antes de mover una aplicación a un nuevo nivel de compatibilidad de aplicaciones, necesita encontrar incompatibilidades de aplicaciones, ajustar las aplicaciones a esas incompatibilidades y verificar que las incompatibilidades ya no existen.

Referencia relacionada

[Función escalar POWER o POW \(Db2 SQL\)](#)

Funciones CHAR9 y VARCHAR9 para la compatibilidad con el formato de serie anterior a DB2 10 de datos decimales

DB2 10 ha cambiado el formato de datos decimales por las funciones incorporadas CHAR y VARCHAR y las especificaciones CAST con un tipo de resultado CHAR o VARCHAR. En Db2 12 puede utilizar funciones alternativas para la compatibilidad con aplicaciones que requieren una salida de serie decimal en los formatos anteriores a DB2 10:

- [CHAR9 función escalar \(Db2 SQL\)](#)
- [VARCHAR9 función escalar \(Db2 SQL\)](#)

Importante: Para aplicaciones portátiles que puedan ejecutarse en plataformas distintas de Db2 for z/OS, no utilice las funciones CHAR9 y VARCHAR9. Otros productos de la familia Db2 no soportan estas funciones.

Acciones a emprender

Revise el valor del parámetro de subsistema BIF_COMPATIBILITY. Si el valor no es CURRENT y tiene aplicaciones que requieren una salida de serie decimal en el formato anterior a DB2 10, puede reescribir sentencias de SQL para utilizar las funciones CHAR9 y VARCHAR9 en su lugar. Este enfoque permite el desarrollo de nuevas aplicaciones que pueden aceptar el formato de serie actual de datos decimales.

Para modificar las aplicaciones para aprovechar las funciones CHAR9, VARCHAR9:

1. Utilice un rastreo IFCID 0376 para identificar aplicaciones que dependen de los formatos anteriores a DB2 10.
2. Vuelva a escribir las sentencias SQL en las aplicaciones identificadas para utilizar las funciones CHAR9 y VARCHAR9 en lugar de las funciones CHAR y VARCHAR.
3. Establezca el valor de BIF_COMPATIBILITY en CURRENT.

Referencia relacionada

[BIF COMPATIBILITY \(parámetro del subsistemaBIF_COMPATIBILITY \) \(Db2 Instalación y migración\)](#)

Parámetro de subsistema BIF_COMPATIBILITY y esquemas SQL para la compatibilidad con el formato de serie anterior a DB2 10 de datos decimales

DB2 10 ha cambiado el formato de datos decimales por las funciones incorporadas CHAR y VARCHAR y las especificaciones CAST con un tipo de resultado CHAR o VARCHAR. Puede alterar temporalmente estos cambios en un nivel de subsistema estableciendo el parámetro de subsistema BIF_COMPATIBILITY en uno de los valores anteriores a DB2 10. También puede alterar temporalmente estos cambios en un nivel de aplicación añadiendo el esquema SYSCOMPAT_V9 a la parte frontal de la opción de enlace PATH o el registro especial CURRENT PATH. Este último enfoque funciona para las funciones CHAR y VARCHAR y no afecta a las especificaciones CAST.

El enfoque recomendado es modificar las aplicaciones para manejar el DB2 10 y el comportamiento posterior para estas funciones, tal como se describe en los pasos siguientes.

Acciones a emprender

Para modificar las aplicaciones para manejar el DB2 10 y el comportamiento posterior para CHAR, VARCHAR y CAST:

1. Identifique las aplicaciones que deben modificarse para manejar este cambio. Ejecute un rastreo para el IFCID 0376 para identificar las aplicaciones afectadas.
2. Asegúrese de que el parámetro de subsistema BIF_COMPATIBILITY está establecido en V9_DECIMAL VARCHAR.

Para manejar el cambio sólo para la función CHAR, puede establecer BIF_COMPATIBILITY en V9 y completar los pasos siguientes para la función CHAR.

3. Cambie las aplicaciones afectadas para manejar el DB2 10 y el comportamiento posterior de CHAR y VARCHAR, incluidos los procedimientos almacenados, las funciones definidas por el usuario no incorporadas y los paquetes desencadenantes. Vuelva a escribir las especificaciones CAST afectadas con la función CHAR o VARCHAR adecuada y una CAST en la longitud correcta, si es necesario.
4. Vuelva a enlazar y prepare los paquetes con la opción de reenlace PATH(SYSCURRENT, SYSIBM). La colocación del esquema SYSCURRENT al principio de la vía de acceso de SQL hace que se utilice el DB2 10 y el comportamiento posterior para las funciones incorporadas CHAR y VARCHAR.
Repita este paso para los procedimientos almacenados nativos (SQLPL) y las funciones escalares de SQL no en línea.
5. Para las vistas que hacen referencia a estas conversiones o funciones incorporadas, determine si es necesario cambiar la vista para que tenga la salida esperada. Descarte y vuelva a crear las vistas con la opción de reenlace PATH(SYSCURRENT, SYSIBM), sólo si es necesario. Vuelva a enlazar cualquier aplicación que haga referencia a las vistas con la opción PATH(SYSCURRENT, SYSIBM) para utilizar DB2 10 y las funciones incorporadas posteriores CHAR y VARCHAR. Repita este paso para las funciones escalares de SQL en línea.
6. Para tablas de consulta materializadas o índices en expresiones que hacen referencia a estas conversiones o funciones incorporadas, descarte y vuelva a crear las tablas de consulta materializada o los índices en expresiones con la opción de reenlace PATH(SYSCURRENT, SYSIBM). Emite la sentencia REFRESH TABLE para las tablas de consulta materializada. Vuelva a enlazar las aplicaciones que hagan referencia a las tablas de consulta materializada o los índices en expresiones con la opción PATH(SYSCURRENT, SYSIBM) para utilizar DB2 10 y las funciones incorporadas posteriores CHAR y VARCHAR.
7. Cambie el valor del parámetro de subsistema BIF_COMPATIBILITY a CURRENT. Cuando el valor del parámetro de subsistema es CURRENT, las nuevas aplicaciones, los reenlaces y las sentencias CREATE utilizan DB2 10 y el comportamiento posterior CHAR, VARCHAR y CAST.

Las tablas de consulta materializada y los índices basados en expresiones utilizan el comportamiento de CHAR, VARCHAR y CAST que se especifica durante su creación. Si una sentencia de referencia tiene un comportamiento distinto especificado por el parámetro BIF_COMPATIBILITY o una vía de acceso distinta, no se utiliza la tabla de consulta materializada o el índice basado en expresiones.

Referencia relacionada

[BIF COMPATIBILITY \(parámetro del subsistemaBIF_COMPATIBILITY \) \(Db2 Instalación y migración\)](#)

La entrada de serie combinada EBCDIC para las funciones incorporadas RTRIM, TRIM, LTRIM y STRIP debe ser válida

A partir del nivel de compatibilidad de aplicaciones V12R1M500 o superior, Db2 12 aplica más comprobaciones de validación para la entrada de cadenas mixtas EBCDIC a las funciones integradas RTRIM, TRIM, LTRIM y STRIP.

Por lo general, Db2 ha requerido datos de cadena mixta EBCDIC válidos para la entrada en estas funciones desde la versión 10, pero Db2 12 ahora detecta más casos que las versiones anteriores.

Con la nueva comprobación de validación, cuando Db2 12 realiza una operación de recorte y el argumento *de expresión de cadena* de la función incorporada RTRIM, TRIM, LTRIM o STRIP contiene datos mixtos EBCDIC no válidos, Db2 emite SQLCODE -171.

Acciones a emprender

Comprueba si los datos mixtos EBCDIC especificados para *el argumento de expresión de cadena* de una función incorporada RTRIM, TRIM, LTRIM o STRIP son válidos y resuelve los datos no válidos.

En datos mixtos válidos, las partes de doble byte de las series de entrada empiezan por X'OE' (carácter de desplazamiento desde teclado estándar), finalizan con X'OF' (carácter de desplazamiento a teclado

estándar) y tienen un número par de bytes entre los caracteres X'OE' y X'OF'. Los datos que no cumplen estos criterios son datos mixtos no válidos. Cuando se especifican datos mixtos no válidos para el argumento *expresión-serie* de una función incorporada RTRIM, TRIM, LTRIM o STRIP, se devuelve SQLCODE -171 en algunos casos. Si Db2 recorta los caracteres especificados antes de que alcance una parte no válida de una serie mixta, la operación de recorte se realiza satisfactoriamente.

Establecer una compatibilidad de aplicación de V12R1M100 o inferior (si se aplica PTF para APAR PH25783) podría evitar el error mientras resuelve los datos no válidos. En los niveles inferiores de compatibilidad de la aplicación, Db2 12 intenta tolerar los datos mixtos no válidos identificados por la nueva comprobación de validación, y permite que la operación de recorte se complete si es posible. Sin embargo, si Db2 sigue sin poder realizar la operación de recorte, se emite SQLCODE -171.

El número máximo de funciones escalares externas definidas por el usuario que se ejecutan en un hilo de ejecución de aplicaciones (Db2) ya no es ilimitado (APAR PH44833)

A partir del nivel de compatibilidad de aplicaciones V12R1M100 (si se aplica el PTF para APAR PH44833), Db2 12 introduce el parámetro del subsistema MAX_UDF. MAX_UDF controla el número máximo de funciones escalares externas definidas por el usuario que pueden ejecutarse simultáneamente en un hilo de ejecución (Db2). El valor máximo de MAX_UDF es 99999. Antes de la introducción de MAX_UDF, el número máximo de funciones escalares externas definidas por el usuario que podían ejecutarse simultáneamente en un hilo de ejecución (Db2) era ilimitado.

Acciones a emprender

Si una aplicación contiene sentencias SQL que invocan funciones escalares externas definidas por el usuario, y una de esas sentencias SQL es rechazada con SQLCODE -904 y código de razón 00E70082, aumente el valor del parámetro del subsistema MAX_UDF o cambie la aplicación para que ejecute menos funciones simultáneamente en un hilo de ejecución (Db2).

Referencia relacionada

[MAX UDFS \(parámetro del subsistemaMAX_UDF \) \(Db2 Instalación y migración\)](#)

Información relacionada

[00E70082 \(Db2 Codes\)](#)

Palabras reservadas de SQL



Db2 12 presenta varias nuevas palabras reservadas de SQL, que se listan en [Palabras reservadas en Db2 for z/OS \(Db2 SQL\)](#).

En algunos casos, el uso de estas palabras reservadas puede causar una incompatibilidad antes de que se active la nueva función en Db2 12, independientemente del valor del distintivo APPLCOMPAT.

Acciones a emprender

Compruebe si sus aplicaciones utilizan las siguientes palabras reservadas de SQL:

- [FL 504 CURRENT_SERVER](#)
- [FL 504 CURRENT_TIMEZONE](#)
- LIMIT
- OFFSET

Ajuste las aplicaciones que utilicen estas palabras cambiando la palabra reservada por un identificador delimitado o utilizando una palabra que no esté reservada en Db2 12. La lista completa puede consultarse en [Palabras reservadas en Db2 for z/OS \(Db2 SQL\)](#).

Funciones incorporadas y funciones definidas por el usuario existentes

Para funciones incorporadas y definidas por el usuario, la combinación del nombre de función y la lista de parámetros forman la *firma* que Db2 utiliza para identificar la función. Si las firmas de las funciones integradas nuevas o modificadas en Db2 12 coinciden con las firmas de las funciones definidas por el usuario existentes, las aplicaciones con referencias no cualificadas a las funciones definidas por el usuario existentes podrían empezar a invocar las funciones integradas nuevas o modificadas en lugar de las funciones definidas por el usuario. Db2 12 presenta los siguientes cambios en las funciones integradas:

Db2 12 presenta o cambia las siguientes funciones incorporadas.

Información importante sobre las funciones definidas por el usuario existentes: Cuando un nuevo nivel de compatibilidad de aplicación introduce una función incorporada nueva o modificada que tiene el mismo nombre y firma que una función definida por el usuario existente, las referencias no calificadas a la función definida por el usuario pueden resolverse incorrectamente. Las aplicaciones que tienen referencias no calificadas a la función definida por el usuario pueden fallar. Para evitar esta situación, modifique las aplicaciones para calificar explícitamente las referencias a funciones definidas por el usuario con el mismo nombre y firma que las funciones incorporadas nuevas o modificadas.

Nivel de APPLCOMPAT	Nombre de función	Modificación introducida	¿Cambio incompatible?
V12R1M507	Varios	<p>Las funciones siguientes están soportadas recientemente en Db2 for z/OS como expresiones de sólo paso a través, que se pasan a IBM Db2 Analytics Accelerator for z/OS.</p> <ul style="list-style-type: none"> • Función escalar ADD_DAYS (Db2 SQL) • BTRIM función escalar (Db2 SQL) • Función escalar DÍAS_ENTRE (Db2 SQL) • Función escalar NEXT_MONTH (SQL Db2) • REGR_AVGX • REGR_AVGY • REGR_COUNT • REGR_INTERCEPT o REGR_ICPT • REGR_R2 • REGR_SLOPE • REGR_SXX • REGR_SXY • REGR_SYY • Función escalar ROUND_TIMESTAMP (Db2 SQL) si se invoca con una expresión de fecha 	Nee
V12R1M506	HASH función escalar (Db2 SQL)	Nueva función incorporada.	Nee

Nivel de APPLCOMPA T	Nombre de función	Modificación introducida	¿Cambio incompatibl e?
V12R1M506	Función escalar CHARACTER_LENGTH o CHAR_LENGTH (Db2 SQL)	CHAR_LENGTH ahora está soportado como un nombre de función alternativo.	Nee
V12R1M506	CLOB o función escalar TO_CLOB (Db2 SQL)	Ahora se da soporte a TO_CLOB como nombre de función alternativo.	Nee
V12R1M506	Función agregada COVAR_POP o COVARIANCE o COVAR (Db2 SQL)	Ahora se da soporte a COVAR_POP como un nombre de función alternativo.	Nee
V12R1M506	LEFT o función escalar STRLEFT (Db2 SQL)	STRLEFT ahora está soportado como un nombre de función alternativo.	Nee
V12R1M506	Función escalar POWER o POW (Db2 SQL)	Ahora se da soporte a POW como un nombre de función alternativo.	Nee
V12R1M506	POSSTR o función escalar STRPOS (Db2 SQL)	Ahora se da soporte a STRPOS como un nombre de función alternativo.	Nee
V12R1M506	RANDOM o función escalar RAND (Db2 SQL)	Ahora se da soporte a RANDOM como nombre de función alternativo.	Nee
V12R1M506	RIGHT o función escalar STRRIGHT (Db2 SQL)	Ahora se da soporte a STRRIGHT como un nombre de función alternativo.	Nee
V12R1M506	Función escalar TIMESTAMP_FORMAT o TO_TIMESTAMP (Db2 SQL)	Ahora se da soporte a TO_TIMESTAMP como nombre de función alternativo.	Nee
V12R1M505	DECRYPT_DATAKEY_INTEG ER, DECRYPT_DATAKEY_BIGINT , DECRYPT_DATAKEY_DECIMAL, DECRYPT_DATAKEY_VARCHAR, DECRYPT_DATAKEY_CLOB, DECRYPT_DATAKEY_VARGRAPHIC, DECRYPT_DATAKEY_DBCL OBy DECRYPT_DATAKEY_BIT	Nuevas funciones incorporadas.	Nee
V12R1M505	ENCRYPT_DATAKEY	Nueva función incorporada.	Nee
V12R1M504	Varios	Las funciones siguientes están soportadas recientemente en Db2 for z/OS como expresiones de sólo paso a través, que se pasan a IBM Db2 Analytics Accelerator for z/OS. <ul style="list-style-type: none"> • CUME_DIST • Función agregada CUME_DIST (Db2 SQL)FIRST_VALUE • LAG • LAST_VALUE 	Nee

Nivel de APPLCOMPAT	Nombre de función	Modificación introducida	¿Cambio incompatible?
		<ul style="list-style-type: none"> • LEAD • NTH_VALUE • NTILE • PERCENT_RANK • Función agregada PERCENT_RANK (Db2 SQL) • RATIO_TO_REPORT • Función escalar REGEXP_COUNT (Db2 SQL) • Función escalar REGEXP_INSTR (Db2 SQL) • Función escalar REGEXP_LIKE (Db2 SQL) • Función escalar REGEXP_REPLACE (Db2 SQL) • Función escalar REGEXP_SUBSTR (Db2 SQL) 	
V12R1M502	GRAPHIC función escalar (Db2 SQL)	El primer argumento ahora acepta tipos de datos numéricos, incluidos SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE, FLOAT y DECFLOAT.	Nee
V12R1M502	VARGRAPHIC función escalar (Db2 SQL)	El primer argumento acepta tipos de datos numéricos, incluidos SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE, FLOAT y DECFLOAT.	Nee
V12R1M501	LISTAGG función agregada (Db2 SQL)	Nueva función incorporada.	Nee
V12R1M500	Función agregada ARRAY_AGG (SQL Db2)	Función incorporada recién soportada cuando se utiliza para la agregación de matrices asociativas.	Nee
V12R1M500	GENERATE_UNIQUE_BINARY	Nueva función incorporada.	Nee
V12R1M500	HASH_CRC32, HASH_MD5, HASH_SHA1 y HASH_SHA256	Nuevas funciones incorporadas.	Nee
V12R1M500	LOWER función escalar (Db2 SQL)	Ahora se pueden especificar las siguientes configuraciones regionales: <ul style="list-style-type: none"> • UNI_60 • UNI_90 	
V12R1M500	PERCENTILE_CONT	Nueva función incorporada.	Nee
V12R1M500	PERCENTILE_DISC	Nueva función incorporada.	Nee
V12R1M500	TRANSLATE función escalar (Db2 SQL)	Ahora se pueden especificar las siguientes configuraciones regionales:	Nee

Nivel de APPLCOMPA T	Nombre de función	Modificación introducida	¿Cambio incompatibl e?
		<ul style="list-style-type: none"> • UNI_60 • UNI_90 	
V12R1M500	UPPER función escalar (Db2 SQL)	Ahora se pueden especificar las siguientes configuraciones regionales: <ul style="list-style-type: none"> • UNI_60 • UNI_90 	Nee
V12R1M500	WRAP función escalar (Db2 SQL)	Nueva función incorporada.	Nee
V12R1M100	Función de tabla BLOCKING_THREADS (Db2 SQL)	Nueva función incorporada.	



Acciones a emprender

Para continuar ejecutando una función definida por el usuario con el mismo nombre o firma que una nueva función o firma incorporada, califique el nombre de la función definida por el usuario existente en la aplicación. Para obtener más información sobre Db2, resuelve las referencias calificadas y no calificadas en las funciones, consulte [Resolución de función \(Db2 SQL\)](#).

Cambios de SQLCODE

Es posible que algunos números SQLCODE y texto de mensaje hayan cambiado en Db2 12. Además, las condiciones bajo las que se emiten algunos SQLCODE pueden haber cambiado. Para obtener más información, consulte [Códigos nuevos, modificados y suprimidos \(Db2 Codes\)](#).



Determinar el valor de cualquier opción de procesamiento SQL que afecte al diseño de su programa

Cuando procesa instrucciones SQL en un programa de aplicación, puede especificar opciones que describen las características básicas del programa. También puede indicar cómo desea que se vean los listados de salida. Aunque la mayoría de estas opciones no afectan a la forma en que diseña o codifica el programa, algunas sí lo hacen.

Acerca de esta tarea

Las opciones de procesamiento SQL especifican características del programa como los siguientes elementos:

- El idioma de origen en el que está escrito el programa
- La máxima precisión de los números decimales en el programa
- ¿Cuántas líneas hay en una página del listado del precompilador?

En muchos casos, es posible que desee aceptar el valor predeterminado proporcionado.

Procedimiento

Revise la lista de opciones de procesamiento SQL y decida los valores de las opciones que afecten a la forma en que escribe su programa.

Por ejemplo, antes de empezar a programar, debe saber si está utilizando NOFOR o STDSQL(YES).

Tareas relacionadas

Procesamiento de instrucciones SQL para la preparación de programas

El primer paso en la preparación de una aplicación de SQL para su ejecución es procesar las sentencias SQL en el programa. Para procesar las declaraciones, utilice el Db2 coprocesor o el Db2 precompilador. Durante la realización de este paso, las sentencias de SQL se sustituyen por llamadas a módulos de interfaz de lenguaje de Db2 y se crea un módulo de solicitud de base de datos (DBRM).

Referencia relacionada

Descripciones de opciones de proceso de SQL

Puede especificar cualquier opción de proceso de SQL tanto si utiliza el precompilador de Db2 como si utiliza el coprocesador de Db2. Sin embargo, es probable que el coprocesador de Db2 omita ciertas opciones ya que existen opciones del compilador del lenguaje principal que proporcionan la misma información.

Cambios que invalidan paquetes

Los cambios realizados en los objetos de programa o de base de datos pueden invalidar paquetes.

Un cambio en su programa probablemente invalide uno o más de sus paquetes. Para algunos cambios, debe vincular un nuevo objeto. Para otros, basta con rehacer el encuadernado. Un paquete también puede dejar de ser válido por razones que no dependen de las operaciones de su programa. Por ejemplo, cuando se elimina un índice que se utiliza en una ruta de acceso por una de sus consultas, un paquete puede dejar de ser válido.

Db2 podría volver a vincular automáticamente los paquetes no válidos la próxima vez que se ejecute el paquete. Para obtener más información, consulte [“Revinculación automática”](#) en la página 957.

Cómo marca Db2 los paquetes no válidos

En la mayoría de los casos, Db2 marca un paquete que debe ser automáticamente rebotado como *inválido* estableciendo VALID='N' en las tablas de catálogo SYSIBM.SYSPLAN y SYSIBM.SYSPACKAGE.

Acciones que provocan que Db2 invalide paquetes

Db2 marca los paquetes como no válidos cuando dependen del objeto de destino y, a veces, de objetos relacionados que se ven afectados por efectos en cascada de las acciones que se enumeran en la siguiente tabla.

Objeto u operación	Cambios que invalidan paquetes
Tablas	<ul style="list-style-type: none">Añadir una columna HORA, MARCA DE HORA o FECHA cuando el valor predeterminado para las filas añadidas es FECHA ACTUAL, HORA ACTUAL, MARCA DE HORA ACTUAL (<i>p</i>) SIN ZONA HORARIA o MARCA DE HORA ACTUAL (<i>p</i>) CON ZONA HORARIA, respectivamenteAñadir una restricción con una regla de eliminación de SET NULL o CASCADE. Los paquetes que dependen de tablas que se eliminan en cascada a la tabla principal modificada también se invalidan.Añadir una etiqueta de seguridadModificar una columna

Objeto u operación	Cambios que invalidan paquetes
	<ul style="list-style-type: none"> • Cambiar el nombre de una columna (se aplican efectos en cascada). Consulte “Efectos en cascada en paquetes de renombrar una columna” en la página 19) • Alterar una columna de la tabla de tal manera que una vista no pueda regenerarse • Alteración del atributo AUDIT. • Soltar una columna. Para los cambios de definición pendientes, la invalidación del paquete se produce cuando el cambio de definición pendiente se aplica a la tabla. • Modificar la organización hash o eliminar la organización hash • Añadir o eliminar un periodo BUSINESS_TIME para el control de versiones temporal • Habilitar o deshabilitar el archivado transparente • Añadir, modificar o eliminar una definición de tabla de consulta materializada (MQT) • Eliminar una tabla clonada • Activación o desactivación del control de acceso a nivel de fila para una tabla • Activar el control de acceso a nivel de columna si la tabla tiene una columna habilitada, o desactivar el control de acceso a nivel de columna • Para las tablas temporales creadas, añadir una columna
Espacios de tabla	<ul style="list-style-type: none"> • Cambiar el atributo CCSID de SBCS • Aumentar el atributo MAXPARTITIONS • Cambiar el atributo SEGSIZE para convertir el espacio de tabla en un espacio de tabla de partición por rango (UTS) • Cambiar el atributo DSSIZE de un espacio de tabla particionado • Cambiar el tamaño de la página del búfer • Materializar los cambios de definición pendientes en los espacios de tablas con la utilidad REORG TABLESPACE. Para obtener más información, consulte Cambios de definición de datos pendientes (Db2 Administration Guide). • Modificar particiones de espacios de tabla particionados por rango (PBR) o particionados (no UTS), lo que incluye: añadir particiones, modificar claves de límite o rotar particiones.
Particiones	<ul style="list-style-type: none"> • Modificar particiones de espacios de tabla particionados por rango (PBR) o particionados (no UTS), lo que incluye: añadir particiones, modificar claves de límite o rotar particiones.
Índices	<ul style="list-style-type: none"> • Añadir una columna • Modificar un índice para regenerarlo • Modificar el atributo ACOGIDO o NO ACOGIDO • Modificación de un valor clave de límite de un índice de partición • Especificar NOT CLUSTER para el índice de partición de una tabla que utiliza partición controlada por índice, para convertir la tabla para que utilice partición controlada por tabla

Objeto u operación	Cambios que invalidan paquetes
	<ul style="list-style-type: none"> Materializar los cambios de definición pendientes en los índices con la utilidad REORG INDEX. Para obtener más información, consulte Cambios de definición de datos pendientes (Db2 Administration Guide).
Vistas	<ul style="list-style-type: none"> Modificar una vista para regenerarla Alterar una columna de la tabla de tal manera que una vista no pueda regenerarse
Paquetes	<ul style="list-style-type: none"> Dejar el paquete Dejar caer un paquete que proporciona el privilegio de ejecución para un plan
Rutinas	<ul style="list-style-type: none"> Procedimientos de regeneración. Para más información, consulte la información sobre la invalidación de paquetes en Declaración ALTER PROCEDURE (SQL - procedimiento nativo) (Db2 SQL). Alteración de una función externa Modificación de una función escalar SQL integrada Modificar una versión de una función escalar SQL compilada para cambiar ciertas opciones que están especificadas para la versión activa. Para más información, consulte la información sobre la invalidación de paquetes en Declaración ALTER FUNCTION (función escalar SQL compilada) (Db2 SQL). Alterar un procedimiento con la opción ACTIVATE VERSION <i>routine-version-id</i>, si el valor de <i>routine-version-id</i> es diferente de la versión activa actual del procedimiento. Para más información, consulte la información sobre la invalidación de paquetes en Declaración ALTER PROCEDURE (SQL - procedimiento nativo) (Db2 SQL). Alteración de las funciones de la tabla SQL: <ul style="list-style-type: none"> – Alteración del atributo SECURED (SEGURO) o NOT SECURED (NO SEGURO) – Alterar el atributo DETERMINISTIC o NOT DETERMINISTIC, independientemente de si se especifica RESTRICT – Regenerar una función de tabla
Descartar objetos	<ul style="list-style-type: none"> Dejar el paquete Dejar caer un paquete que proporciona el privilegio de ejecución para un plan Dejar caer objetos como alias, funciones de e , variables globales, índices, tablas de consulta materializadas, roles, secuencias, sinónimos, tablas, espacios de tablas, disparadores, vistas Eliminar una tabla clonada Soltar una columna. Para los cambios de definición pendientes, la invalidación del paquete se produce cuando el cambio de definición pendiente se aplica a la tabla. Eliminar permisos de fila o máscaras de columna si se aplica el control de acceso de columna para una tabla
Cambios en la autorización y el control de acceso	<ul style="list-style-type: none"> Revocar la autorización del propietario del paquete para acceder a una tabla, índice o vista

Objeto u operación	Cambios que invalidan paquetes
	<ul style="list-style-type: none"> Revocar la autorización del propietario del paquete para ejecutar un procedimiento almacenado, si el paquete utiliza el formato <i>CALL nombre</i>-procedimiento de la instrucción CALL para llamar al procedimiento almacenado Habilitar o deshabilitar máscaras si el control de acceso a columnas está en vigor Dejar caer un paquete que proporciona el privilegio de ejecución para un plan Eliminar permisos de fila o máscaras de columna si se aplica el control de acceso de columna para una tabla Activación o desactivación del control de acceso a nivel de fila para una tabla Activar el control de acceso a nivel de columna si la tabla tiene una columna habilitada, o desactivar el control de acceso a nivel de columna
operaciones de programa de utilidad	<ul style="list-style-type: none"> Materializar los cambios de definición pendientes en los espacios de tablas con la utilidad REORG TABLESPACE. Para obtener más información, consulte Cambios de definición de datos pendientes (Db2 Administration Guide). Materializar los cambios de definición pendientes en los índices con la utilidad REORG INDEX. Para obtener más información, consulte Cambios de definición de datos pendientes (Db2 Administration Guide). Ejecutar la utilidad REORG con la palabra clave REBALANCE Ejecutar la utilidad REPAIR en una base de datos con la opción DBD REBUILD

Consejo: Algunas alteraciones no invalidan los paquetes que dependen de los objetos requeridos. Sin embargo, es posible que a veces aún tenga que volver a vincular los paquetes para que la aplicación recopile los cambios. Para obtener más información, consulte “[Cambios que podrían requerir reempaquetados](#)” en la página 21.

Efectos en cascada en paquetes de renombrar una columna

ALTER TABLE RENAME COLUMN invalida cualquier paquete que dependa de la tabla en la que se cambia el nombre de la columna. Cualquier intento de ejecutar el paquete invalidado desencadena una reasociación automática del paquete.

La reasignación automática falla si se hace referencia a la columna en el paquete porque la columna a la que se hace referencia ya no existe en la tabla. En este caso, las aplicaciones que hacen referencia al paquete deben modificarse, recompilarse y readjustarse para que devuelvan el resultado esperado.

La reimpresión automática se realiza en cualquiera de los siguientes casos:

- El paquete no hace referencia a la columna. En este caso, el cambio de nombre de la columna no afecta a los resultados de la consulta que devuelve el paquete. No es necesario modificar la aplicación como resultado de cambiar el nombre de la columna.
- El paquete hace referencia a la columna, pero después de cambiar el nombre de la columna, se añade a la tabla otra columna con el nombre de la columna original. En este caso, cualquier consulta que haga referencia al nombre de la columna original podría devolver un conjunto de resultados diferente. Para restaurar los resultados esperados, sería necesario modificar la aplicación para especificar el nuevo nombre de la columna.

El siguiente escenario muestra cómo el cambio de nombre de una columna puede hacer que un paquete devuelva resultados inesperados:

```

CREATE TABLE MYTABLE (MYCOL1 INT);
INSERT INTO TABLE MYTABLE
    VALUES (1);
SELECT MYCOL1 FROM MYTABLE -- this is the statement in          -- the package MYPACKAGE,
                           -- the query returns
                           -- a value of 1
ALTER TABLE MYTABLE
    RENAME COLUMN
        MYCOL1 TO MYCOL2;           -- MYPACKAGE is invalidated
                           -- and automatic rebind
                           -- of MYPACKAGE will fail
                           -- at this point
ALTER TABLE MYTABLE
    ADD COLUMN MYCOL1 VARCHAR(10);   -- automatic rebind
                           -- of MYPACKAGE
                           -- will be successful
INSERT INTO TABLE MYTABLE (MYCOL1)
    VALUES ('ABCD');

```

En este punto, una aplicación ejecuta MYPACKAGE, lo que da como resultado una reasociación automática correcta. Sin embargo, la declaración en el paquete devolverá 'ABCD' en lugar de la esperada '1'.

Conceptos relacionados

Revinculación automática

Las reasociaciones automáticas (a veces llamadas "autobindings") se producen cuando un usuario autorizado ejecuta un paquete o plan y las estructuras de tiempo de ejecución del plan o paquete no se pueden utilizar. Esta situación se da normalmente cuando se modifican los atributos de los datos de los que depende el paquete o plan, o si se modifica el entorno en el que se ejecuta el paquete o plan.

"Paquetes desencadenantes" en la página 172

Un paquete de activación es un tipo especial de paquete que se crea solo cuando ejecuta una instrucción CREATE TRIGGER. Un paquete desencadenante se ejecuta sólo cuando se activa el desencadenante asociado.

Invalidate dynamic statements in memory cache (Performance of Db2)

Tareas relacionadas

Identificación de paquetes con características que afectan al rendimiento, la simultaneidad o la capacidad de ejecución (Performance of Db2)

"Revinculación de aplicaciones" en la página 947

Debe volver a enlazar las aplicaciones para cambiar las opciones de enlace. También es necesario volver a enlazar las aplicaciones cuando realice cambios que afecten al plan o al paquete, como por ejemplo la creación de un índice, pero no haya cambiado las sentencias SQL.

Referencia relacionada

Paquetes no válidos y no operativos (Managing Security)

Información relacionada

00E30305 (Db2 Codes)

Identificación de paquetes invalidados

Puede identificar los paquetes que se invalidarán cuando se realicen determinados cambios en los objetos.

Acerca de esta tarea

Ciertos cambios en los objetos invalidan los paquetes. Al identificar estos paquetes invalidados antes de realizar los cambios, puede preparar las operaciones de reencuadernación necesarias en consecuencia.

Procedimiento

Para identificar todos los paquetes que quedarán invalidados por un cambio en un objeto específico, ejecute la siguiente consulta:

```

SELECT    DISTINCT DCOLLID, DNAME, DTYP
FROM      SYSIBM.SYSPACKDEP
WHERE     BQUALIFIER = object_qualifier
        AND      BNAME      = object_name
        AND      BTYPE      = object_type
ORDER BY  DCOLLID, DNAME;

```

objeto_calificador

El calificador del objeto

nombre_objeto

El nombre del objeto

tipo_objeto

El tipo de objeto

Resultados

La consulta devuelve una tabla que contiene información del paquete basada en los valores seleccionados en la consulta. Para obtener más información sobre los valores seleccionados, consulte [Tabla de catálogo SYSPACKDEP \(Db2 SQL\)](#).

Cambios que podrían requerir reempaquetados

Algunos cambios en los objetos de la base de datos que no hacen que los paquetes se invaliden podrían requerir una nueva vinculación para que los cambios surtan efecto en la aplicación.

Las siguientes sentencias SQL hacen que Db2 establezca el valor de la columna VALID en 'A' en la tabla del catálogo SYSIBM.SYSPACKAGE. Este valor indica que una instrucción SQL cambió la descripción de la tabla o tabla base de una vista a la que hace referencia el paquete. Estos cambios no invalidan el paquete. Sin embargo, puede ser necesario volver a encuadrinar el paquete para que refleje los cambios del extracto.

- Sentencias ALTER TABLE con las cláusulas siguientes:
 - AÑADIR COLUMNA (excepto en los casos que invaliden los paquetes; véase “[Cambios que invalidan paquetes](#)” en la página 16)
 - AÑADIR o ELIMINAR CLAVE EXTRANJERA
 - AÑADIR o ELIMINAR ÚNICO
 - Restricción DROP
 - AÑADIR CLAVE DE PARTICIÓN
 - AÑADIR o ELIMINAR CHEQUE
 - VALIDPROC
 - VOLÁTIL o NO VOLÁTIL
 - APÉNDICE SÍ o NO
 - AÑADIR PUNTO si el paquete está sujeto con BUSTIMESENSITIVE (NO)
 - AÑADIR o ELIMINAR VERSIONES si el paquete está vinculado con SENSIBLE AL SISTEMA (NO)
 - ACTIVAR o DESACTIVAR ARCHIVO si el paquete está vinculado con ARCHIVOSENSIBLE (NO)
- Declaraciones de CAMBIO

Conceptos relacionados

[Cambios que invalidan paquetes](#)

Los cambios realizados en los objetos de programa o de base de datos pueden invalidar paquetes.

Referencia relacionada

[Tabla de catálogo SYSPACKAGE \(Db2 SQL\)](#)

[ALTER TABLE declaración \(Db2 SQL\)](#)

[EXCHANGE declaración \(Db2 SQL\)](#)

Determinar el valor de cualquier opción vinculante que afecte al diseño de su programa

Varias opciones de los comandos BIND PACKAGE y BIND PLAN pueden afectar al diseño de su programa. Por ejemplo, puede utilizar una opción de enlace para asegurarse de que un paquete o plan solo se puede ejecutar desde una conexión concreta CICS conexión o IMS región. Su código no necesita forzar esta situación.

Procedimiento

Revise la lista de opciones de enlace y decida los valores de las opciones que afecten a la forma en que escribe su programa.

Por ejemplo, debe decidir los valores de las opciones ACQUIRE y RELEASE antes de escribir su programa. Estas opciones determinan cuándo su aplicación adquiere y libera bloqueos en los objetos que utiliza.

Referencia relacionada

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2\)](#)

Programación de aplicaciones para mejorar el rendimiento

Puede mejorar el rendimiento de Db2 teniendo en cuenta el rendimiento cuando programe y despliegue aplicaciones.

Procedimiento

Para mejorar el rendimiento de los programas de aplicación que acceden a los datos en Db2, utilice los siguientes enfoques al escribir y preparar los programas:

- Programe sus aplicaciones para mejorar la simultaneidad.

El objetivo es programar y preparar aplicaciones de una forma que:

- Protege la integridad de los datos que se están leyendo o actualizando de ser modificados por otras aplicaciones.
- Minimiza la longitud de tiempo que impide otro acceso a los datos.

Para obtener más información sobre la simultaneidad de datos en Db2 y recomendaciones para mejorar la simultaneidad en sus programas de aplicación, consulte los temas siguientes:

- [Programación para mejorar la simultaneidad \(Db2 Performance\)](#)
- [Diseño de bases de datos para mejorar la simultaneidad \(Db2 Performance\)](#)
- [Simultaneidad y bloqueos \(Db2 Performance\)](#)
- [Mejora de la simultaneidad \(Db2 Performance\)](#)
- [Mejora de la simultaneidad en entornos de compartición de datos \(Db2 Data Sharing Planning and Administration\)](#)

- Escriba sentencias SQL que accedan a datos de forma eficiente.

Los predicados, las subconsultas y otras estructuras de sentencias de SQL afectan a las vías de acceso que Db2 utiliza para acceder a los datos.

Para obtener información sobre cómo grabar sentencias SQL que acceden a los datos de forma eficaz, consulte los temas siguientes:

- [Modos de mejorar el rendimiento de las consultas \(Introducción a DB2 para z/OS\)](#)
- [Escritura eficiente de consultas SQL \(Db2 Performance\)](#)
- Utilice las herramientas de optimización de EXPLAIN o SQL para analizar las vías de acceso que Db2 elige para procesar las sentencias SQL.

Al analizar la vía de acceso que Db2 utiliza para acceder a los datos de una sentencia de SQL, puede descubrir problemas potenciales. Puede utilizar esta información para modificar la sentencia para que actúe mejor.

Para obtener información sobre cómo utilizar las tablas de EXPLAIN para analizar las vías de acceso para las sentencias de SQL, consulte los temas siguientes:

- [Investigación de problemas de la vía de acceso \(Db2 Performance\)](#)
 - [00C200A4 \(Db2 Codes\)](#)
 - [Investigación del rendimiento de SQL utilizando EXPLAIN \(Db2 Performance\)](#)
 - [Interpretación del acceso a datos mediante el uso de EXPLAIN \(Db2 Performance\)](#)
 - [Tablas de EXPLAIN \(Db2 Performance\)](#)
 - [EXPLAIN declaración \(Db2 SQL\)](#)
 - Tenga en cuenta el rendimiento a la hora de diseñar aplicaciones que accedan a datos distribuidos. El objetivo es reducir la cantidad de tráfico de red necesaria para acceder a los datos distribuidos y para gestionar el uso de los recursos del sistema como las conexiones y hebras de acceso a la base de datos distribuida.
- Para obtener información acerca de la mejora del rendimiento de aplicaciones que acceden a los datos distribuidos, consulte los temas siguientes:
- [Modos de reducir el tráfico de la red \(Introducción a DB2 para z/OS\)](#)
 - [Gestión de hebras de Db2 \(Db2 Performance\)](#)
 - [Mejora del rendimiento para las aplicaciones que acceden a datos distribuidos \(Db2 Performance\)](#)
 - [Mejora del rendimiento para sentencias SQL en aplicaciones distribuidas \(Db2 Performance\)](#)
 - Utilice procedimientos almacenados para mejorar el rendimiento y tenga en cuenta el rendimiento cuando cree procedimientos almacenados.

Para obtener información sobre procedimientos almacenados y rendimiento de Db2, consulte los temas siguientes:

- [Implementación de procedimientos almacenados de Db2 \(procedimientos almacenados proporcionados por Db2\)](#)
- [Mejora del rendimiento de procedimientos almacenados y funciones definidas por el usuario \(Db2 Performance\)](#)

Conceptos relacionados

Análisis del rendimiento de consultas y aplicaciones ([Introducción a DB2 para z/OS](#))

Programación para la interfaz de recurso de instrumentación (IFI) ([Db2 Performance](#))

Tareas relacionadas

[Descripción general de las aplicaciones de programación que acceden a datos de Db2 for z/OS](#)

Las aplicaciones que interactúan con Db2, en primer lugar se deben conectar a Db2. Entonces pueden leer, añadir o modificar datos o manipular objetos de Db2.

[Establecimiento de límites para el uso de recursos de sistema utilizando el recurso de límite de recursos \(Db2 Performance\)](#)

[Planificación y diseño de aplicaciones de Db2](#)

Es necesario tomar decisiones acerca de la planificación y el diseño antes de escribir o ejecutar el programa. Es necesario tomar estas decisiones tanto si escribe una nueva aplicación de Db2 como si migra una aplicación existente de un release anterior de Db2.

Diseño de su solicitud de recuperación

Si su aplicación falla o si Db2 se cierra de forma anormal, debe asegurarse de la integridad de los datos que se manipularon en su aplicación. Debe tener en cuenta posibles situaciones de recuperación cuando diseñe su aplicación.

Procedimiento

Para diseñar su solicitud de recuperación:

1. Introduzca los cambios que lógicamente deban realizarse al mismo tiempo en la misma unidad de trabajo. Esta acción garantiza que, en caso de que Db2 se cierre de forma anormal o de que su aplicación falle, los datos se mantengan en un estado coherente.

Una unidad de trabajo es un procedimiento lógicamente distinto que contiene pasos que modifican los datos. Si todos los pasos se completan correctamente, querrá que los cambios de datos sean permanentes. Pero, si alguno de los pasos falla, querrás que todos los datos modificados vuelvan al valor original antes de que comenzara el procedimiento. Por ejemplo, supongamos que dos empleados de la muestra intercambian oficinas en la tabla DSN8C10.EMP. Debe intercambiar sus números de teléfono de la oficina en la columna NÚMERO DE TELÉFONO. Debe utilizar dos sentencias UPDATE para actualizar cada número de teléfono. Ambas declaraciones, tomadas en conjunto, son una unidad de trabajo. Desea que ambas declaraciones se completen correctamente. Por ejemplo, si solo una declaración tiene éxito, querrá que ambos números de teléfono vuelvan a sus valores originales antes de intentar otra actualización.

2. Considere con qué frecuencia debe realizar cambios en los datos.

Si su programa falla o el sistema falla, Db2 retrocede todos los cambios de datos no confirmados. Los datos modificados vuelven a su estado original sin interferir con otras actividades del sistema.

Para IMS y CICS aplicaciones, si el sistema falla, los datos de Db2 encia no siempre vuelven a un estado coherente de inmediato. Db2 no procesa datos indudables (datos que no están ni comprometidos ni descomprometidos) hasta que reinicie IMS o la CICS función de adjuntar archivos. Para asegurarse de que Db2 y IMS estén sincronizados, reinicie tanto Db2 como IMS. Para asegurarse de que Db2 y CICS estén sincronizados, reinicie tanto Db2 como el CICS función de adjuntos.

3. Considere si su aplicación debe interceptar abends.

Si su aplicación intercepta un error, Db2 continúa trabajando, porque no es consciente de que se ha producido un error. Si desea que Db2 revierta el trabajo automáticamente cuando se produce un error en su programa, no permita que el programa o el entorno de tiempo de ejecución intercepten el error. Si su programa utiliza Entorno de lenguaje y desea que Db2 revierta el trabajo automáticamente cuando se produce un abend en el programa, especifique las opciones de tiempo de ejecución ABTERMENC(ABEND) y TRAP(ON).

4. **Solo para aplicaciones TSO** : Emite instrucciones COMMIT antes de conectarse a otro DBMS.

Si el sistema falla en este punto, Db2 no puede saber si su transacción se ha completado. En este caso, como en el caso de un fallo durante una operación de consignación en una sola fase para un único subsistema, debe tomar sus propias medidas para mantener la integridad de los datos.

5. **Solo para aplicaciones TSO** : Determine si desea proporcionar una rutina de salida abend en su programa.

Si proporciona esta rutina, debe utilizar indicadores de seguimiento para determinar si se produce un error durante el procesamiento de un Db2 . Si se produce un error cuando Db2 tiene el control, debe permitir que se complete la finalización de la tarea. Db2 detecta la finalización de la tarea y finaliza el hilo con el parámetro ABRT. No vuelva a ejecutar el programa.

Permitir que se complete la finalización de la tarea es la única acción que puede realizar para los cierres que se deben al comando CANCEL o a DETACH. No puede utilizar instrucciones SQL adicionales en este punto. Si intenta ejecutar otra instrucción SQL desde el programa de aplicación o su rutina de recuperación, pueden producirse errores inesperados.

Conceptos relacionados

[Unidad de trabajo \(Introducción a DB2 para z/OS\)](#)

Unidad de trabajo en TSO

Las aplicaciones que utilizan la función de adjunto TSO pueden definir explícitamente unidades de trabajo mediante las sentencias SQL COMMIT y ROLLBACK.

En las aplicaciones TSO, una unidad de trabajo comienza cuando se producen las primeras actualizaciones de un objeto de intercambio de datos (Db2). Una unidad de trabajo finaliza cuando se produce una de las siguientes condiciones:

- El programa emite una declaración COMMIT posterior. En este punto del procesamiento, su programa ha determinado que los datos son coherentes; todos los cambios de datos que se realizaron desde el punto de confirmación anterior se realizaron correctamente.
- El programa emite una declaración ROLLBACK posterior. En este punto del procesamiento, su programa ha determinado que los cambios de datos no se han realizado correctamente y, por lo tanto, no deberían ser permanentes. Una declaración ROLLBACK hace que se deshagan todos los cambios de datos que se hayan realizado desde el último punto de confirmación.
- El programa finaliza y vuelve al procesador de comandos DSN, que vuelve al programa de monitorización de terminal (TMP) TSO.

La primera y la tercera condiciones de la lista anterior se denominan punto de confirmación. *Un punto de confirmación* se produce cuando se emite una instrucción COMMIT o cuando el programa finaliza con normalidad.

Referencia relacionada

[COMMIT declaración \(Db2 SQL \)](#)

[ROLLBACK declaración \(Db2 SQL \)](#)

Unidad de trabajo en CICS

CICS pueden definir explícitamente unidades de trabajo utilizando el CICS Comando SYNCPOINT. De forma alternativa, las unidades de trabajo se definen implícitamente mediante varios puntos de ruptura lógica.

Todo el procesamiento que ocurre en su programa entre dos puntos de confirmación se conoce como unidad lógica de trabajo (LUW) o unidad de trabajo. En CICS aplicaciones, una unidad de trabajo se marca como completa mediante un punto de confirmación o sincronización (sync), que se define de una de las siguientes maneras:

- Implícitamente al final de una transacción, que se señala mediante un CICS Comando RETURN en el nivel lógico más alto.
- De forma explícita mediante CICS SYNCPOINT que el programa emite en puntos lógicamente apropiados de la transacción.
- Implícitamente a través de una llamada o comando de terminación DL/I PSB (TERM).
- Implícitamente cuando un programa DL/I por lotes emite una llamada de punto de control DL/I. Esta llamada puede producirse cuando el programa DL/I por lotes comparte una base de datos con CICS aplicaciones a través de la función de uso compartido de bases de datos.

Por ejemplo, considere un programa que resta la cantidad de artículos vendidos de un archivo de inventario y luego agrega esa cantidad a un archivo de reorden. Cuando ambas transacciones se completen (y no antes) y los datos de los dos archivos sean coherentes, el programa podrá emitir una llamada DL/I TERM o un comando SYNCPOINT. Si uno de los pasos falla, desea que los datos vuelvan al valor que tenían antes de que comenzara la unidad de trabajo. Es decir, que desea que vuelva a un punto anterior de coherencia. Puede lograr este estado utilizando el comando SYNCPOINT con la opción ROLLBACK.

Al utilizar un comando SYNCPOINT con la opción ROLLBACK, puede deshacer los cambios de datos no confirmados. Por ejemplo, un programa que actualiza un conjunto de filas relacionadas a veces encuentra un error después de actualizar varias de ellas. El programa puede utilizar el comando SYNCPOINT con la opción ROLLBACK para deshacer todas las actualizaciones sin perder el control.

Las sentencias SQL COMMIT y ROLLBACK no son válidas en un CICS entorno. Puede coordinar Db2 con CICS funciones que se utilizan en los programas, de modo que los datos de Db2 y los que no son de Db2 sean coherentes.

Planificación de la recuperación de programas en IMS programas

Para estar preparado para situaciones de recuperación de IMS programas que acceden a datos de Db2 , es necesario tomar varias decisiones de diseño que son específicas de IMS los programas. Estas decisiones se suman a las recomendaciones generales que debe seguir al diseñar su solicitud de recuperación.

Acerca de esta tarea

Tanto IMS y Db2 manejan la recuperación en un IMS programa de aplicación que accede a datos de Db2 . IMS coordina el proceso y Db2 se encarga de la recuperación de los datos de Db2 .

Procedimiento

Para planificar la recuperación del programa en IMS programas:

1. Para un programa que procesa mensajes como entrada, decida si especifica transacciones de modo único o de modo múltiple en la sentencia TRANSACT de la macro APPLCTN para el programa.

Modo único

Indica que se produce un punto de confirmación en un Db2 a cada vez que el programa emite una llamada para recuperar un mensaje nuevo. Especificar el modo único puede simplificar la recuperación; si el programa falla, puede reiniciar desde la última llamada para un mensaje nuevo. Cuando IMS reinicia el programa, este comienza procesando el siguiente mensaje.

Múltiples modos

Indica que se produce un punto de confirmación cuando el programa emite una llamada de punto de control o cuando finaliza con normalidad. Esos dos eventos son los únicos momentos durante el programa en los que IMS envía los mensajes de salida del programa a sus destinos. Dado que en los programas de modo múltiple se procesan menos puntos de compromiso que en los programas de modo único, los programas de modo múltiple podrían funcionar ligeramente mejor que los programas de modo único. Cuando un programa de modo múltiple falla, IMS solo puede reiniciar desde una llamada de punto de control. En lugar de tener que volver a procesar solo el mensaje más reciente, un programa puede tener varios mensajes que procesar. El número de mensajes a procesar depende de cuándo el programa emitió la última llamada de comprobación.

Db2 realiza algún procesamiento con programas de modo único y múltiple. Cuando un programa de modo múltiple realiza una llamada para recuperar un mensaje nuevo, Db2 realiza una comprobación de autorización y cierra todos los cursores abiertos en el programa.

2. Decidir si se emiten llamadas de punto de control (CHKP) y, en caso afirmativo, con qué frecuencia.

Cada llamada indica IMS que el programa ha alcanzado un punto de sincronización y establece un lugar en el programa desde el que puede reiniciar el programa.

Tenga en cuenta los siguientes factores a la hora de decidir cuándo utilizar las llamadas de control:

- Cuánto tiempo se tarda en deshacer y recuperar esa unidad de trabajo. El programa debe emitir puntos de control con la suficiente frecuencia para que sea fácil retroceder y recuperarse.
- Cuánto tiempo se bloquean los recursos de la base de datos en Db2 y IMS.
- Para programas de modo múltiple: Cómo desea que se agrupen los mensajes de salida. Las llamadas de punto de control establecen cómo un programa de modo múltiple agrupa sus mensajes de salida. Los programas deben emitir puntos de control con la suficiente frecuencia para evitar la acumulación de demasiados mensajes de salida.

Restricción: No puede utilizar instrucciones SQL COMMIT y ROLLBACK en el entorno de soporte de lotes DL/I de Db2 , porque IMS coordina la unidad de trabajo.

3. Emite declaraciones CERRAR CURSOR antes de cualquier llamada de punto de control o llamada GU a la cola de mensajes, no después.
4. Después de cualquier llamada de punto de control, establezca el valor de cualquier registro especial que se haya restablecido si sus valores son necesarios después del punto de control:

Una llamada de CHKP hace que IMS se vuelva a conectar a Db2 , lo que restablece los registros especiales que se muestran en la siguiente tabla.

Tabla 1. Registros especiales que se restablecen mediante una llamada de punto de control.

Registro especial	Valor al que se restablece después de una llamada de punto de control
CURRENT PACKAGESET	blancos
CURRENT SERVER	blancos
CURRENT SQLID	blancos
CURRENT DEGREE	1

5. Después de cualquier punto de confirmación, vuelva a abrir los cursos que desee y restablezca el posicionamiento

6. Decida si desea especificar la opción CON RETENCIÓN para cualquier cursor.

Esta opción determina si el programa conserva la posición del cursor en la base de datos de Db2 después de que usted emita IMS Llamadas CHKP. Siempre se pierde el posicionamiento de la base de datos del programa en DL/I después de una IMS Llamada CHKP.

El posicionamiento de la base de datos del programa en Db2 se ve afectado de acuerdo con los siguientes criterios:

- Si no especifica la opción CON RETENCIÓN para un cursor, perderá la posición de ese cursor.
- Si especifica la opción CON RETENCIÓN para un cursor y la aplicación está controlada por mensajes, perderá la posición de ese cursor.
- Si especifica la opción CON RETENCIÓN para un cursor y la aplicación está funcionando en DL/I batch o DL/I BMP, conservará la posición de ese cursor.

7. Utilice IMS rollback calls, ROLL y ROLB, para revertir los cambios de Db2 y DL/I al último punto de confirmación.

Estas opciones tienen las siguientes diferencias:

ROLLO

Especifica que todos los cambios desde el último punto de confirmación deben revertirse y que el programa debe finalizar. IMS termina el programa con el código de error de usuario U0778 y sin volcado de memoria.

Cuando emite una llamada ROLL, la única opción que proporciona es la función de llamada, ROLL.

ROLLB

Especifica que todos los cambios desde el último punto de confirmación deben revertirse y el control debe devolverse al programa para que pueda continuar el procesamiento.

Una llamada ROLB tiene las siguientes opciones:

- La función de llamada, ROLB
- El nombre de la PCB de E/S

La forma en que las llamadas ROLL y ROLB afectan a los cambios DL/I en un entorno por lotes depende del IMS registro del sistema y las opciones de retroceso especificadas, como se muestra en la siguiente tabla.

Tabla 2. Efectos de las llamadas ROLL y ROLLB en los cambios DL/I en un entorno por lotes

Opciones especificadas			
Llamada de reversión	Opción de registro del sistema	Opción de devolución	Resultado
ROLLO	Cinta	any	DL/I no realiza actualizaciones y se producen errores de conexión a Internet (U0778). Db2 retrocede las actualizaciones al punto de control anterior.
	disco	BKO=NO	
	disco	BKO=SÍ	DL/I se retracta de las actualizaciones y se produce un error de U0778. Db2 retrocede las actualizaciones al punto de control anterior.

Tabla 2. Efectos de las llamadas ROLL y ROLLB en los cambios DL/I en un entorno por lotes (continuación)

Opciones especificadas			
Llamada de reversión	Opción de registro del sistema	Opción de devolución	Resultado
ROLB	Cinta	any	DL/I no retrocede las actualizaciones, y se devuelve un código de estado AL en el PCB. Db2 retrocede las actualizaciones al punto de control anterior. El soporte DL/I de Db2 hace que el programa de aplicación falle cuando ROLB falla.
	disco	BKO=NO	
		BKO=SÍ	DL/I se retira de las actualizaciones de la base de datos y el control se devuelve al programa de aplicación. Db2 retrocede las actualizaciones al punto de control anterior.
		Restricción: No puede especificar la dirección de un área de E/S como una de las opciones de la llamada; si lo hace, su programa recibe un código de estado AD. Sin embargo, debe tener un PCB de E/S para su programa. Especifique CMPAT=YES en la palabra clave CMPAT en la declaración PSBGEN para el PSB de su programa.	

Conceptos relacionados

Puntos de comprobación en programas IMS

La emisión de un punto de comprobación llama a los recursos bloqueados de los releases y establece un lugar en el programa desde el que puede reiniciar el programa. La decisión sobre si el programa debe emitir puntos de comprobación (y si es así, con cuánta frecuencia) depende de su programa.

Unidad de trabajo en IMS programas en línea

IMS las aplicaciones pueden definir explícitamente unidades de trabajo mediante una llamada CHKP, SYNC, ROLL o ROLB, o, para transacciones monomodo, una llamada GU.

En IMS, una unidad de trabajo comienza cuando ocurre uno de los siguientes eventos:

- Cuando se inicia el programa

- Después de que se haya completado una llamada CHKP, SYNC, ROLL o ROLB
- Para transacciones monomodo, cuando se emite una llamada GU al PCB de E/S

Una unidad de trabajo finaliza cuando se produce uno de los siguientes eventos:

- El programa emite una llamada CHKP o SYNC posterior o, para transacciones monomodo, una llamada GU al PCB de E/S. En este punto del procesamiento, los datos son coherentes. Todos los cambios de datos que se realizaron desde el punto de confirmación anterior se realizan correctamente.
- El programa emite una llamada ROLB o ROLL posterior. En este punto del procesamiento, su programa ha determinado que los cambios de datos no son correctos y, por lo tanto, que los cambios de datos no deben ser permanentes.
- El programa finaliza.

Restricción: Las sentencias SQL COMMIT y ROLLBACK no son válidas en un IMS entorno.

Un punto de confirmación se produce en un programa como resultado de cualquiera de los siguientes eventos:

- El programa finaliza normalmente. La finalización normal del programa es siempre un punto de compromiso.
- El programa emite una llamada de punto de control. *Las llamadas de punto de control* son un medio del programa para indicar explícitamente IMS que ha alcanzado un punto de compromiso en su procesamiento.
- El programa emite una llamada SYNC. Una *llamada SYNC* es una llamada de servicio del sistema Fast Path para solicitar el procesamiento del punto de confirmación. Solo puede utilizar una llamada SYNC en un programa Fast Path no basado en mensajes.
- Para un programa que procesa mensajes como entrada, un punto de confirmación puede ocurrir cuando el programa recupera un nuevo mensaje. Este comportamiento depende del modo que especifique en la macro APPLCTN para el programa:
 - Si especifica transacciones de modo único, se produce un punto de confirmación en un Db2 e cada vez que el programa emite una llamada para recuperar un nuevo mensaje.
 - Si especifica transacciones de modo múltiple o no especifica un modo, se produce un punto de confirmación cuando el programa emite una llamada de punto de control o cuando finaliza normalmente.

En el momento de un punto de confirmación, se producen las siguientes acciones:

- IMS y Db2 pueden liberar bloqueos que el programa ha mantenido desde el último punto de confirmación. Al desbloquear estos bloqueos, los datos quedan disponibles para otros programas de aplicación y usuarios.
- Db2 cierra cualquier cursor abierto que el programa haya estado utilizando.
- IMS y Db2 hacen que los cambios del programa en la base de datos sean permanentes.
- Si el programa procesa mensajes, IMS envía los mensajes de salida que el programa de aplicación produce a sus destinos finales. Hasta que el programa llegue a un punto de confirmación, IMS mantiene los mensajes de salida del programa en un destino temporal.

Si el programa falla antes de llegar al punto de confirmación, se producen las siguientes acciones:

- Tanto IMS y Db2 deshacen todos los cambios que el programa ha hecho en la base de datos desde el último punto de confirmación.
- IMS elimina cualquier mensaje de salida que el programa haya producido desde el último punto de confirmación (para PCB no expresas).
- Si el programa procesa mensajes, las personas en los terminales y otros programas de aplicación reciben información del programa de aplicación terminal.

Si el sistema falla, una unidad de trabajo se resuelve automáticamente cuando Db2 y IMS los programas por lotes se vuelven a conectar. Cualquier unidad de trabajo en duda se resuelve en el momento de la reconexión.

Especificar la frecuencia de los puntos de control en IMS programas

Un punto de control indica un punto de confirmación en IMS los programas. Debe especificar la frecuencia de los puntos de control en su programa de manera que se pueda cambiar fácilmente, en caso de que la frecuencia que especifique inicialmente no sea la adecuada.

Procedimiento

Para especificar la frecuencia de los puntos de control en IMS programas:

1. Utilice un contador en su programa para llevar un registro de uno de los siguientes elementos:
 - Tiempo transcurrido
 - El número de segmentos raíz a los que accede su programa
 - El número de actualizaciones que realiza su programa
2. Emite una llamada de punto de control después de un intervalo de tiempo determinado, un número de segmentos raíz o un número de actualizaciones.

Puntos de control en IMS programas

La emisión de un punto de comprobación llama a los recursos bloqueados de los releases y establece un lugar en el programa desde el que puede reiniciar el programa. La decisión sobre si el programa debe emitir puntos de comprobación (y si es así, con cuánta frecuencia) depende de su programa.

Por lo general, los siguientes tipos de programas deberían emitir llamadas de punto de control:

- Programas de modalidad múltiple
- BMP orientados a lotes
- Programas Fast Path no basados en mensajes. (Estos programas pueden utilizar una llamada especial de Fast Path, pero también pueden utilizar llamadas de punto de control simbólico)
- La mayoría de los programas por lotes
- Programas que se ejecutan en un entorno de intercambio de datos. (El intercambio de datos hace posible que los programas de aplicación en línea y por lotes en sistemas separados IMS sistemas, en procesadores iguales o distintos, para acceder a bases de datos simultáneamente. Es importante emitir llamadas de punto de control con frecuencia en programas que se ejecutan en un entorno de intercambio de datos, porque los programas en varios IMS sistemas acceden a la base de datos)

No es necesario que emita puntos de control en los siguientes tipos de programas:

- Programas de un solo modo
- Programas de carga de bases de datos
- Programas que acceden a la base de datos en modo de solo lectura (definido con la opción de procesamiento GO durante un PSBGEN) y son lo suficientemente cortos como para reiniciarse desde el principio
- Programas que, por su naturaleza, deben tener uso exclusivo de la base de datos

Una llamada de CHKP hace que IMS realizar las siguientes acciones:

- Informe a Db2 de que los cambios que su programa ha realizado en la base de datos pueden ser permanentes. Db2 hace que los cambios en los datos de Db2 sean permanentes, y IMS hace que los cambios en IMS permanentes.
- Enviar un mensaje que contenga la identificación del punto de control que se da en la llamada al operador de la consola del sistema y al IMS operador de terminal maestro (MTO).
- Devuelve el siguiente mensaje de entrada al área de E/S del programa si el programa procesa mensajes de entrada. En los MPP y los BMP orientados a transacciones, una llamada de punto de control actúa como una llamada para un nuevo mensaje.
- Vuelva a iniciar sesión en Db2 .

Los programas que emiten llamadas simbólicas de punto de control pueden especificar hasta siete áreas de datos en el programa que se va a restaurar al reiniciar. Db2 siempre se recupera hasta el último punto de control. Debe reiniciar el programa desde ese punto.

Si utiliza llamadas de punto de control simbólico, puede utilizar una llamada de reinicio (XRST) para reiniciar un programa después de un fallo. Esta llamada restaura las áreas de datos del programa a su estado anterior al cierre anormal del programa, y reinicia el programa desde la última llamada de punto de control que el programa emitió antes de cerrarse de forma anormal.

Restricción: Para los programas BMP que procesan bases de datos de Db2 , puede reiniciar el programa solo desde el último punto de control y no desde cualquier punto de control, como en IMS.

Puntos de control en los MPP y BMP orientados a las transacciones

En los programas de modo único, las llamadas de punto de control y las llamadas de recuperación de mensajes (llamadas get-unique) establecen puntos de confirmación. Las llamadas de punto de control recuperan mensajes de entrada y sustituyen a las llamadas get-unique. Los BMP que acceden a bases de datos no DL/I y los MPP pueden emitir tanto llamadas únicas como llamadas de punto de control para establecer puntos de confirmación. Sin embargo, los BMP basados en mensajes deben emitir llamadas de punto de control en lugar de llamadas únicas para establecer puntos de confirmación, ya que solo pueden reiniciarse desde un punto de control. Si un programa falla después de emitir una llamada get-unique, IMS retrocede las actualizaciones de la base de datos al punto de confirmación más reciente, que es la llamada get-unique.

En los BMP y MPP de modo múltiple, los únicos puntos de confirmación son las llamadas de punto de control que emite el programa y la finalización normal del programa. Si el programa falla y no ha emitido llamadas de punto de control, IMS retira las actualizaciones de la base de datos del programa y cancela los mensajes que ha creado desde el inicio del programa. Si el programa ha emitido llamadas de punto de control, IMS retira los cambios del programa y cancela los mensajes de salida que ha creado desde la llamada de punto de control más reciente.

Puntos de control en BMP orientados a lotes

Si un BMP orientado a lotes no emite puntos de comprobación con la suficiente frecuencia, IMS puede fallar ese BMP u otro programa de aplicación por una de las siguientes razones:

- Otros programas no pueden acceder a los datos que necesitan en un plazo de tiempo determinado.
Si un BMP recupera y actualiza muchos registros de la base de datos entre llamadas de punto de control, puede monopolizar grandes porciones de las bases de datos y causar largas esperas para otros programas que necesitan esos segmentos. (La excepción a esta situación es un BMP con una opción de procesamiento de GO; IMS no pone en cola segmentos para programas con esta opción de procesamiento) La emisión de llamadas de punto de control libera los segmentos que el BMP ha puesto en cola y los pone a disposición de otros programas.
- No hay suficiente espacio de almacenamiento disponible para los segmentos que el programa ha leído y actualizado.

Si IMS utiliza la puesta en cola de aislamiento de programas, el espacio necesario para poner en cola la información sobre los segmentos que el programa ha leído y actualizado no debe exceder la cantidad de almacenamiento definida para el IMS sistema. (La cantidad de almacenamiento disponible se especifica durante la IMS definición del sistema.) Si un BMP pone en cola demasiados segmentos, la cantidad de almacenamiento necesaria para los segmentos en cola puede superar la cantidad de almacenamiento disponible. En ese caso, IMS el programa se cierra de forma anormal. A continuación, debe aumentar la frecuencia de los puntos de control del programa antes de volver a ejecutarlo.

Cuando se emite una llamada DL/I CHKP desde un programa de aplicación que utiliza bases de datos de Db2 , IMS procesa la llamada CHKP para todas las bases de datos DL/I, y Db2 compromete todos los recursos de la base de datos Db2 . No se registra información de puntos de control para bases de datos de Db2 en el IMS registro o el registro de Db2 . El programa de aplicación debe registrar la información relevante sobre las bases de datos de Db2 para un punto de control, si es necesario. Una forma de registrar dicha información es ponerla en un área de datos que se incluye en la llamada DL/I CHKP.

El rendimiento puede verse ralentizado por el procesamiento de confirmación que realiza Db2 durante una llamada DL/I CHKP, ya que el programa necesita restablecer la posición dentro de una base de datos de Db2. La forma más rápida de restablecer una posición en una base de datos e Db2 es utilizar un índice en la tabla de destino, con una clave que coincida uno a uno con cada columna del predicado SQL.

Recuperación de datos en IMS programas

Los IMS se encargan de la recuperación y el reinicio. Para una región de lotes, los procedimientos operativos controlan la recuperación y el reinicio de su ubicación.

Procedimiento

Realice una o más de las siguientes acciones según el tipo de programa:

Tipo de programa	Acción recomendada
Solicitudes por lotes DL/I	Utilice la utilidad de reversión de lotes DL/I para revertir los cambios DL/I. Db2 retira automáticamente los cambios cuando el programa de aplicación falla.
Aplicaciones que utilizan puntos de control simbólicos	Utilice una llamada de reinicio (XRST) para reiniciar un programa después de un fallo. Esta llamada restaura las áreas de datos del programa a su estado anterior al cierre anormal del programa, y reinicia el programa desde la última llamada de punto de control que el programa emitió antes de cerrarse anormalmente.
Programas BMP que acceden a bases de datos de Db2	Reinicia el programa desde el último punto de control. Restricción: Puede reiniciar el programa solo desde el último punto de control y no desde cualquier punto de control, como en IMS.
Aplicaciones que utilizan sistemas en línea IMS sistemas	No es necesaria ninguna acción. La recuperación y el reinicio forman parte del IMS sistema
Aplicaciones que residen en la región de lotes	Siga los procedimientos operativos de su ubicación para controlar la recuperación y el reinicio.

Deshacer los cambios seleccionados en una unidad de trabajo utilizando puntos de salvaguarda

Los puntos de guardado le permiten deshacer los cambios seleccionados dentro de una unidad de trabajo. Su aplicación puede establecer cualquier número de puntos de salvaguarda y, a continuación, especificar un punto de salvaguarda para indicar los cambios que se han de deshacer en la unidad de trabajo.

Procedimiento

Para deshacer los cambios seleccionados dentro de una unidad de trabajo mediante el uso de puntos de guardado:

1. Establezca cualquier punto de guardado mediante instrucciones SQL SAVEPOINT.

Los puntos de guardado establecen un punto en el que se pueden deshacer los cambios dentro de una unidad de trabajo.

Tenga en cuenta las siguientes capacidades y restricciones al establecer puntos de guardado:

- Puede establecer un punto de guardado con el mismo nombre varias veces dentro de una unidad de trabajo. Cada vez que establezca el punto de guardado, el nuevo valor del punto de guardado sustituirá al antiguo.
- Si no desea que un punto de guardado tenga valores diferentes dentro de una unidad de trabajo, utilice la opción UNIQUE en la instrucción SAVEPOINT. Si una aplicación ejecuta una instrucción

SAVEPOINT con el mismo nombre que un punto de guardado que se definió previamente como único, se produce un error SQL.

- Si establece un punto de guardado antes de ejecutar una instrucción CONNECT, el alcance de ese punto de guardado es el sitio local. Si establece un punto de guardado después de ejecutar la instrucción CONNECT, el alcance de ese punto de guardado es el sitio al que está conectado.
- Cuando los puntos de guardado están activos, lo cual ocurre hasta que se completa la unidad de trabajo, no se puede acceder a sitios remotos utilizando nombres de tres partes o alias para nombres de tres partes. Sin embargo, puede utilizar el acceso DRDA con sentencias CONNECT explícitas.
- No puede utilizar puntos de guardado en transacciones globales, disparadores, funciones definidas por el usuario o procedimientos almacenados que estén anidados dentro de disparadores o funciones definidas por el usuario.

2. Especifique los cambios que desea deshacer dentro de una unidad de trabajo mediante la instrucción SQL ROLLBACK TO SAVEPOINT.

Db2 deshace todos los cambios desde el punto de guardado especificado. Si no especifica un nombre de punto de guardado, Db2 revierte el trabajo al punto de guardado creado más recientemente.

3. Opcional: Si ya no necesita un punto de guardado, elimínelo utilizando la instrucción SQL RELEASE SAVEPOINT.

Recomendación: Si ya no necesita un punto de guardado antes de que finalice una transacción, libérelo. De lo contrario, los puntos de guardado se liberan automáticamente al final de una unidad de trabajo. Liberar puntos de guardado es esencial si necesita utilizar nombres de tres partes para acceder a ubicaciones remotas, ya que no puede realizar esta acción mientras los puntos de guardado estén activos.

ejemplos

Ejemplo: Retroceder al punto de guardado creado más recientemente

Cuando se ejecuta la instrucción ROLLBACK TO SAVEPOINT en el siguiente código, Db2 revierte el trabajo al punto de guardado B.

```
EXEC SQL SAVEPOINT A;  
...  
EXEC SQL SAVEPOINT B;  
...  
EXEC SQL ROLLBACK TO SAVEPOINT;
```

Ejemplo: Establecimiento de puntos de guardado durante el procesamiento distribuido

Una aplicación realiza las siguientes tareas:

1. Establece el punto de guardado C1.
2. Realiza algún procesamiento local.
3. Ejecuta una instrucción CONNECT para conectarse a un sitio remoto.
4. Establece el punto de guardado C2.

Debido a que el punto de guardado C1 se establece antes de que la aplicación se conecte a un sitio remoto, el punto de guardado C1 solo se conoce en el sitio local. Sin embargo, dado que el punto de guardado C2 se establece después de que la aplicación se conecta al sitio remoto, el punto de guardado C2 solo se conoce en el sitio remoto.

Establecer varios puntos de guardado con el mismo nombre

Supongamos que se producen las siguientes acciones dentro de una unidad de trabajo:

1. La aplicación A establece el punto de guardado S.
2. La aplicación A llama al procedimiento almacenado P.
3. El procedimiento almacenado P establece el punto de guardado S.
4. El procedimiento almacenado P ejecuta la siguiente instrucción: ROLLBACK TO SAVEPOINT S

Cuando Db2 ejecuta la instrucción ROLLBACK, Db2 revierte el trabajo al punto de guardado que se estableció en el procedimiento almacenado, porque ese valor es el valor más reciente del punto de guardado S.

Referencia relacionada

[RELEASE SAVEPOINT declaración \(Db2 SQL\)](#)

[ROLLBACK declaración \(Db2 SQL\)](#)

[SAVEPOINT declaración \(Db2 SQL\)](#)

Planificación para la recuperación de espacios de mesa que no están registrados

Para suprimir el registro, puede especificar la opción NOT LOGGED cuando cree o modifique un espacio de tabla. Sin embargo, debido a que los registros se utilizan generalmente en la recuperación, la planificación de la recuperación de los espacios de tabla para los que no se registran los cambios requiere cierta planificación adicional.

Acerca de esta tarea

Aunque puede planificar la recuperación, aún debe tomar algunas medidas correctivas después de cualquier falla del sistema para recuperar los datos y reparar cualquier espacio de tabla afectado. Por ejemplo, si un espacio de tabla que no está registrado estaba abierto para su actualización en el momento en que se produce la terminación de un Db2 , el reinicio posterior coloca ese espacio de tabla en LPL y lo marca con el estado RECOVER-pending. Debe tomar medidas correctivas para borrar el estado RECOVER-pending (Pendiente de RECUPERAR).

Procedimiento

Para planificar la recuperación de espacios de mesa que no están registrados:

1. Asegúrese de que puede recuperar los datos perdidos realizando una de las siguientes acciones:

- Asegúrese de que dispone de una fuente de recuperación de datos que no dependa de un registro para recrear los datos perdidos.
- Limite las modificaciones que no se registran a cambios fácilmente repetibles que se pueden repetir rápidamente.

2. Evite reservar una mesa que no esté registrada en estado RECOVER-pending (pendiente de recuperación).

Las siguientes acciones colocan un espacio de tabla en estado RECOVER-pending (en espera de recuperación):

- Emisión de una sentencia ROLLBACK o ROLLBACK TO SAVEPOINT tras modificar una tabla en un espacio de tablas que no está registrado.
- Causar duplicados de claves o violaciones de integridad referenciales al modificar un espacio de tabla que no está registrado.

Si el espacio de la mesa se coloca en estado RECOVER-pending (En espera de recuperación), no estará disponible hasta que lo arregle manualmente.

3. Para los espacios de tabla que no están registrados y tienen espacios de tabla LOB o XML asociados, tome copias de imagen como un conjunto de recuperación.

Esta acción garantiza que el espacio de la tabla base y todos los espacios de tabla LOB o XML asociados se copien en el mismo momento. Una operación posterior de RECUPERAR EN LA ÚLTIMA COPIA para todo el conjunto da como resultado datos coherentes en todo el espacio de la tabla base y en todos los espacios de tabla LOB y XML asociados.

Tareas relacionadas

[Borrado del estado pendiente de recuperación \(RECP\) \(Db2 Administration Guide\)](#)

Referencia relacionada

[RECOVER \(Db2 Utilities\)](#)

Diseño de la aplicación para acceder a datos distribuidos

Puede diseñar aplicaciones que acceden a datos en otro sistema de gestión de base de datos (DBMS) que no sea el sistema local. También debe tener en cuenta las limitaciones y recomendaciones para esos problemas cuando diseña las aplicaciones.

Procedimiento

Para diseñar su aplicación para acceder a datos distribuidos:

1. Asegúrese de que se ha concedido la autorización de identificación adecuada en el servidor remoto para conectarse a ese servidor y utilizar sus recursos.
2. Si su aplicación contiene instrucciones SQL que se ejecutan en el solicitante, incluya en el solicitante un módulo de solicitud de base de datos (DBRM) que esté vinculado directamente a un paquete que esté incluido en la lista de paquetes del plan.
3. Copie el paquete del solicitante en cualquier servidor remoto al que acceda la aplicación mediante un comando bind package copy e incluya los paquetes remotos en la lista de paquetes del plan de la aplicación.

Recomendación: Especifique un asterisco (*) en lugar de un nombre específico en el nombre de ubicación de cualquier entrada de paquete de un plan para que el plan no tenga que ser reenviado cada vez que la aplicación acceda a una nueva ubicación o se acceda a una ubicación diferente.

4. Para aplicaciones TSO y por lotes que actualizan datos en un servidor remoto, asegúrese de que se cumpla una de las siguientes condiciones:
 - No existen otras conexiones.
 - Todas las conexiones existentes son a servidores que están restringidos a operaciones de solo lectura.

Restricción: Si no se cumple ninguna de estas condiciones, la aplicación se limita a operaciones de solo lectura.

Si se cumple una de estas condiciones, y si la primera conexión en una unidad lógica de trabajo es a un servidor que admite la confirmación en dos fases, ese servidor y todos los servidores que admiten la confirmación en dos fases pueden actualizar los datos. Sin embargo, si la primera conexión es a un servidor que no admite la confirmación en dos fases, solo ese servidor podrá actualizar los datos.

5. Para los programas que acceden al menos a un sistema restringido, asegúrese de que su programa no infringe ninguna de las limitaciones de acceso a sistemas restringidos.

Un sistema restringido es un DBMS que no implementa el procesamiento de confirmación en dos fases.

El acceso a sistemas restringidos tiene las siguientes limitaciones:

- Para programas que acceden a CICS o IMS, no puede actualizar datos en sistemas restringidos.
- Dentro de una unidad de trabajo, no se puede actualizar un sistema restringido después de actualizar un sistema no restringido.
- Dentro de una unidad de trabajo, si actualiza un sistema restringido, no puede actualizar ningún otro sistema.

Si está accediendo a una combinación de sistemas, algunos de los cuales podrían estar restringidos, puede realizar las siguientes acciones:

- Leer desde cualquiera de los sistemas en cualquier momento.
- Actualizar cualquiera de los sistemas muchas veces en una unidad de trabajo.
- Actualizar muchos sistemas, incluyendo CICS o IMS, en una unidad de trabajo, siempre que ninguno de ellos sea un sistema restringido. Si el primer sistema que actualiza en una unidad de trabajo

no está restringido, cualquier intento de actualizar un sistema restringido en esa unidad de trabajo devuelve un error.

- Actualizar un sistema restringido en una unidad de trabajo, siempre que no intente actualizar ningún otro sistema en la misma unidad de trabajo. Si el primer sistema que actualiza en una unidad de trabajo está restringido, cualquier intento de actualizar cualquier otro sistema en esa unidad de trabajo devuelve un error.

Conceptos relacionados

[Fase 6: Acceso a los datos en un sitio remoto \(Db2 Installation and Migration\)](#)

Tareas relacionadas

[Mejora del rendimiento para las aplicaciones que acceden a datos distribuidos \(Db2 Performance\)](#)

Servidores remotos y datos distribuidos

Los datos distribuidos son datos que residen en un sistema de gestión de bases de datos (DBMS) distinto de su sistema local. El DBMS local es el sistema en el que se vincula el plan de aplicación. Los demás DBMS son remotos.

Si solicita servicios de un DBMS remoto, ese DBMS es un servidor y su sistema local es un solicitante o cliente.

Su aplicación puede conectarse a muchos DBMS a la vez; el que está realizando el trabajo en ese momento es *el servidor actual*. Cuando el sistema local está realizando un trabajo, también se le llama servidor actual.

Un servidor remoto puede ser físicamente remoto, o puede ser otro subsistema del mismo sistema operativo en el que se ejecuta su DBMS local. Un servidor remoto puede ser una instancia de Db2 for z/OS, o puede ser una instancia de otro producto.

Un DBMS, ya sea local o remoto, es conocido por su sistema de Db2 s por su nombre de ubicación. El nombre de ubicación de un DBMS remoto se registra en la base de datos de comunicaciones.

Tareas relacionadas

[Elección de nombres para el subsistema local \(Db2 Installation and Migration\)](#)

Preparación para actualizaciones coordinadas de dos o más fuentes de datos

Se coordinan dos o más actualizaciones si todas deben confirmarse o revertirse en la misma unidad de trabajo.

Acerca de esta tarea

Esta situación es común en la banca. Supongamos que se resta una cantidad de una cuenta y se añade a otra. Las dos acciones deben, o bien confirmarse, o bien revertirse al final de la unidad de trabajo.

Procedimiento

Asegúrese de que todos los sistemas a los que accede su programa implementan el procesamiento de confirmación en dos fases. Este procesamiento garantiza que las actualizaciones de dos o más DBMS se coordinen automáticamente.

Por ejemplo, Db2 y IMS, y Db2 y CICS, implementan conjuntamente un proceso de compromiso de dos fases. Puede actualizar una base de datos IMS y una tabla Db2 en la misma unidad de trabajo. Si se produce un fallo del sistema o de la comunicación entre la realización del trabajo en IMS y en Db2, los dos programas restauran los dos sistemas a un punto coherente cuando se reanuda la actividad.

No se pueden realizar verdaderas actualizaciones coordinadas dentro de un DBMS que no implemente el procesamiento de confirmación en dos fases, porque Db2 le impide actualizar dicho DBMS y cualquier otro sistema dentro de la misma unidad de trabajo. En este contexto, la actualización incluye

las sentencias INSERT, UPDATE, MERGE, DELETE, CREATE, ALTER, DROP, GRANT, REVOKE, RENAME, COMMENT y LABEL.

Sin embargo, si no puede implementar el procesamiento de confirmación en dos fases en todos los sistemas a los que accede su programa, puede simular el efecto de las actualizaciones coordinadas realizando las siguientes acciones:

- a. Actualizar un sistema y confirmar ese trabajo.
 - b. Actualizar el segundo sistema y confirmar su funcionamiento.
 - c. Asegúrese de que su programa tiene código para deshacer la primera actualización si se produce un fallo después de que se haya realizado la primera actualización y antes de que se realice la segunda.
- No existe ninguna disposición automática para que los dos sistemas vuelvan a un punto coherente.

Conceptos relacionados

[Proceso de confirmación de dos fases \(Db2 Administration Guide\)](#)

Forzar reglas de sistema restringidas en su programa

Un sistema restringido es un DBMS que no implementa el procesamiento de confirmación en dos fases. Estos sistemas tienen una serie de restricciones de actualización. Puede restringir su programa completamente a las reglas para estos sistemas restringidos, independientemente de si el programa está accediendo a sistemas restringidos o no restringidos.

Acerca de esta tarea

El acceso a sistemas restringidos tiene las siguientes limitaciones:

- Para programas que acceden a CICS o IMS, no puede actualizar datos en sistemas restringidos.
- Dentro de una unidad de trabajo, no se puede actualizar un sistema restringido después de actualizar un sistema no restringido.
- Dentro de una unidad de trabajo, si actualiza un sistema restringido, no puede actualizar ningún otro sistema.

Procedimiento

Cuando prepare su programa, especifique la opción de procesamiento SQL CONNECT(1).

Esta opción aplica las reglas de la sentencia CONNECT de tipo 1.

Restricción: No utilice paquetes que estén precompilados con la opción CONNECT(1) y paquetes que estén precompilados con la opción CONNECT(2) en la misma lista de paquetes. La primera sentencia CONNECT que ejecuta su programa determina qué reglas están en vigor para toda la ejecución: tipo 1 o tipo 2. Si su programa intenta ejecutar una instrucción CONNECT posterior que está precompilada con el otro tipo, Db2 devuelve un error.

Conceptos relacionados

[Opciones para el proceso de sentencias de SQL](#)

Utilice las opciones de proceso de SQL para especificar cómo el precompilador de Db2 y el coprocesador de Db2 interpretan y procesan la entrada y cómo presentan la salida.

Capítulo 2. Conectarse a Db2 desde su programa de aplicación

Los programas de aplicación se comunican con Db2 a través de un archivo adjunto. Debe invocar un mecanismo de conexión, implícita o explícitamente, antes de que su programa pueda interactuar con Db2.

Acerca de esta tarea

Puede utilizar los siguientes servicios de archivos adjuntos en un z/OS entorno:

Recurso de conexión de CICS

Utilice esta función para acceder a Db2 desde CICS programas de aplicación.

Recurso de conexión de IMS

Utilice esta función para acceder a Db2 desde IMS programas de aplicación.

Opción de tiempo compartido (TSO) servicio de archivo adjunto

Utilice esta función en un entorno TSO o por lotes para comunicarse con un subsistema de Db2 e local. Esta función invoca el procesador de comandos DSN.

recurso de conexión de lamada (CAF)

Utilice este servicio como alternativa al servicio de adjuntar TSO cuando su aplicación necesite un control estricto sobre el entorno de la sesión.

Recurso de conexión de Resource Recovery Services (RRSAF)

Utilice esta función para procedimientos almacenados que se ejecuten en un espacio de direcciones establecido por WLM o como alternativa al CAF. RRSAF proporciona apoyo a z/OS RRS como coordinador de recuperación y apoya otras capacidades no presentes en CAF

Para aplicaciones distribuidas, utilice el servicio de datos distribuidos (DDF).

Requisito: Asegúrese de que cualquier aplicación que solicite servicios de Db2 cumpla con las siguientes características de entorno, independientemente del servicio de archivos adjuntos que utilice:

- La aplicación debe estar ejecutándose en modo TCB. El modo SRB no es compatible.
- Una tarea de aplicación no puede tener activas rutinas de recuperación funcional (FRR) de tareas desbloqueadas habilitadas (EUT) al solicitar servicios de Db2 . Si un EUT FRR está activo, la recuperación funcional de la red (Db2) puede fallar y su aplicación puede sufrir algunos fallos impredecibles.
- No pueden estar activos simultáneamente diferentes servicios de conexión dentro del mismo espacio de direcciones. En concreto, existen los siguientes requisitos:
 - Una aplicación no debe utilizar CAF o RRSAF en un CICS o IMS espacio de direcciones.
 - Una aplicación que se ejecuta en un espacio de direcciones que tiene una conexión CAF a Db2 no puede conectarse a Db2 mediante RRSAF.
 - Una aplicación que se ejecuta en un espacio de direcciones que tiene una conexión RRSAF a Db2 no puede conectarse a Db2 mediante CAF.
 - Una aplicación no puede invocar la z/OS Macro AXSET después de ejecutar la llamada CAF CONNECT y antes de ejecutar la llamada CAF DISCONNECT.
- Un servicio de envío no puede iniciar otro. Por ejemplo, su aplicación CAF o RRSAF no puede utilizar DSN, y un subcomando DSN RUN no puede llamar a su aplicación CAF o RRSAF.
- Los módulos de interfaz de lenguaje para CAF y RRSAF, DSNALI y DSNRLI, se envían con los atributos de vinculación AMODE(31) y RMODE(ANY). Si sus aplicaciones cargan CAF o RRSAF por debajo de la línea de 16 MB, debe volver a vincular y editar DSNALI o DSNRLI.

Conceptos relacionados

[Recursos de conexión de Db2 \(Introducción a Db2 para z/OS\)](#)

[Recurso de datos distribuidos \(Introducción a Db2 para z/OS\)](#)

Invocación del recurso de conexión de llamada

Invoque el recurso de conexión de llamada (CAF) cuando desee que su programa de aplicación establezca y controle su propia conexión con Db2. Las aplicaciones que utilizan CAF pueden controlar explícitamente el estado de sus conexiones en Db2 utilizando funciones de conexión que CAF proporciona.

Antes de empezar

Antes de poder invocar CAF, realice las siguientes acciones:

- Asegúrese de que la interfaz de lenguaje CAF (DSNALI) esté disponible.
- Asegúrese de que su solicitud cumple los requisitos de los programas que acceden a CAF.
- Asegúrese de que su aplicación cumple las características generales de entorno para conectarse a Db2.
- Asegúrese de estar familiarizado con los siguientes z/OS conceptos e instalaciones:
 - Convenciones de vinculación de macros CALL y módulos estándar
 - Opciones de direccionamiento y residencia del programa (AMODE y RMODE)
 - Crear y controlar tareas; multitarea
 - Instalaciones de recuperación funcional como ESTAE, ESTAI y FRR
 - Eventos asíncronos y salidas de atención TSO (STAX)
 - Técnicas de sincronización como WAIT/POST.

Acerca de esta tarea

Las aplicaciones que utilizan CAF pueden estar escritas en lenguaje ensamblador, C, COBOL, Fortran, y PL/I. Al elegir un idioma para codificar su aplicación, tenga en cuenta las siguientes restricciones:

- Si necesita utilizar z/OS macros (ATTACH, WAIT, POST, etc.), utilice un lenguaje de programación que las admita o incorpórelas en módulos escritos en lenguaje ensamblador.
- La función CAF TRANSLATE no está disponible en Fortran. Para utilizar esta función, codifíquela en una rutina que esté escrita en otro lenguaje y, a continuación, llame a esa rutina desde Fortran.

Recomendaciones: Para IMS y DSN, tenga en cuenta las siguientes recomendaciones:

- Para IMS solicitudes por lotes, no utilice CAF. En su lugar, utilice el soporte de lotes DL/I de Db2 . Aunque es posible que IMS las aplicaciones por lotes accedan a las bases de datos de Db2 a través de CAF, ese método no coordina el compromiso de trabajo entre los IMS y los sistemas de Db2 .
- Para las aplicaciones DSN, no utilice CAF a menos que proporcione un controlador de aplicaciones para gestionar la aplicación DSN y sustituir las funciones DSN necesarias. También es posible que tenga que cambiar la aplicación para comunicar correctamente los fallos de conexión al controlador. Ejecutar aplicaciones DSN con CAF no es ventajoso, y la pérdida de servicios DSN puede afectar al buen funcionamiento de su programa.

Procedimiento

Realice una de las acciones siguientes:

- Invoque explícitamente CAF incluyendo en su programa sentencias CALL DSNALI con las opciones adecuadas.

La primera opción es una función de conexión de CAF, que describe la acción que desea que CAF realice. El efecto de cualquier función depende en parte de las funciones que el programa ya haya ejecutado.

Requisito: Para las aplicaciones C y PL/I, también debe incluir en su programa las directivas del compilador que se enumeran en la siguiente tabla, porque DSNALI es un programa en lenguaje ensamblador.

Tabla 3. Directivas de compilación para incluir en aplicaciones C y PL/I que contengan sentencias CALL DSNALI

Idioma	Directiva del compilador para incluir
C	#pragma linkage(dsnali, OS)
C++	extern "OS" { int DSNALI(char * functn, ...); }
PL/I	DCL DSNALI ENTRY OPTIONS(ASM,INTER,RETCODE);

- invoque implícitamente CAF incluyendo sentencias SQL o llamadas IFI en su programa tal como lo haría en cualquier programa. La instalación de CAF establece las conexiones con Db2 con los valores predeterminados para el nombre del subsistema y el nombre del plan.

Restricción: Si su programa puede realizar su primera llamada SQL desde diferentes módulos con diferentes DBRM, no puede utilizar un nombre de plan predeterminado y, por lo tanto, no puede invocar implícitamente CAF. En su lugar, debe invocar explícitamente CAF utilizando la función OPEN.

Requisito: Si su aplicación incluye llamadas SQL e IFI, debe emitir al menos una llamada SQL antes de emitir cualquier llamada IFI. Esta acción garantiza que su aplicación utilice el plan correcto.

Aunque no se recomienda hacerlo, puede ejecutar aplicaciones DSN existentes con CAF permitiéndoles realizar conexiones implícitas a Db2. Para que Db2 realice una conexión implícita correctamente, el nombre del plan para la aplicación debe ser el mismo que el nombre de miembro del módulo de solicitud de base de datos (DBRM) que Db2 produjo cuando precompiló el programa fuente que contiene la primera llamada SQL. También debe sustituir el módulo de interfaz de idioma DSNALI por el módulo de interfaz de idioma TSO, DSNELI.

Si no especifica los parámetros de código de retorno y código de motivo en sus llamadas CAF o invoca CAF implícitamente, CAF pone un código de retorno en el registro 15 y un código de motivo en el registro 0.

Para determinar si una conexión implícita se ha realizado correctamente, el programa de aplicación debe examinar los códigos de retorno y de motivo inmediatamente después de la primera instrucción SQL ejecutable en el programa de aplicación realizando una de las siguientes acciones:

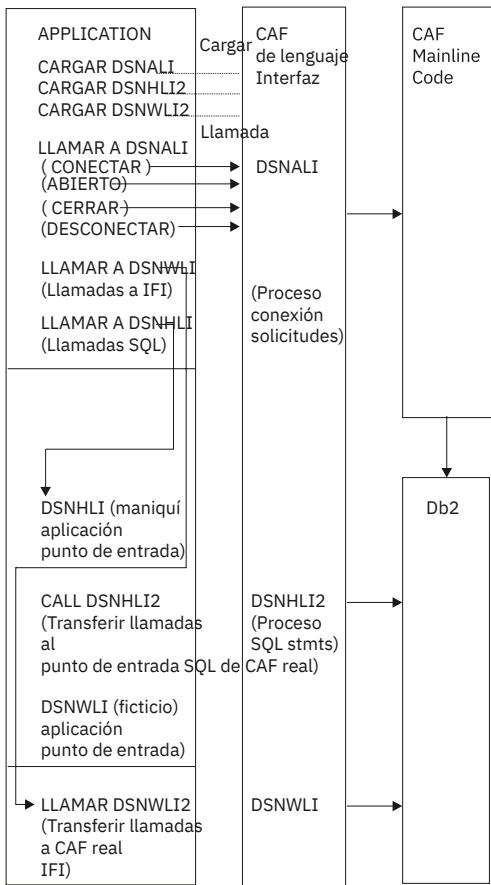
- Examinar los registros 0 y 15 directamente.
- Examinar el SQLCA y, si el SQLCODE es -991, obtener el código de retorno y el motivo del texto del mensaje. El código de devolución es el primer token y el código de motivo es el segundo token.

Si la conexión implícita se ha realizado correctamente, la aplicación puede examinar el SQLCODE para la primera y las siguientes sentencias SQL.

ejemplos

Ejemplo de configuración de CAF

La siguiente figura muestra un ejemplo conceptual de invocar y utilizar CAF. La aplicación contiene instrucciones para cargar DSNALI, DSNHLI2 y DSNWLI2. La aplicación accede a Db2 mediante la interfaz de lenguaje CAF. Llama a DSNALI para gestionar las solicitudes de CAF, a DSNWLI para gestionar las llamadas de IFI y a DSNHLI para gestionar las llamadas de SQL.



Programas de muestra que utilizan CAF

Puede encontrar un programa ensamblador de muestra (DSN8CA) y un programa COBOL de muestra (DSN8CC) que utilizan *el prefijo* de biblioteca CAF enSDNSAMP. Una aplicación PL/I (DSN8SPM) llama a DSN8CA, y una aplicación COBOL (DSN8SCM) llama a DSN8CC.

Conceptos relacionados

Ejemplos de aplicaciones suministrados con Db2 for z/OS

Db2 proporciona ejemplos de aplicaciones para ayudarle con las técnicas de programación e Db2 es y las prácticas de codificación dentro de cada uno de los cuatro entornos: batch, TSO, IMS, y CICS. Las aplicaciones de ejemplo contienen varias aplicaciones que pueden aplicarse a la gestión de una empresa.

Referencia relacionada

Funciones de conexión CAF

Una función de conexión CAF especifica la acción que desea que realice CAF. Especifique estas funciones cuando invoque CAF a través de sentencias CALL DSNALI.

Recurso de conexión de llamada

Un recurso de conexión habilita programas para comunicarse con Db2. La función de adjuntar llamadas (CAF) proporciona dicha conexión para programas que se ejecutan en z/OS lotes, en primer plano TSO y en segundo plano TSO. El CAF debe controlar de forma estricta el entorno de la sesión.

Un programa que utiliza CAF puede realizar las siguientes acciones:

- Acceda a Db2 desde z/OS espacios de direcciones donde TSO, IMS, o no existen CICS no existen.
 - Acceder a Db2 desde múltiples z/OS tareas en un espacio de direcciones.
 - Acceda al IFI (Db2).
 - Ejecutar cuando Db2 no esté disponible.

Restricción: La aplicación no puede ejecutar SQL cuando Db2 no está disponible.

- Ejecutar con o sin el programa de monitorización de terminales (TMP) TSO.
- Ejecutarse sin ser una subtarea del procesador de comandos DSN o de cualquier código e Db2 .
- Ejecutar por encima o por debajo de la línea de 16 MB. (El código CAF se encuentra debajo de la línea)
- Establecer una conexión explícita con Db2, a través de una interfaz de LLAMADA, con control sobre el estado exacto de la conexión.
- Establecer una conexión implícita con Db2, mediante el uso de sentencias SQL o llamadas IFI sin llamar primero a CAF, con un nombre de plan predeterminado y un identificador de subsistema.
- Verifique que la aplicación esté utilizando la versión correcta de Db2.
- Suministrar bloques de control de eventos (ECB), para que Db2 los publique, que indiquen el inicio o la finalización.
- Interceptar códigos de retorno, códigos de motivo y códigos de error de Db2 y traducirlos en mensajes.

Cualquier tarea en un espacio de direcciones puede establecer una conexión con Db2 a través de CAF. Solo puede existir una conexión por cada bloque de control de tareas (TCB). Una solicitud de servicio de Db2 emitida por un programa que se ejecuta en una tarea determinada está asociada a la conexión de esa tarea con Db2. La solicitud de servicio funciona independientemente de cualquier actividad de otra acción en cualquier otra tarea.

Cada tarea conectada puede ejecutar un plan. Varias tareas en un único espacio de direcciones pueden especificar el mismo plan, pero cada instancia de un plan se ejecuta de forma independiente de las demás. Una tarea puede finalizar su plan y ejecutar un plan diferente sin interrumpir por completo su conexión con Db2.

CAF no genera estructuras de tareas.

Cuando diseñe su aplicación, tenga en cuenta que el uso de múltiples conexiones simultáneas puede aumentar la posibilidad de bloqueos y de contención de recursos e Db2 .

Un servicio de rastreo proporciona mensajes de diagnóstico que ayudan a depurar programas y diagnosticar errores en el código CAF. En particular, los intentos de utilizar CAF de forma incorrecta provocan mensajes de error en el flujo de seguimiento.

Restricción: CAF no proporciona salidas de procesamiento de atención ni rutinas de recuperación funcional. Puede proporcionar cualquier atención de manejo y recuperación funcional que su aplicación necesite, pero debe utilizar rutinas de recuperación de tipo ESTAE/ESTAI y no rutinas FRR de tareas habilitadas y desbloqueadas (EUT).

Propiedades de las conexiones CAF

La función de llamada adjunta (CAF) permite que los programas se comuniquen con Db2.

La conexión que CAF establece con Db2 tiene las propiedades básicas que se enumeran en la siguiente tabla.

Tabla 4. Propiedades de las conexiones CAF

Propiedad	Valor	Comentarios
Nombre de conexión	DB2CALL	Puede utilizar el comando DISPLAY THREAD para mostrar una lista de las aplicaciones CAF que tienen el nombre de conexión DB2CALL.

Tabla 4. Propiedades de las conexiones CAF (continuación)

Propiedad	Valor	Comentarios
Tipo de conexión	BATCH	Las conexiones BATCH utilizan un proceso de confirmación de fase única que está coordinado por Db2. Los programas de aplicación también pueden controlar cuándo se confirman las sentencias mediante las sentencias SQL COMMIT y ROLLBACK.
ID de autorización	ID de autorización que están asociados con el espacio de direcciones	Db2 establece identificaciones de autorización para la conexión de cada tarea cuando procesa dicha conexión. Para el tipo de conexión BATCH, Db2 crea una lista de ID de autorización basada en el ID de autorización asociado al espacio de direcciones. Esta lista es la misma para todas las tareas. Una ubicación puede proporcionar una rutina de salida de autorización de conexión de Db2 para cambiar la lista de ID.
Ámbito	CAF procesa las conexiones como si cada tarea estuviera completamente aislada. Cuando una tarea solicita una función, el CAF pasa las funciones a Db2 y desconoce el estado de conexión de otras tareas en el espacio de direcciones. Sin embargo, el programa de aplicación y el subsistema de Db2 a conocen el estado de conexión de múltiples tareas en un espacio de direcciones.	none

Si una tarea conectada finaliza normalmente antes de que la función CLOSE desasigne el plan, Db2 confirma cualquier cambio en la base de datos que el hilo haya realizado desde el último punto de confirmación. Si una tarea conectada falla antes de que la función CLOSE desasigne el plan, Db2 revierte cualquier cambio en la base de datos desde el último punto de confirmación. En cualquier caso, Db2 desasigna el plan, si es necesario, y termina la conexión de la tarea antes de permitir que la tarea termine.

Si Db2 se cierra de forma anormal mientras se está ejecutando una aplicación, la aplicación se revierte al último punto de confirmación. Si Db2 se cierra mientras procesa una solicitud de confirmación, Db2 confirma o revierte cualquier cambio en el siguiente reinicio. La acción tomada depende del estado de la solicitud de confirmación cuando se termina Db2 .

Conceptos relacionados

Rutinas de conexión y rutinas de inicio de sesión (Managing Security)

Atención rutinas de salida para CAF

Una rutina de salida de atención le permite recuperar el control de Db2 durante solicitudes erróneas o de larga duración. La función de llamada de archivo adjunto (CAF) no tiene rutinas de salida de atención, pero puede proporcionar las suyas propias si es necesario.

Una rutina de salida de atención funciona separando el TCB que actualmente está esperando que se complete una solicitud SQL o IFI. Después de que se separe el TCB, el sistema de control de terminación (Db2) detecta el error resultante y realiza el procesamiento de terminación para esa tarea. El proceso de cancelación incluye cualquier reversión necesaria de las transacciones.

Puede proporcionar sus propias rutinas de salida de atención. Sin embargo, es posible que su rutina no se controle si solicita atención mientras se ejecuta un código de error de desbloqueo de tarea habilitada (Db2, EUT), ya que Db2 utiliza rutinas de recuperación funcional (FRR) de desbloqueo de tarea habilitada (EUT).

Rutinas de recuperación para CAF

Puede utilizar rutinas de recuperación de fallos y rutinas de recuperación funcional (FRR) para gestionar errores inesperados. Una rutina de recuperación de fallos controla lo que ocurre cuando se produce un fallo mientras Db2 tiene el control. Una rutina de recuperación funcional puede obtener información sobre errores de programa y recuperarse de ellos.

El CAF no tiene rutinas de recuperación de errores, pero usted puede proporcionar las suyas propias. Cualquier rutina de recuperación de errores que proporcione debe utilizar indicadores de seguimiento para determinar si se ha producido un error durante el procesamiento de un Db2. Si se produce un fallo mientras Db2 tiene el control, la rutina de recuperación puede realizar una de las siguientes acciones:

- Permitir que finalice la tarea. No vuelva a probar el programa. Db2 detecta la finalización de la tarea y finaliza el hilo con el parámetro ABRT. Se pierden todos los cambios de la base de datos hasta el último punto de sincronización o punto de confirmación.

Esta acción es la única que puede realizar para las ausencias que se deben al comando CANCEL o a DETACH. No puede utilizar instrucciones SQL adicionales. Si intenta ejecutar otra instrucción SQL desde el programa de aplicación o su rutina de recuperación, recibirá un código de retorno de +256 y un código de motivo de X'00F30083'.

- En una rutina ESTAE, emita una llamada a la función CLOSE con el parámetro ABRT seguido de una llamada a la función DISCONNECT. La rutina de salida ESTAE puede volver a intentarlo para que no tenga que restablecer la tarea de aplicación.

Los FRR deben cumplir los siguientes requisitos y restricciones:

- Solo puede utilizar FRR de tareas desbloqueadas habilitadas (EUT) en sus rutinas que llamen a Db2. Las rutinas estándar de recuperación funcional (FRR, por sus siglas en inglés) z/OS rutinas de recuperación funcional (FRR) se aplican únicamente al código que se ejecuta en modo de bloque de solicitud de servicio (SRB), y Db2 no admite llamadas desde rutinas en modo SRB.
- No tenga un EUT FRR activo cuando utilice CAF, procese solicitudes SQL o llame a IFI. Con z/OS, si tiene un EUT FRR activo, todas las solicitudes de Db2 fallan, incluida la solicitud inicial de CONNECT u OPEN. Las solicitudes fallan porque Db2 siempre crea un ESTA de tipo ARR, y z/OS no permite la creación de ESTAEs de tipo ARR cuando hay un FRR activo.
- Un EUT FRR no puede reintentar las solicitudes de e Db2. Un reintento EUT FRR omite las rutinas ESTAE de Db2. La siguiente solicitud de Db2, de cualquier tipo, incluida una solicitud de DESCONEXIÓN, falla con un código de retorno de +256 y un código de motivo de X'00F30050'.

Disponibilidad de la interfaz de lenguaje CAF (DSNALI)

Antes de invocar el recurso de conexión de llamada (CAF), primero debe hacer que DSNALI esté disponible.

Acerca de esta tarea

Parte de CAF es un módulo de carga de datos (Db2) llamado DSNALI, que también se conoce como interfaz de lenguaje CAF. DSNALI tiene los alias DSNHLI2 y DSNWLI2. El módulo tiene cinco puntos de entrada: DSNALI, DSNHLI, DSNHLI2, DSNWLI y DSNWLI2. Estos puntos de entrada cumplen las siguientes funciones:

- El punto de entrada DSNALI gestiona las solicitudes de servicio de conexión explícitas de Db2.

- DSNHLI y DSNHLI2 gestionan las llamadas SQL. Utilice DSNHLI si su programa de aplicación edita enlazado DSNALI. Utilice DSNHLI2 si su programa de aplicación carga DSNALI.
- DSNWLI y DSNWLI2 gestionan las llamadas de IFI. Utilice DSNWLI si su programa de aplicación edita enlaces DSNALI. Utilice DSNWLI2 si su programa de aplicación carga DSNALI.

Procedimiento

Para que DSNALI esté disponible:

1. Decida cuál de los siguientes métodos desea utilizar para que DSNALI esté disponible:

- Emisión explícita de solicitudes LOAD cuando se ejecuta el programa.

Al cargar explícitamente el módulo DSNALI, aísla de forma beneficiosa el mantenimiento de su aplicación del futuro IBM mantenimiento de la interfaz de idioma. Si cambia el idioma de la interfaz, es probable que el cambio no afecte a su módulo de carga.

- Incluir el módulo DSNALI en su módulo de carga cuando edite su programa en modo enlace.

Si no necesita llamadas explícitas a DSNALI para funciones CAF, la edición de enlaces DSNALI en su módulo de carga tiene algunas ventajas. Cuando se incluye DSNALI durante la edición del enlace, no es necesario codificar un punto de entrada ficticio DSNHLI en el programa ni especificar la opción ATTACH del precompilador. El módulo DSNALI contiene un punto de entrada para DSNHLI, que es idéntico a DSNHLI2, y un punto de entrada DSNWLI, que es idéntico a DSNWLI2.

Una desventaja de editar el enlace DSNALI en su módulo de carga es que cualquier IBM mantenimiento de DSNALI requiere una nueva edición de enlace de su módulo de carga.

De forma alternativa, si utiliza conexiones explícitas a través de CALL DSNALI, puede editar enlaces de su programa con DSNULI, la interfaz de lenguaje universal.

2. Dependiendo del método que haya elegido en el paso 1, realice una de las siguientes acciones:

- **Si desea emitir explícitamente solicitudes LOAD cuando se ejecute su programa:**

En su programa, emita z/OS Cargar solicitudes de servicio para los puntos de entrada DSNALI y DSNHLI2. Si utiliza los servicios de IFI, también debe cargar DSNWLI2. Las direcciones de punto de entrada que devuelve LOAD se guardan para su uso posterior con la macro CALL. Indique a Db2 qué punto de entrada utilizar de una de las dos formas siguientes:

- Especifique la opción del precompilador ATTACH(CAF).

Esta opción hace que Db2 genere llamadas que especifiquen el punto de entrada DSNHLI2.

Restricción: No puede utilizar esta opción si su solicitud está escrita en un Fortran.

- Cree un punto de entrada ficticio llamado DSNHLI dentro de su módulo de carga.

Si no especifica la opción del precompilador ATTACH, el precompilador de Db2 genera llamadas al punto de entrada DSNHLI para cada solicitud SQL. El precompilador no conoce las diferentes funciones de adjuntar archivos (Db2) y es independiente de ellas. Cuando las llamadas generadas por el precompilador Db2 pasan el control a DSNHLI, su código que corresponde al punto de entrada ficticio debe conservar la lista de opciones que se pasó en R1 y especificar la misma lista de opciones cuando llame a DSNHLI2.

- **Si desea incluir el módulo DSNALI en su módulo de carga cuando edite su programa mediante enlace:**

Incluya DSNALI en su módulo de carga durante un paso de edición de enlace. El módulo debe estar en una biblioteca de módulos de carga, que se incluye en la concatenación SYSLIB o en otra biblioteca INCLUDE que se define en el JCL del editor de enlaces. Debido a que todos los módulos de interfaz de lenguaje contienen una declaración de punto de entrada para DSNHLI, el editor de enlaces JCL debe contener una instrucción de control del editor de enlaces INCLUDE para DSNALI; por ejemplo, INCLUDE SYSLIB(DSNALI). Al codificar estas opciones, evitará seleccionar sin querer el módulo de interfaz de idioma incorrecto.

Conceptos relacionados

Variables de referencia de archivo LOB

En una aplicación de host, puede utilizar una variable de referencia de archivo para insertar un valor LOB o XML desde un archivo en una tabla de Db2. También puede utilizar una variable de referencia de archivo para seleccionar un valor LOB o XML desde una tabla de Db2 en un archivo.

Ejemplos de invocación de CAF

El recurso de conexión de llamadas (CAF) permite a los programas comunicarse con Db2. Si invoca explícitamente CAF en el programa, puede utilizar las funciones de conexión CAF para controlar el estado de la conexión.

“Interfaz de lenguaje universal (DSNULI)” en la página 121

El subcomponente de la interfaz de lenguaje universal (DSNULI) determina el entorno en tiempo de ejecución y carga y se bifurca dinámicamente al modelo de interfaz de lenguaje adecuado.

Tareas relacionadas

Edición de enlaces de una aplicación con DSNULI

Para crear un único módulo de carga que se pueda utilizar en más de un entorno de conexión, puede editar el enlace de su programa o procedimiento almacenado con el módulo de interfaz de lenguaje universal (DSNULI) en lugar de con uno de los módulos de interfaz de lenguaje específicos del entorno (DSNELI, DSNALI, DSNRLI, DSNCLI o DFSLI000).

Cómo guardar almacenamiento al manipular LOB mediante localizadores LOB

Los localizadores LOB le permiten manipular datos LOB sin recuperar datos de la tabla de Db2. Utilizando localizadores, evita la necesidad de asignar la gran cantidad de almacenamiento necesario para que las variables host contengan datos LOB.

Requisitos para programas que utilizan CAF

El recurso de conexión de llamadas (CAF) permite a los programas comunicarse con Db2. Antes de invocar CAF en su programa, asegúrese de que su programa cumple con todos los requisitos para utilizar CAF.

Cuando escriba programas que utilicen CAF, asegúrese de que cumplan los siguientes requisitos:

- El programa tiene en cuenta el tamaño del código CAF. El código CAF requiere unos 16 KB de almacenamiento virtual por espacio de direcciones y 10 KB adicionales por cada TCB que utilice CAF.
- Si su entorno local intercepta y reemplaza el z/OS LOAD SVC que utiliza CAF, debe asegurarse de que su versión de LOAD gestione las cadenas de elementos de la lista de carga (LLE) y de entrada de directorio de contenidos (CDE) como la macro estándar z/OS Macro LOAD estándar. CAF utiliza z/OS SVC LOAD para cargar dos módulos como parte de la inicialización después de su primera solicitud de servicio. Ambos módulos se cargan en un almacenamiento protegido contra extracciones que tiene la clave de protección de pasos de trabajo.
- Si utiliza CAF desde IMS lote, debe escribir datos en un solo sistema en cualquier unidad de trabajo. Si escribe en ambos sistemas dentro de la misma unidad, un fallo del sistema puede dejar las dos bases de datos incoherentes sin posibilidad de recuperación automática. Para finalizar una unidad de trabajo en Db2, ejecute la instrucción SQL COMMIT. Para finalizar una unidad de trabajo en IMS, emita el comando SYNCPOINT.

Puede preparar programas de aplicación para que se ejecuten en CAF de forma similar a como prepara aplicaciones para que se ejecuten en otros entornos, como CICS, IMS y TSO. Puede preparar una solicitud de CAF en el entorno por lotes o mediante el proceso de preparación del programa Db2 . Puede utilizar el sistema de preparación de programas a través de DB2I o a través de DSNH CLIST.

Tareas relacionadas

Preparación de una aplicación para ejecutarse en Db2 for z/OS

Para preparar y ejecutar aplicaciones que contengan sentencias SQL estáticas incrustadas o sentencias SQL dinámicas, debe procesar, compilar, editar vínculos y vincular las sentencias SQL.

Cómo modifica CAF el contenido de los registros

Si no especifica los parámetros de código de retorno y de código de razón en las llamadas de función de CAF o si invoca a CAF implícitamente, CAF coloca un código de retorno en el registro 15 y un código de razón en el registro 0. El contenido de los registros 2 a 14 se conserva en todas las llamadas.

La siguiente tabla enumera las convenciones de llamada estándar para los registros R1, R13, R14 y R15.

Tabla 5. Uso estándar de los registros R1, R13, R14 y R15

Registrar	Uso
R1	LLAMAR DSNALI puntero de lista de parámetros
R13	Dirección del área de seguridad de la persona que llama
R14	Dirección del remitente de la llamada
R15	Dirección del punto de entrada CAF

Su programa de CAF debe respetar estas convenciones de registro.

CAF también admite lenguajes de alto nivel que no pueden examinar el contenido de registros individuales.

Conceptos relacionados

[Lista de parámetros de la sentencia CALL DSNALI](#)

La sentencia CALL DSNALI invoca explícitamente CAF. Cuando incluye sentencias CALL DSNALI en el programa, debe especificar todos los parámetros que van antes del parámetro de código de retorno.

Conexiones implícitas a CAF

Si la interfaz de lenguaje CAF (DSNALI) está disponible y no especifica explícitamente las sentencias CALL DSNALI en la aplicación, CAF inicia implícitamente las solicitudes CONNECT y OPEN para Db2. Estas solicitudes están sujetas a los mismos códigos de razón y códigos de retorno de Db2 como han especificado explícitamente las solicitudes.

Las conexiones implícitas utilizan los siguientes valores predeterminados:

Nombre de subsistema

El nombre predeterminado que se especifica en el módulo DSNHDECP. CAF utiliza el DSNHDECP predeterminado de la instalación, a menos que su propio módulo DSNHDECP esté en una biblioteca en una declaración STEPLIB de una concatenación JOBLIB o en la lista de enlaces. En un grupo de intercambio de datos, el nombre predeterminado del subsistema es el nombre del archivo adjunto del grupo.

Las conexiones implícitas a CAF siempre utilizan DSNHDECP como módulo predeterminado de la aplicación especificado por el usuario.

Asegúrese de que conoce el nombre predeterminado y que designa el subsistema específico de Db2 que desea utilizar.

Nombre de plan

El nombre de miembro del módulo de solicitud de base de datos (DBRM) que Db2 produjo cuando precompiló el programa fuente que contiene la primera llamada SQL.

Existen diferentes tipos de conexiones implícitas. Lo más sencillo es que una aplicación no llame a las funciones CONNECT ni OPEN. También puede utilizar solo la función CONECTAR o solo la función ABRIR. Cada una de estas llamadas conecta implícitamente su aplicación con Db2. Para finalizar una conexión implícita, debe utilizar las llamadas adecuadas.

Conceptos relacionados

[Resumen del comportamiento de CAF](#)

El efecto de cualquier función CAF depende en parte de las funciones que ya está ejecutando el programa. Debe planificar las llamadas a la función CAF que hace el programa para evitar errores y problemas estructurales importantes en la aplicación.

Listado de parámetros de la sentencia CALL DSNALI

La sentencia CALL DSNALI invoca explícitamente CAF. Cuando incluye sentencias CALL DSNALI en el programa, debe especificar todos los parámetros que van antes del parámetro de código de retorno.

Para las declaraciones CALL DSNALI, utilice una lista de parámetros CALL estándar z/OS Lista de parámetros CALL. Registrar 1 puntos en una lista de direcciones de palabras completas que apuntan a los parámetros reales. La última dirección debe contener un 1 en el bit de orden superior.

En las sentencias CALL DSNALI, no se puede omitir ninguno de los parámetros que van antes del parámetro de código de retorno codificando ceros o espacios en blanco. No existen valores predeterminados para esos parámetros para solicitudes de conexión explícitas. Los valores predeterminados se proporcionan solo para conexiones implícitas. Todos los parámetros que comienzan con el parámetro de código de retorno son opcionales.

Cuando desee utilizar el valor predeterminado para un parámetro pero especifique parámetros posteriores, codifique la instrucción CALL DSNALI de la siguiente manera:

- Para el lenguaje C, cuando codifica sentencias CALL DSNALI en C, debe especificar la dirección de cada parámetro requerido, utilizando "la dirección del" operador (&), y no el parámetro en sí. Por ejemplo, para pasar el parámetro *startecb* en CONNECT, especifique la dirección del entero de 4 bytes (&*secb*).

```
char functn[13] = "CONNECT      ";
char    ssid[5] = "DB2A";
int     tecb      = 0;
int     secb      = 0;
ptr    ribptr;
int    retcode;
int    reascode;
ptr    eibptr;

fnret = dsnali(&functn[0], &ssid[0], &tecb, &secb, &ribptr, &retcode, &reascode,
               NULL, &eibptr);
```

- Para otros idiomas excepto el lenguaje ensamblador, código cero para ese parámetro en la instrucción CALL DSNALI. Por ejemplo, supongamos que está codificando una llamada CONNECT en un programa COBOL y desea especificar todos los parámetros excepto el parámetro de código de retorno. Puede escribir una declaración similar a la siguiente:

```
CALL 'DSNALI' USING FUNCTN SSID TECB SECB RIBPTR
      BY CONTENT ZERO BY REFERENCE REASCODE SRDURA EIBPTR.
```

- Para el lenguaje ensamblador, codifique una coma para ese parámetro en la instrucción CALL DSNALI. Por ejemplo, para especificar todos los parámetros opcionales excepto el parámetro de código de retorno, escriba una instrucción similar a la siguiente:

```
CALL DSNALI,(FUNCTN,SSID,TERMECB,STARTECB,RIBPTR,,REASCODE,SRDURA,EIBPTR,
      GROUPOVERRIDE)
```

La siguiente figura muestra un ejemplo de estructura de lista de parámetros para la función CONNECT.

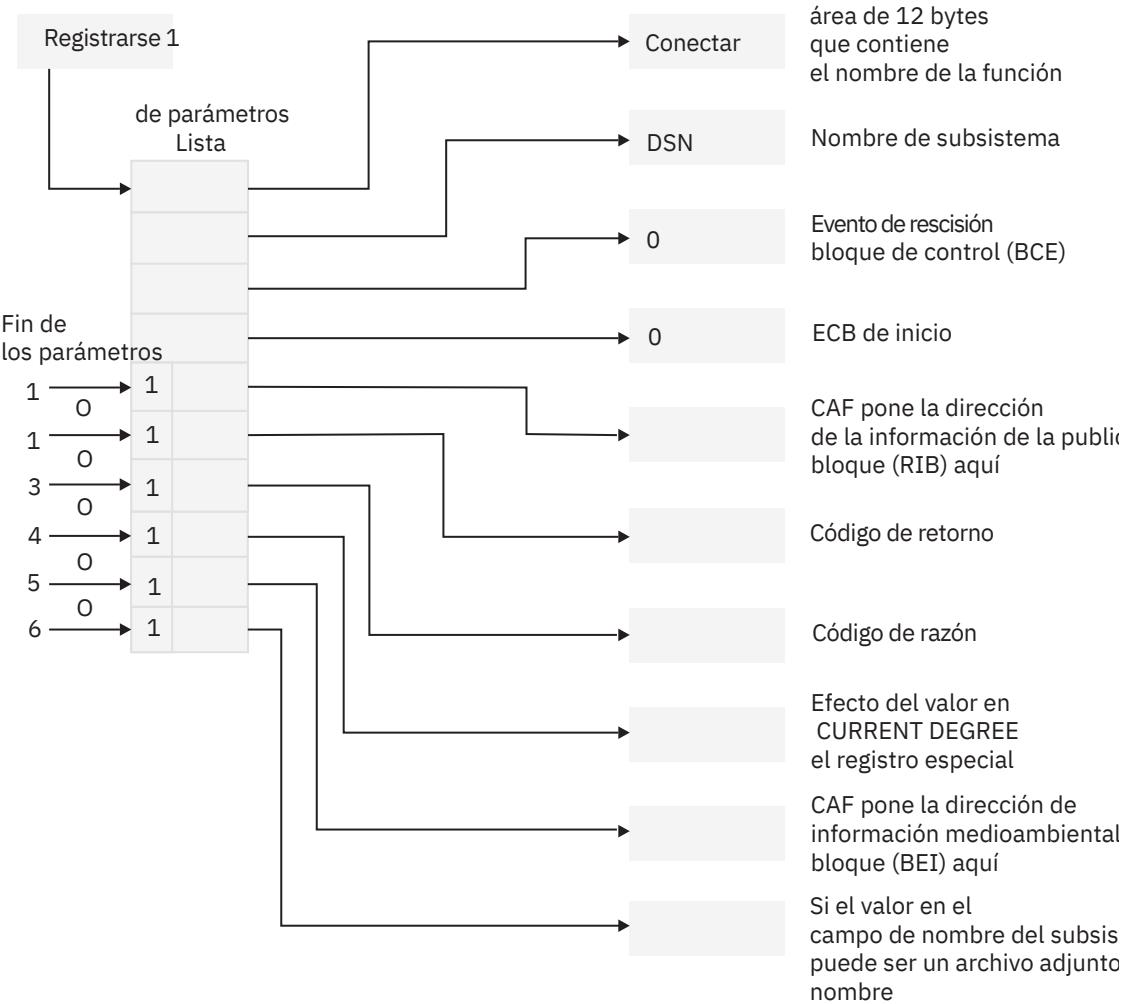


Figura 1. La lista de parámetros para una llamada CONNECT

La figura anterior ilustra cómo puede omitir parámetros para la sentencia CALL DSNALI para controlar los campos de código de retorno y código de motivo después de una llamada CONNECT. Puede finalizar la lista de parámetros en cualquiera de los siguientes puntos. Estos puntos de terminación se aplican a todas las listas de parámetros de estado CALL DSNALI.

1. Finaliza la lista de parámetros sin especificar los parámetros *retcode*, *reascode* y *srdura* y coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

Terminar la lista de parámetros en este punto garantiza la compatibilidad con los programas CAF que requieren un código de retorno en el registro 15 y un código de motivo en el registro 0.

2. Finaliza la lista de parámetros después del parámetro *retcode* y coloca el código de retorno en la lista de parámetros y el código de razón en el registro 0.

Terminar la lista de parámetros en este punto permite que el programa de aplicación actúe, basándose en el código de retorno, sin examinar más el código de razón asociado.

3. Finaliza la lista de parámetros después del parámetro *reascode* y coloca el código de retorno y el código de razón en la lista de parámetros.

Terminar la lista de parámetros en este punto proporciona soporte a lenguajes de alto nivel que no pueden examinar el contenido de registros individuales.

Si codifica su aplicación CAF en lenguaje ensamblador, puede especificar el parámetro de código de razón y omitir el parámetro de código de retorno.

4. Finaliza la lista de parámetros después del parámetro *srdura*.

Si codifica su aplicación CAF en lenguaje ensamblador, puede especificar este parámetro y omitir *los parámetros retcode y reascode*.

5. Finaliza la lista de parámetros después del parámetro *eibptr*.

Si codifica su aplicación CAF en lenguaje ensamblador, puede especificar este parámetro y omitir *los parámetros retcode, reascode o srdura*.

6. Finaliza la lista de parámetros después de *la anulación de grupo de parámetros*.

Si codifica su aplicación CAF en lenguaje ensamblador, puede especificar este parámetro y omitir *los parámetros retcode, reasc ode, srdura o eibptr*.

Incluso si especifica que el código de retorno se coloque en la lista de parámetros, también se coloca en el registro 15 para acomodar lenguajes de alto nivel que soportan procesamiento especial de códigos de retorno.

Conceptos relacionados

Cómo modifica CAF el contenido de los registros

Si no especifica los parámetros de código de retorno y de código de razón en las llamadas de función de CAF o si invoca a CAF implícitamente, CAF coloca un código de retorno en el registro 15 y un código de razón en el registro 0. El contenido de los registros 2 a 14 se conserva en todas las llamadas.

Resumen del comportamiento de CAF

El efecto de cualquier función CAF depende en parte de las funciones que ya está ejecutando el programa. Debe planificar las llamadas a la función CAF que hace el programa para evitar errores y problemas estructurales importantes en la aplicación.

La siguiente tabla resume el comportamiento del CAF después de varias entradas de programas de aplicación. La fila superior muestra las posibles funciones CAF que los programas pueden llamar. La primera columna muestra el historial más reciente de solicitudes de conexión de la tarea. Por ejemplo, el valor "CONNECT seguido de OPEN" en la primera columna significa que la tarea emitió CONNECT y luego OPEN sin otras llamadas CAF en el medio. La intersección de una fila y una columna muestra el efecto de la siguiente llamada si sigue el historial de conexión correspondiente. Por ejemplo, si la llamada es OPEN y el historial de conexión es CONNECT, el efecto es OPEN; se realiza la función OPEN. Si la llamada es SQL y el historial de conexiones está vacío (lo que significa que la llamada SQL es la primera función CAF del programa), el efecto es que se realizan las funciones implícitas CONNECT y OPEN, seguidas de la función SQL.

Tabla 6. Efectos de las llamadas a través de la función de marcación automática, según el historial de conexión

función Previous	función Next					
	CONNECT	OPEN	SQL	CLOSE	DISCONNECT	TRANSLATE
Vacio: primera llamada	CONNECT	OPEN	CONNECT, OPEN, seguido de la llamada SQL o IFI	Error 2031	Error 2041	Error 2051
CONNECT	Error 2011	OPEN	OPEN, seguido de la llamada SQL o IFI	Error 2031	DISCONNECT	TRANSLATE
CONNECT seguido de OPEN	Error 2011	Error 2021	La llamada SQL o IFI	CERR AR2	DISCONNECT	TRANSLATE
CONNECT seguido de llamada SQL o IFI	Error 2011	Error 2021	La llamada SQL o IFI	CERR AR2	DISCONNECT	TRANSLATE

Tabla 6. Efectos de las llamadas a través de la función de marcación automática, según el historial de conexión (continuación)

función Previous	función Next					
	CONNECT	OPEN	SQL	CLOSE	DISCONNECT	TRANSLATE
OPEN	Error 2011	Error 2021	La llamada SQL o IFI	CERR AR2	Error 2041	TRANSLATE
Llamada SQL o IFI	Error 2011	Error 2021	La llamada SQL o IFI	CERR AR2	Error 2041	TRADUCIR3

Notas:

1. Un error se muestra en esta tabla como *Error nnn*. El código de motivo correspondiente es X'00C10nnn'. El número del mensaje es *DSNAnnnI* o *DSNAnnnE*.
2. Las conexiones de tareas y espacio de direcciones permanecen activas. Si la llamada CLOSE falla porque Db2 no está disponible, los bloques de control CAF se restablecen, la función produce el código de retorno 4 y el código de motivo X'00C10824', y CAF está listo para más solicitudes de conexión cuando Db2 esté disponible.
3. Se acepta una solicitud de TRADUCCIÓN, pero en este caso es redundante. CAF emite automáticamente una solicitud de TRANSLATE cuando falla una solicitud SQL o IFI.

Referencia relacionada

Códigos de retorno y códigos de razón de CAF

CAF proporciona los códigos de retorno para los parámetros correspondientes que se especifican en una llamada a función CAF o, si decide no utilizar esos parámetros, a los registros 15 y 0.

Funciones de conexión CAF

Una función de conexión CAF especifica la acción que desea que realice CAF. Especifique estas funciones cuando invoque CAF a través de sentencias CALL DSNALI.

Puede especificar las siguientes funciones CAF en una sentencia CALL DSNALI:

CONNECT

Establece la tarea (TCB) como usuario del subsistema Db2 mencionado. Cuando la primera tarea dentro de un espacio de direcciones emite una solicitud de conexión, el espacio de direcciones también se inicializa como un usuario de Db2.

OPEN

Asigna un plan de e Db2 . Debe asignar un plan antes de que Db2 pueda procesar las sentencias SQL. Si no solicitó la función CONNECT, la función OPEN establece implícitamente la tarea y, opcionalmente, el espacio de direcciones, como usuario de Db2.

CLOSE

Realiza o finaliza de forma anormal cualquier cambio en la base de datos y desasigna el plan. Si la función OPEN solicita implícitamente la función CONNECT, la función CLOSE elimina la tarea y, posiblemente, el espacio de direcciones, como usuario de Db2.

DISCONNECT

Elimina la tarea como usuario de Db2 y, si esta tarea es la última o la única en el espacio de direcciones con una conexión Db2 , finaliza la conexión del espacio de direcciones a Db2.

TRANSLATE

Devuelve un código SQL y un texto imprimible que describen un código de motivo de error hexadecimal de tipo " Db2 ". Esta información se devuelve al SQLCA.

Restricción: No puede llamar a la función TRANSLATE desde el idioma e Fortran.

Recomendación: Dado que el efecto de cualquier función CAF depende de las funciones que el programa ya haya ejecutado, planifique cuidadosamente las llamadas que su programa realiza a estas funciones

de conexión CAF. Lea el resumen del comportamiento de CAF y haga estas llamadas de función en consecuencia.

Conceptos relacionados

Resumen del comportamiento de CAF

El efecto de cualquier función CAF depende en parte de las funciones que ya está ejecutando el programa. Debe planificar las llamadas a la función CAF que hace el programa para evitar errores y problemas estructurales importantes en la aplicación.

Lista de parámetros de la sentencia CALL DSNALI

La sentencia CALL DSNALI invoca explícitamente CAF. Cuando incluye sentencias CALL DSNALI en el programa, debe especificar todos los parámetros que van antes del parámetro de código de retorno.

Función CONNECT para CAF

La función CAF CONNECT inicia una conexión con Db2. Esta función es diferente de la instrucción SQL CONNECT que accede a una ubicación remota dentro de Db2.

La función CONNECT establece la tarea del llamante como usuario de los servicios de Db2 . Si ninguna otra tarea en el espacio de direcciones mantiene actualmente una conexión con el subsistema especificado, la función CONNECT también inicializa el espacio de direcciones para la comunicación con los espacios de direcciones Db2 . La función CONNECT establece la autorización de memoria cruzada del espacio de direcciones en Db2 y construye bloques de control del espacio de direcciones. Puede emitir una solicitud CONNECT desde cualquiera o todas las tareas en el espacio de direcciones, pero el nivel de espacio de direcciones se inicializa solo una vez cuando se conecta la primera tarea.

El uso de la función CONNECT es opcional. Si no llama a la función CONNECT, establezca una conexión implícita sin llamar a la función CONNECT ni a la función OPEN. En un programa que no contenga las funciones CONNECT u OPEN, cuando se ejecuta la primera sentencia SQL, la conexión implícita se realiza al subsistema predeterminado de Db2 . El nombre predeterminado del subsistema de Db2 es el nombre del subsistema especificado por el parámetro SSID=*xxxxx* en la tarea de instalación DSNTIJUA. Job DSNTIJUA ensambla el módulo de valores predeterminados de la aplicación de solo datos de Db2 .

Si no llama a la función CONNECT, la primera solicitud de una tarea, ya sea una solicitud OPEN o una llamada SQL o IFI, hace que CAF emita una solicitud CONNECT implícita. Si una tarea está conectada implícitamente, la conexión con Db2 finaliza cuando se llama a la función CLOSE o cuando finaliza la tarea.

Llame a la función CONNECT en todas las situaciones siguientes:

- Debe especificar un nombre de subsistema (*ssnm*) concreto que no sea el nombre de subsistema predeterminado.
- Necesitas que el valor del registro especial CURRENT DEGREE dure tanto como la conexión (*srdura*).
- Debe supervisar el ECB de inicio (*startecb*) de Db2 , el ECB de terminación (*termecb*) de Db2 o el nivel de liberación (release level) de Db2 .
- Planea tener varias tareas abiertas en el espacio de direcciones y cerrar planes o una sola tarea abierta en el espacio de direcciones y cerrar planes más de una vez.

Establecer conexiones a nivel de espacio de tareas y direcciones implica una sobrecarga significativa. El uso de la función CONNECT para establecer una conexión de tarea minimiza explícitamente esta sobrecarga al garantizar que la conexión con Db2 permanece después de que la función CLOSE desasigne un plan. En este caso, la conexión finaliza solo cuando se utiliza la función DESCONECTAR o cuando finaliza la tarea.

La función CONNECT también permite a la persona que llama conocer los siguientes elementos:

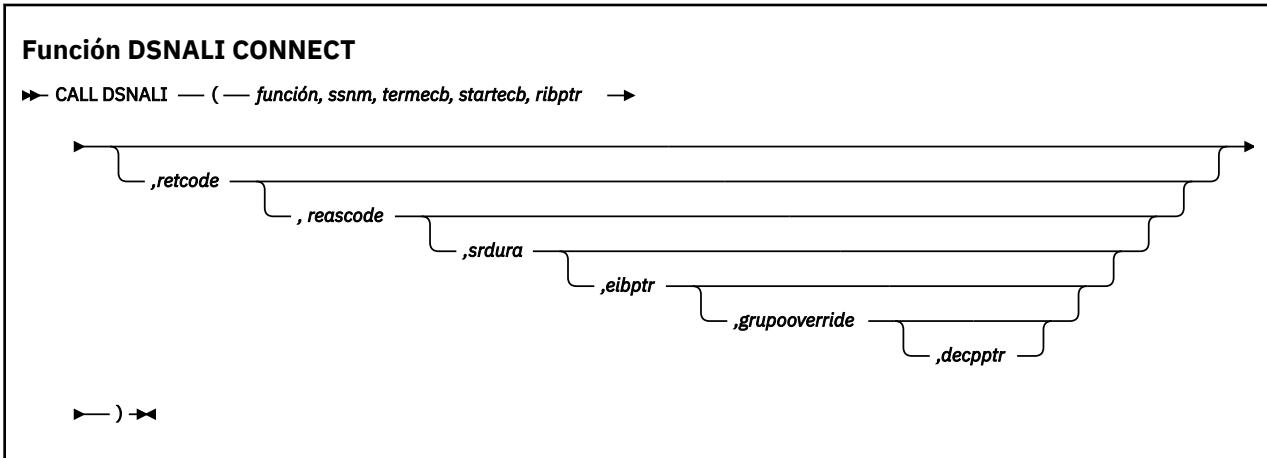
- Que el operador haya emitido una orden de PARADA DE DB2 . Cuando se produce este evento, Db2 publica el ECB de terminación, *termecb*. Tu solicitud puede esperar o simplemente mirar el ECB.
- Db2 , se está cerrando de forma anormal. Cuando se produce este evento, Db2 publica el ECB de terminación, *termecb*.

- Que Db2 vuelve a estar disponible después de un intento de conexión fallido debido a que Db2 no estaba disponible. Tu aplicación puede esperar o mirar el ECB de inicio, *startecb*. Db2 ignora este ECB si estaba activo en el momento de la solicitud de CONNECT.
- El nivel de lanzamiento actual de Db2. Para encontrar esta información, acceda al campo RIBREL en el bloque de información de liberación (RIB). Si RIBREL es '999', el nivel real de versión, lanzamiento y modificación de Db2 se indica en el campo RIBRELX y sus subcampos.

Restricción: No emita solicitudes CONNECT desde un TCB que ya tenga una conexión Db2 activa.

Recomendación: No mezcle solicitudes explícitas CONNECT y OPEN con conexiones establecidas implícitamente en el mismo espacio de direcciones. Especifique explícitamente qué subsistema de Db2 desea utilizar o permita que todas las solicitudes utilicen el subsistema predeterminado.

El siguiente diagrama muestra la sintaxis de la función CONNECT.



Los parámetros apuntan a las siguientes áreas:

función

Un área de 12 bytes que contiene CONNECT seguido de cinco espacios en blanco.

ssnm

Un nombre de subsistema de 4 bytes de Db2 , o un nombre de adjunto de grupo o de adjunto de subgrupo (si se utiliza en un grupo de intercambio de datos) al que se realiza la conexión.

Si el *ssnm* tiene menos de cuatro caracteres, rellénalo a la derecha con espacios en blanco hasta alcanzar una longitud de cuatro caracteres.

termecb

Un entero de 4 bytes que representa el bloque de control de eventos (ECB) de la aplicación para la terminación de un Db2 . Db2 publica este ECB cuando el operador introduce el comando STOP DB2 o cuando Db2 se está cerrando de forma anormal. El BCE indica el tipo de terminación mediante un código POST, como se muestra en la siguiente tabla:

Tabla 7. Códigos POST y tipos de terminación relacionados

Código POSTAL	Tipo de terminación
8	QUIESCE
12	FORCE
16	ABTERM

Antes de comprobar el *término* en su programa de solicitud CAF, compruebe primero el código de retorno y el código de motivo de la llamada CONNECT para asegurarse de que la llamada se completó correctamente.

startech

Un entero de 4 bytes que representa el ECB de inicio de la aplicación. Si Db2 aún no se ha iniciado cuando la aplicación emite la llamada, Db2 publica el ECB cuando completa con éxito su proceso de inicio. Db2 publica como máximo un ECB de inicio por espacio de direcciones. El ECB es el asociado a la llamada CONNECT más reciente desde ese espacio de direcciones. Su programa de aplicación debe examinar cualquier código de motivo CAF y Db2 distinto de cero antes de emitir un WAIT en este ECB.

Si *ssnm* es un nombre de adjunto de grupo o de subgrupo, el primer subsistema de Db2 que comienza en el sistema local z/OS sistema local y coincide con el nombre de conexión de grupo especificado, publica el ECB.

ribptr

Un área de 4 bytes en la que CAF coloca la dirección del bloque de información de liberación (RIB) después de la llamada. Puede determinar qué nivel de liberación de Db2 está ejecutando actualmente examinando el campo RIBREL. Si RIBREL es '999', el nivel de versión, lanzamiento y modificación real de Db2 se indica en el campo RIBRELX y sus subcampos. Puede determinar el nivel de modificación dentro del nivel de lanzamiento examinando los campos RIBCNUMB y RIBCINFO. Si el valor del campo RIBCNUMB es mayor que cero, compruebe el campo RIBCINFO para ver los niveles de modificación.

Si el RIB no está disponible (por ejemplo, si nombra un subsistema que no existe), Db2 establece el área de 4 bytes a ceros.

El área a la que apunta *el cursor* está por debajo de la línea de 16 MB.

Su programa no tiene que utilizar el bloque de información de liberación, pero no puede omitir el parámetro *ribptr*.

Macro DSNDRIB mapea el bloque de información de lanzamiento (RIB). Se puede encontrar en *el prefijo.SDSNMACS(DSNDRIB)*.

retcode

Un área de 4 bytes en la que CAF coloca el código de retorno.

Este campo es opcional. Si no especifica *el código de retorno*, CAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que CAF coloca un código de motivo.

Este campo es opcional. Si no especifica *el código de reasignación*, CAF coloca el código de motivo en el registro 0. Si especifica *reascode*, también debe especificar *retcode*.

srdura

Un área de 10 bytes que contiene la cadena 'SRDURA(CD)'. Este campo es opcional. Si especifica *srdura*, el valor en el registro especial CURRENT DEGREE permanece vigente desde el momento de la llamada CONNECT hasta el momento de la llamada DISCONNECT. Si no especifica *srdura*, el valor en el registro especial CURRENT DEGREE permanece vigente desde el momento de la llamada OPEN hasta el momento de la llamada CLOSE. Si especifica este parámetro en cualquier lenguaje excepto ensamblador, también debe especificar *retcode* y *reascode*. En lenguaje ensamblador, puede omitir estos parámetros especificando comas como marcadores de posición.

eibptr

Un área de 4 bytes en la que CAF coloca la dirección del bloque de información del entorno (EIB). El EIB contiene información que puede utilizar si se conecta a un subsistema de intercambio de datos (Db2) que forma parte de un grupo de intercambio de datos. Por ejemplo, puede determinar el nombre del grupo de intercambio de datos, el miembro al que se conecta y si hay nuevas funciones activadas en el subsistema. Si el subsistema de intercambio de datos (Db2, EIB) al que se conecta no forma parte de un grupo de intercambio de datos, los campos del EIB relacionados con el intercambio de datos estarán en blanco. Si el EIB no está disponible (por ejemplo, si nombra un subsistema que no existe), Db2 establece el área de 4 bytes a ceros.

El área a la que apunta *eibptr* está por encima de la línea de 16 MB.

Puede omitir este parámetro cuando realice una llamada CONNECT.

Si especifica este parámetro en cualquier lenguaje excepto ensamblador, también debe especificar *retcode*, *reascode* y *srdura*. En lenguaje ensamblador, puede omitir *retcode*, *reascode* y *srdura* especificando comas como marcadores de posición.

Macro DSNDEIB mapea el EIB. Se puede encontrar en *el prefijo.SDSNMACS(DSNDEIB)*.

anulación de grupo

Un área de 8 bytes que proporciona la aplicación. Este parámetro es opcional. Si no desea que se intente adjuntar un grupo, especifique 'NOGROUP'. Esta cadena indica que el nombre del subsistema especificado por *ssnm* debe utilizarse como nombre de subsistema e Db2 , incluso si *ssnm* coincide con un nombre de adjunto de grupo o de adjunto de subgrupo. Si no se proporciona *groupoverride*, *ssnm* se utiliza como nombre de archivo adjunto de grupo o de subgrupo si coincide con un nombre de archivo adjunto de grupo o de subgrupo.

Si especifica este parámetro en cualquier lenguaje excepto ensamblador, también debe especificar *retcode*, *reascode*, *srdura* y *eibptr*. En lenguaje ensamblador, puede omitir *retcode*, *reascode*, *srdura* y *eibptr* especificando comas como marcadores de posición.

Recomendación: Evite utilizar *el parámetro groupoverride* cuando sea posible, porque limita la capacidad de realizar el enrutamiento dinámico de la carga de trabajo en un Parallel Sysplex. Sin embargo, debe utilizar este parámetro en un entorno de intercambio de datos cuando desee conectarse a un miembro específico de un grupo de intercambio de datos, y el nombre del subsistema de ese miembro sea el mismo que el nombre de adjunto del grupo o del subgrupo.

decpptr

Un área de 4 bytes en la que CAF debe poner la dirección del bloque de control DSNHDECP o del módulo de valores predeterminados de la aplicación especificado por el usuario que fue cargado por el subsistema *ssnm* cuando se inició ese subsistema. Esta área de 4 bytes es un puntero de 31 bits. Si no se encuentra *ssnm*, el área de 4 bytes se establece en 0.

El área a la que los puntos decpptr pueden estar por encima de la línea de 16 MB.

Si especifica este parámetro en cualquier lenguaje excepto ensamblador, también debe especificar *los parámetros retcode*, *reascode*, *srdura*, *eibptr* y *groupoverride*. En lenguaje ensamblador, puede omitir *los parámetros retcode*, *reascode*, *srdura*, *eibptr* y *groupoverride* especificando comas como marcadores de posición.

Ejemplo de llamadas a la función CAF CONNECT

La siguiente tabla muestra una llamada CONNECT en cada idioma.

Tabla 8. Ejemplos de llamadas de función de CAF CONNECT

Idioma	Ejemplo de llamada
Assembler	<pre>CALL DSNALI, (FUNCTN,SSID,TERMECB,STARTECB,RIBPTR,RETCODE,REASCODE,SRDURA, EIBPTR, GRPOVER)</pre>
C1	<pre>fnret=dsnali(&functn[0],&ssid[0], &tecb, &secb,&ribptr,&retcode, &reascode, &srdura[0], &eibptr, &grpover[0]);</pre>
COBOL	<pre>CALL 'DSNALI' USING FUNCTN SSID TERMECB STARTECB RIBPTR RETCODE REASCODE SRDURA EIBPTR GRPOVER.</pre>
Fortran	<pre>CALL DSNALI(FUNCTN,SSID,TERMECB,STARTECB,RIBPTR,RETCODE,REASCODE,SRDURA, EIBPTR,GRPOVER)</pre>

Tabla 8. Ejemplos de llamadas de función de CAF CONNECT (continuación)

Idioma	Ejemplo de llamada
PL/I	<pre>CALL DSNALI(FUNCTN,SSID,TERMECB,STARTECB,RIBPTR,RETCODE,REASCODE,SRDURA, EIBPTR,GRPOVER)</pre>

Nota:

- Para las aplicaciones C y PL/I, debe incluir las directivas de compilador adecuadas, ya que DSNALI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar CAF.

Conceptos relacionados

Ejemplos de invocación de CAF

El recurso de conexión de llamadas (CAF) permite a los programas comunicarse con Db2. Si invoca explícitamente CAF en el programa, puede utilizar las funciones de conexión CAF para controlar el estado de la conexión.

Tareas relacionadas

Invocación del recurso de conexión de llamada

Invoque el recurso de conexión de llamada (CAF) cuando desee que su programa de aplicación establezca y controle su propia conexión con Db2. Las aplicaciones que utilizan CAF pueden controlar explícitamente el estado de sus conexiones en Db2 utilizando funciones de conexión que CAF proporciona.

Referencia relacionada

Sincronización de tareas (Macros de WAIT, POST e EVENTS)(MVS Programming: Assembler Services Guide)

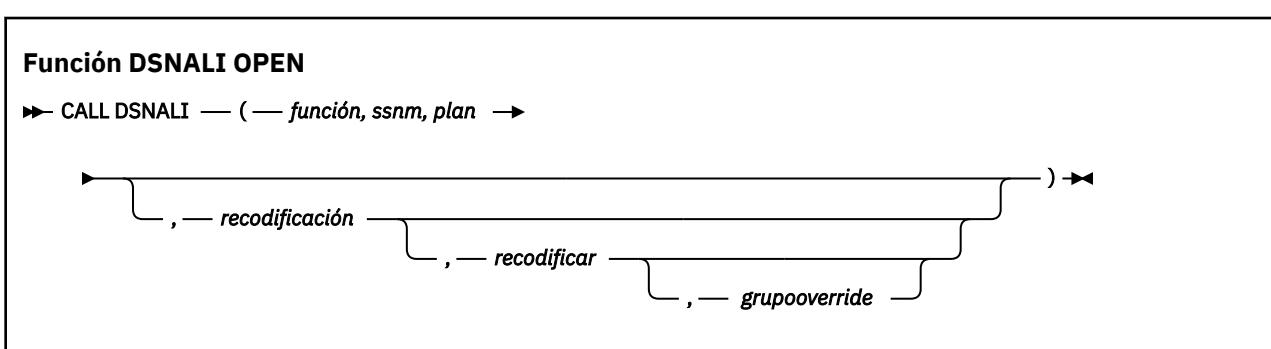
Función OPEN para CAF

La función OPEN asigna los recursos de Internet (Db2) necesarios para ejecutar el plan especificado o para emitir solicitudes IFI. Si la tarea solicitada aún no tiene conexión con el subsistema Db2, la función OPEN la establece.

El uso de la función ABRIR es opcional. Si no llama a la función OPEN, las acciones que realiza la función OPEN se producen implícitamente en la primera llamada SQL o IFI de la tarea.

Restricción: No utilice la función ABRIR si la tarea ya tiene un plan asignado.

El siguiente diagrama muestra la sintaxis de la función OPEN.



Los parámetros apuntan a las siguientes áreas:

función

Un área de 12 bytes que contiene la palabra OPEN seguida de ocho espacios en blanco.

ssnm

Un nombre de subsistema de 4 bytes de Db2, o un nombre de adjunto de grupo o de adjunto de subgrupo (si se utiliza en un grupo de intercambio de datos). La función OPEN (abrir) asigna el plan

especificado a este subsistema de e Db2 . Además, si la tarea solicitada no tiene ya una conexión con el subsistema Db2 mencionado, la función OPEN la establece.

Debe especificar el parámetro *ssnm*, incluso si la tarea solicitante también emite una llamada CONNECT. Si una tarea emite una llamada CONNECT seguida de una llamada OPEN, los nombres de subsistema para ambas llamadas deben ser los mismos.

Si *el ssnm* tiene menos de cuatro caracteres, rellénelo a la derecha con espacios en blanco hasta alcanzar una longitud de cuatro caracteres.

plan

Un nombre de plan de 8 bytes (Db2).

retcode

Un área de 4 bytes en la que CAF coloca el código de retorno.

Este campo es opcional. Si no especifica *el código de retorno*, CAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que CAF coloca un código de motivo.

Este campo es opcional. Si no especifica *el código de razón*, CAF lo coloca en el registro 0. Si especifica *reascode*, también debe especificar *retcode*.

anulación de grupo

Un área de 8 bytes que proporciona la aplicación. Este campo es opcional. Si no desea que se intente adjuntar un grupo, especifique 'NOGROUP'. Esta cadena indica que el nombre del subsistema especificado por *ssnm* debe utilizarse como nombre de subsistema e Db2 , incluso si *ssnm* coincide con un nombre de adjunto de grupo o de adjunto de subgrupo. Si no especifica groupoverride, *ssnm* se utiliza como nombre de archivo adjunto de grupo y de subgrupo si coincide con un nombre de archivo adjunto de grupo o de subgrupo. Si especifica este parámetro en cualquier lenguaje excepto ensamblador, también debe especificar *retcode* y *reascode*. En lenguaje ensamblador, puede omitir estos parámetros especificando comas como marcadores de posición.

Recomendación: Evite utilizar *el parámetro groupoverride* cuando sea posible, ya que limita la capacidad de realizar un enrutamiento dinámico de la carga de trabajo en un Parallel Sysplex. Sin embargo, debe utilizar este parámetro en un entorno de intercambio de datos cuando desee conectarse a un miembro específico de un grupo de intercambio de datos, y el nombre del subsistema de ese miembro sea el mismo que el nombre de la conexión del grupo o de la conexión del subgrupo.

Ejemplos de llamadas CAF OPEN

La siguiente tabla muestra una llamada ABIERTA en cada idioma.

Tabla 9. Ejemplos de llamadas CAF OPEN

Idioma	Ejemplo de llamada
Assembler	CALL DSNALI,(FUNCTN,SSID,PLANNAME, RETCODE,REASCODE,GRPOVER)
C1	fnret=dsnali(&functn[0],&ssid[0], &planname[0],&retcode, &reascode,&grpover[0]);
COBOL	CALL 'DSNALI' USING FUNCTN SSID PLANNAME RETCODE REASCODE GRPOVER.
Fortran	CALL DSNALI(FUNCTN,SSID,PLANNAME, RETCODE,REASCODE,GRPOVER)
PL/I1	CALL DSNALI(FUNCTN,SSID,PLANNAME, RETCODE,REASCODE,GRPOVER);

Nota:

- Para las aplicaciones C y PL/I, debe incluir las directivas de compilador adecuadas, ya que DSNALI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar CAF.

Conceptos relacionados

Conexiones implícitas a CAF

Si la interfaz de lenguaje CAF (DSNALI) está disponible y no especifica explícitamente las sentencias CALL DSNALI en la aplicación, CAF inicia implícitamente las solicitudes CONNECT y OPEN para Db2. Estas solicitudes están sujetas a los mismos códigos de razón y códigos de retorno de Db2 como han especificado explícitamente las solicitudes.

Tareas relacionadas

Invocación del recurso de conexión de llamada

Invoque el recurso de conexión de llamada (CAF) cuando desee que su programa de aplicación establezca y controle su propia conexión con Db2. Las aplicaciones que utilizan CAF pueden controlar explícitamente el estado de sus conexiones en Db2 utilizando funciones de conexión que CAF proporciona.

Función CERRAR para CAF

La función CAF CLOSE desasigna el plan que se creó explícitamente mediante una llamada a la función OPEN o implícitamente en la primera llamada SQL. Opcionalmente, la función CLOSE también desconecta la tarea, y posiblemente el espacio de direcciones, de Db2.

Si no emitió una llamada CONNECT explícita para la tarea, la función CLOSE elimina la conexión de la tarea con Db2. Si ninguna otra tarea en el espacio de direcciones tiene una conexión activa con Db2, Db2 también elimina las estructuras de bloques de control que se crearon para el espacio de direcciones y elimina la autorización de memoria cruzada.

El uso de la función CERRAR es opcional. Tenga en cuenta las siguientes reglas y recomendaciones sobre cuándo utilizar y no utilizar la función CERRAR:

- No utilice la función CERRAR cuando su tarea actual no tenga un plan asignado.
- Si desea utilizar un nuevo plan, debe emitir una llamada explícita de CIERRE, seguida de una llamada de ABRIR con el nombre del nuevo plan.
- Al cerrar la aplicación, puede mejorar el rendimiento de este cierre llamando explícitamente a la función CLOSE antes de que finalice la tarea. Si omite la llamada de CIERRE, Db2 realiza un CIERRE implícito. En este caso, Db2 realiza las mismas acciones cuando finaliza su tarea, utilizando el parámetro SYNC si la finalización es normal y el parámetro ABRT si la finalización es anormal.
- Si se termina un Db2 , emita una llamada explícita de CLOSE para cualquier tarea que no haya emitido una llamada de CONNECT. Esta acción permite a CAF restablecer sus bloques de control para permitir futuras conexiones. Esta llamada CLOSE devuelve el código de retorno de reinicio realizado (+004) y el código de motivo X'00C10824'. Si omite la llamada CLOSE en este caso, cuando Db2 vuelva a estar en línea, la siguiente solicitud de conexión de la tarea fallará. Obtendrá el mensaje YOUR TCB DOES NOT HAVE A CONNECTION, con X'00F30018' en el registro 0, o el mensaje de error CAF DSNA201I o DSNA202I, dependiendo de lo que su aplicación intentó hacer. La tarea debe entonces emitir una llamada de CIERRE antes de poder volver a conectarse a Db2.
- Una tarea que emitió una llamada CONNECT explícita debería emitir una llamada DISCONNECT en lugar de una llamada CLOSE. Esta acción hace que el CAF restablezca sus bloques de control cuando se termina un Db2 .

El siguiente diagrama muestra la sintaxis de la función CLOSE.

Función DSNALI CLOSE

```
► CALL DSNALI — ( — función, termop — , — recodificación — , — recodificar — ) ►
```

Los parámetros apuntan a las siguientes áreas:

función

Un área de 12 bytes que contiene la palabra CLOSE seguida de siete espacios en blanco.

termop

Una opción de terminación de 4 bytes, con uno de los siguientes valores:

SYNC

Especifica que Db2 debe comprometer cualquier dato modificado.

ABRT

Especifica que Db2 debe revertir los datos al punto de confirmación anterior.

retcode

Un área de 4 bytes en la que CAF debe colocar el código de retorno.

Este campo es opcional. Si no especifica *el código de retorno*, CAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que CAF coloca un código de motivo.

Este campo es opcional. Si no especifica *el código de razón*, CAF coloca el código de razón en el registro 0. Si especifica *reascode*, también debe especificar *retcode*.

Ejemplos de llamadas de CAF CLOSE

La siguiente tabla muestra una llamada CERRADA en cada idioma.

Tabla 10. Ejemplos de llamadas de CAF CLOSE

Idioma	Ejemplo de llamada
Assembler	CALL DSNALI,(FUNCTN,TERMOP,RETCODE, REASCODE)
C1	fnret=dsnali(&functn[0], &termop[0], &retcode,&reascode);
COBOL	CALL 'DSNALI' USING FUNCTN TERMOP RETCODE REASCODE.
Fortran	CALL DSNALI(FUNCTN,TERMOP, RETCODE,REASCODE)
PL/I1	CALL DSNALI(FUNCTN,TERMOP, RETCODE,REASCODE);

Nota:

- Para las aplicaciones C y PL/I, debe incluir las directivas de compilador adecuadas, ya que DSNALI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar CAF.

Tareas relacionadas

[Invocación del recurso de conexión de llamada](#)

Invoque el recurso de conexión de llamada (CAF) cuando desee que su programa de aplicación establezca y controle su propia conexión con Db2. Las aplicaciones que utilizan CAF pueden controlar explícitamente el estado de sus conexiones en Db2 utilizando funciones de conexión que CAF proporciona.

Función DESCONECTAR para CAF

La función DESCONECTAR CAF finaliza una conexión con Db2.

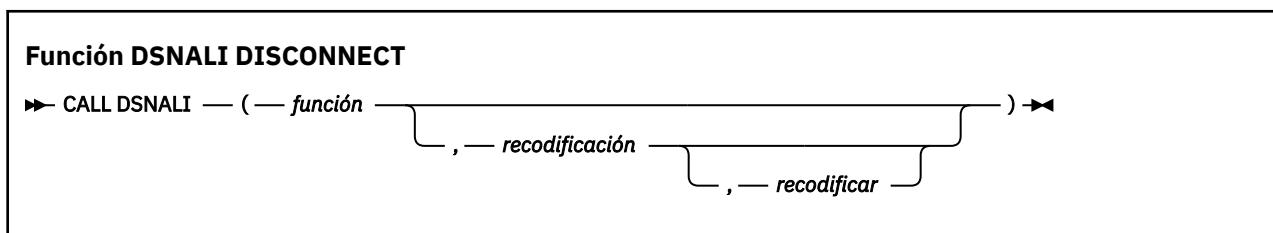
DESCONECTAR elimina la conexión de la tarea de llamada a Db2. Si ninguna otra tarea en el espacio de direcciones tiene una conexión activa con Db2, Db2 también elimina las estructuras de bloques de control que se crearon para el espacio de direcciones y elimina la autorización de memoria cruzada.

Si hay una llamada OPEN en curso, lo que significa que se ha asignado un plan, cuando se emite la llamada DISCONNECT, CAF emite un CLOSE implícito con el parámetro SYNC.

El uso de la función DESCONectar es opcional. Tenga en cuenta las siguientes reglas y recomendaciones sobre cuándo utilizar y cuándo no utilizar la función DESCONectar:

- Solo aquellas tareas que emitieron explícitamente una llamada CONNECT pueden emitir una llamada DISCONNECT. Si no se utilizó una llamada CONNECT, una llamada DISCONNECT causa un error.
 - Al cerrar la aplicación, puede mejorar el rendimiento de este cierre llamando explícitamente a la función DISCONNECT antes de que finalice la tarea. Si omite la llamada de DESCONEXIÓN, Db2 realiza una DESCONEXIÓN implícita. En este caso, Db2 realiza las mismas acciones cuando finaliza su tarea.
 - Si se produce una desconexión (Db2), cualquier tarea que haya emitido una llamada CONNECT debe emitir una llamada DISCONNECT para restablecer los bloques de control CAF. La función DESCONECTAR devuelve los códigos de retorno de reinicio completado y los códigos de motivo (+004 y X'00C10824'). Esta acción garantiza que las futuras solicitudes de conexión de la tarea funcionen cuando Db2 vuelva a estar en línea.
 - Una tarea que no emitió explícitamente una llamada CONNECT debe emitir una llamada CLOSE en lugar de una llamada DISCONNECT. Esta acción restablece los bloques de control CAF cuando se termina Db2.

El siguiente diagrama muestra la sintaxis de la función DISCONNECT.



El único parámetro apunta a la siguiente área:

función

Un área de 12 bytes que contiene la palabra DISCONNECT seguida de dos espacios en blanco.

retcode

Un área de 4 bytes en la que CAF coloca el código de retorno.

Este campo es opcional. Si no especifica el *código de retorno*, CAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que CAF coloca un código de motivo.

Este campo es opcional. Si no especifica *el código de reasignación*, CAF coloca el código de motivo en el registro 0. Si especifica *reascode*, también debe especificar *retcode*.

Ejemplos de llamadas de DESCONEXIÓN CAF

La siguiente tabla muestra una llamada de DESCONEXIÓN en cada idioma.

Tabla 11. Ejemplos de llamadas de DESCONEXIÓN CAF

Idioma	Ejemplo de llamada
Assembler	CALL DSNALI(,FUNCTN,RETCODE,REASCODE)
C1	fnret=dsnali(&functn[0], &retcode, &reascode);
COBOL	CALL 'DSNALI' USING FUNCTN RETCODE REASCODE.
Fortran	CALL DSNALI(FUNCTN,RETCODE,REASCODE)
PL/I1	CALL DSNALI(FUNCTN,RETCODE,REASCODE) ;

Nota:

- Para las aplicaciones C y PL/I, debe incluir las directivas de compilador adecuadas, ya que DSNALI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar CAF.

Tareas relacionadas

Invocación del recurso de conexión de llamada

Invoca el recurso de conexión de llamada (CAF) cuando deseé que su programa de aplicación establezca y controle su propia conexión con Db2. Las aplicaciones que utilizan CAF pueden controlar explícitamente el estado de sus conexiones en Db2 utilizando funciones de conexión que CAF proporciona.

Función de TRADUCCIÓN para CAF

La función TRANSLATE convierte un código de error hexadecimal de razón de error de conexión (Db2) de una solicitud de apertura fallida en un código de error SQL y un mensaje de error imprimible. Db2 coloca la información en las variables de host SQLCODE y SQLSTATE o en los campos relacionados del SQLCA del llamador.

El código de motivo de error de conversión de Db2 , se lee desde el registro 0. La función TRANSLATE no cambia el contenido de los registros 0 y 15, a menos que falle la solicitud de TRANSLATE; en ese caso, el registro 0 se establece en X' C10205 ' y el registro 15 se establece en 200.

Tenga en cuenta las siguientes reglas y recomendaciones sobre cuándo utilizar y no utilizar la función TRADUCIR:

- No puede llamar a la función TRANSLATE desde el idioma e Fortran.
- La función TRANSLATE solo es útil si se ha utilizado una llamada CONNECT explícita antes de una solicitud OPEN que falla. Para los errores que se producen durante las solicitudes SQL o IFI, la función TRANSLATE se ejecuta automáticamente.
- La función TRANSLATE puede traducir aquellos códigos que comienzan con X'00F3', pero no traduce los códigos de razón CAF que comienzan con X'00C1'.

Si recibe el código de motivo de error X'00F30040' (*recurso no disponible*) después de una solicitud OPEN, la función TRANSLATE devuelve el nombre del objeto de base de datos no disponible en los últimos 44 caracteres del campo SQLERRM.

Si la función TRANSLATE no reconoce el código de motivo de error, devuelve SQLCODE -924 (SQLSTATE '58006') y coloca una copia imprimible del código de función original de la función Db2 y los códigos de motivo de error y de retorno en el campo SQLERRM.

El siguiente diagrama muestra la sintaxis de la función TRANSLATE.

Función DSNALI TRANSLATE

```
► CALL DSNALI — ( — función, sqlca — , — recodificación — , — recodificar ) ►
```

Los parámetros apuntan a las siguientes áreas:

función

Un área de 12 bytes que contiene la palabra TRANSLATE seguida de tres espacios en blanco.

sqlca

El área de comunicación SQL (SQLCA) del programa.

retcode

Un área de 4 bytes en la que CAF coloca el código de retorno.

Este campo es opcional. Si no especifica *el código de retorno*, CAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que CAF coloca un código de motivo.

Este campo es opcional. Si no especifica *el código de razón*, CAF coloca el código de razón en el registro 0. Si especifica *reascode*, también debe especificar *retcode*.

Ejemplos de llamadas a CAF TRANSLATE

La siguiente tabla muestra una llamada de TRANSLATE en cada idioma.

Tabla 12. Ejemplos de llamadas a CAF TRANSLATE

Idioma	Ejemplo de llamada
Assembler	CALL DSNALI,(FUNCTN,SQLCA,RETCODE, REASCODE)
C1	fnret=dsnali(&functn[0], &sqlca, &retcode, &reascode);
COBOL	CALL 'DSNALI' USING FUNCTN SQLCA RETCODE REASCODE.
PL/I1	CALL DSNALI(FUNCTN,SQLCA,RETCODE, REASCODE);

Nota:

- Para las aplicaciones C y PL/I, debe incluir las directivas de compilador adecuadas, ya que DSNALI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar CAF.

Tareas relacionadas

Invocación del recurso de conexión de llamada

Invoque el recurso de conexión de llamada (CAF) cuando desee que su programa de aplicación establezca y controle su propia conexión con Db2. Las aplicaciones que utilizan CAF pueden controlar explícitamente el estado de sus conexiones en Db2 utilizando funciones de conexión que CAF proporciona.

Activar un seguimiento de CAF

CAF no captura ningún mensaje de seguimiento de diagnóstico a menos que usted se lo indique activando un seguimiento.

Procedimiento

Asigne un conjunto de datos DSNTRACE de forma dinámica o incluyendo una sentencia DD DSNTRACE en su JCL.

CAF escribe mensajes de seguimiento de diagnóstico en ese conjunto de datos. Los números de los mensajes de seguimiento contienen los tres últimos dígitos de los códigos de motivo.

Conceptos relacionados

Ejemplos de invocación de CAF

El recurso de conexión de llamadas (CAF) permite a los programas comunicarse con Db2. Si invoca explícitamente CAF en el programa, puede utilizar las funciones de conexión CAF para controlar el estado de la conexión.

Códigos de retorno y códigos de razón de CAF

CAF proporciona los códigos de retorno para los parámetros correspondientes que se especifican en una llamada a función CAF o, si decide no utilizar esos parámetros, a los registros 15 y 0.

Cuando el código de motivo comienza por X'00F3' excepto para X'00F30006', puede utilizar la función CAF TRANSLATE para obtener el texto del mensaje de error que se puede imprimir y mostrar. Estos códigos de motivo son emitidos por el soporte del subsistema para memorias aliadas, una parte del subcomponente de soporte del subsistema de Db2 , que atiende todas las solicitudes de conexión y trabajo de Db2 .

Para las llamadas SQL, CAF devuelve códigos SQL estándar en el SQLCA. CAF devuelve los códigos de devolución IFI y los códigos de motivo en el área de comunicación de la instalación de instrumentación (IFCA).

La siguiente tabla enumera los códigos de devolución y los códigos de motivo de la CAF.

Tabla 13. Códigos de retorno y códigos de razón de CAF

Código de retorno	Código de razón	Explicación
0	X'00000000'	Realización satisfactoria.
4	X'00C10824'	Restablecimiento de CAF completado. CAF está lista para establecer una nueva conexión.
8	X'00C10831'	Desajuste de nivel de liberación entre Db2 y el código CAF.
200 1	X'00C10201'	Recibida una segunda solicitud CONNECT del mismo TCB. La primera solicitud de CONNECT podría haber sido implícita o explícita.
200 1	X'00C10202'	Se ha recibido una segunda solicitud de ABRIR del mismo TCB. La primera solicitud de ABRIR podría haber sido implícita o explícita.
200 1	X'00C10203'	Solicitud de CERRAR emitida cuando no existe ninguna solicitud de ABRIR activa.
200 1	X'00C10204'	Solicitud de DESCONEXIÓN emitida cuando no existe una solicitud de CONEXIÓN activa, o la macro AXSET se emitió entre la solicitud de CONEXIÓN y la solicitud de DESCONEXIÓN.
200 1	X'00C10205'	SOLICITUD DE TRADUCCIÓN emitida cuando no existe conexión con Db2 .
200 1	X'00C10206'	Se especificó un número incorrecto de parámetros o el bit de fin de lista estaba desactivado.
200 1	X'00C10207'	Parámetro de función no reconocido.

Tabla 13. Códigos de retorno y códigos de razón de CAF (continuación)

Código de retorno	Código de razón	Explicación
200 1	X'00C10208'	Se han recibido solicitudes de acceso a dos subsistemas de Db2 diferentes desde el mismo TCB.
204	2	Error del sistema CAF. Probable error en el archivo adjunto o en el Db2.

Notas:

1. Un error CAF probablemente causado por errores en las listas de parámetros de los programas de aplicación. Los errores de CAF no cambian el estado actual de su conexión a Db2; puede continuar el proceso con una solicitud corregida.
2. Los errores del sistema causan fallos. Si el rastreo está activado, se escribe un mensaje descriptivo en el conjunto de datos DSNTTRACE justo antes de la noche.

Ejemplos de escenarios de CAF

Una o más tareas pueden utilizar la función de adjuntar llamadas (CAF) para conectarse a Db2. Esta conexión puede establecerse de forma implícita o explícita. Para conexiones explícitas, una tarea llama a una o más de las funciones de conexión CAF.

Una sola tarea con conexiones implícitas

El escenario de conexión más simple es una sola tarea que realiza llamadas a Db2 sin utilizar instrucciones CALL DSNALI explícitas. La tarea se conecta implícitamente al nombre del subsistema predeterminado y utiliza el nombre del plan predeterminado.

Cuando la tarea finaliza, se producen los siguientes eventos:

- Si la terminación fue normal, se confirman los cambios en la base de datos.
- Si la terminación fue anormal, se revertirán los cambios en la base de datos.
- El plan activo y todos los recursos de la base de datos se desasignan.
- Se terminan las conexiones de tareas y espacio de direcciones a Db2 .

Una sola tarea con conexiones explícitas

El siguiente pseudocódigo de ejemplo ilustra un escenario más complejo con una sola tarea.

```
CONNECT
  OPEN      allocate a plan
  SQL or IFI call
  ...
  CLOSE     deallocate the current plan
  OPEN      allocate a new plan
  SQL or IFI call
  ...
  CLOSE
DISCONNECT
```

Una tarea puede tener conexión con un solo subsistema de Db2 en cualquier momento. Se produce un error CAF si el nombre del subsistema en la llamada OPEN no coincide con el nombre del subsistema en la llamada CONNECT. Para cambiar a un subsistema diferente, la aplicación debe desconectarse primero del subsistema actual y, a continuación, emitir una solicitud de conexión con un nuevo nombre de subsistema.

Múltiples tareas

En el siguiente escenario, varias tareas dentro del espacio de direcciones utilizan servicios de Db2 . Cada tarea debe especificar explícitamente el mismo nombre de subsistema en la solicitud de la función

CONNECT o en la solicitud de la función OPEN. La tarea 1 no realiza llamadas SQL o IFI. Su propósito es supervisar los ECB de terminación y arranque de e Db2 , y comprobar el nivel de liberación de e Db2 .

TASK 1	TASK 2	TASK 3	TASK n
CONNECT			
	OPEN SQL	OPEN SQL	OPEN SQL

	CLOSE OPEN SQL	CLOSE OPEN SQL	CLOSE OPEN SQL
	CLOSE	CLOSE	...
DISCONNECT			

Ejemplos de invocación de CAF

El recurso de conexión de llamadas (CAF) permite a los programas comunicarse con Db2. Si invoca explícitamente CAF en el programa, puede utilizar las funciones de conexión CAF para controlar el estado de la conexión.

Ejemplo de JCL para invocar CAF

El siguiente ejemplo de JCL muestra cómo utilizar CAF en un entorno por lotes (no TSO). La declaración DSNTRACE en este ejemplo es opcional.

```
//jobname      JOB      z/OS_jobcard_information
//CAFJCL       EXEC    PGM=CAF_application_program
//STEPLIB       DD      DSN=application_load_library
//                  DD      DSN=DB2_load_library
:
//SYSPRINT     DD      SYSOUT=*
//DSNTRACE      DD      SYSOUT=*
//SYSUDUMP      DD      SYSOUT=*
```

Ejemplo de código ensamblador que invoca CAF

Los siguientes ejemplos muestran partes de un programa de ensamblaje de muestra que utiliza CAF. Demuestran las técnicas básicas para realizar llamadas CAF, pero no muestran el código y z/OS macros necesarios para admitir esas llamadas. Por ejemplo, muchas aplicaciones necesitan una estructura de dos tareas para que las rutinas de manejo de la atención puedan separar las subtareas conectadas para recuperar el control de Db2. Esta estructura no se muestra en los siguientes ejemplos de código. Además, estos ejemplos de código asumen la existencia de una macro WRITE. Dondequiera que se incluya esta macro en el ejemplo, sustituya el código por el suyo propio. Debe decidir qué quiere que haga su aplicación en esas situaciones; probablemente no quiera escribir los mensajes de error mostrados.

Ejemplo de carga y eliminación de la interfaz de idioma CAF

El siguiente segmento de código muestra cómo una aplicación puede cargar los puntos de entrada DSNALI y DSNHLI2 para la interfaz de lenguaje CAF. Almacenar los puntos de entrada en las variables LIALI y LISQL garantiza que la aplicación tenga que cargar los puntos de entrada solo una vez. Cuando el módulo haya terminado con Db2, debe eliminar las entradas.

```
***** GET LANGUAGE INTERFACE ENTRY ADDRESSES
LOAD EP=DSNALI          Load the CAF service request EP
ST   R0,LIALI            Save this for CAF service requests
LOAD EP=DSNHLI2          Load the CAF SQL call Entry Point
ST   R0,LISQL             Save this for SQL calls
*   :   Insert connection service requests and SQL calls here
*   DELETE EP=DSNALI        Correctly maintain use count
*   DELETE EP=DSNHLI2       Correctly maintain use count
```

Ejemplo de conexión a Db2 con CAF

El siguiente código de ejemplo muestra cómo emitir solicitudes explícitas para determinadas acciones, como CONNECT, OPEN, CLOSE, DISCONNECT y TRANSLATE, y utiliza la subrutina CHEKCODE para comprobar los códigos de motivo de devolución de CAF.

```
***** CONNECT *****
L    R15,LIALI      Get the Language Interface address
MVC FUNCNTN,CONNECT   Get the function to call
CALL (15),(FUNCNTN,SSID,TECB,SECB,RIBPTR),VL,MF=(E,CAFCALL)
BAL R14,CHEKCODE     Check the return and reason codes
CLC CONTROL,CONTINUE Is everything still OK
BNE EXIT             If CONTROL not 'CONTINUE', stop loop
USING R8,RIB          Prepare to access the RIB
L    R8,RIBPTR        Access RIB to get DB2 release level
CLC RIBREL,RIBR999    DB2 V10 or later?
BE  USERELX          If RIBREL = '999', use RIBRELX
WRITE 'The current DB2 release level is' RIBREL
B   OPEN              Continue with signon
USERELX WRITE 'The current DB2 release level is' RIBRELX

***** OPEN *****
OPEN   L    R15,LIALI      Get the Language Interface address
MVC FUNCNTN,OPEN       Get the function to call
CALL (15),(FUNCNTN,SSID,PLAN),VL,MF=(E,CAFCALL)
BAL R14,CHEKCODE     Check the return and reason codes

***** SQL *****
*           Insert your SQL calls here. The DB2 Precompiler
*           generates calls to entry point DSNHLI. You should
*           specify the precompiler option ATTACH(CAF), or code
*           a dummy entry point named DSNHLI to intercept
*           all SQL calls. A dummy DSNHLI is shown below.
***** CLOSE *****
CLC CONTROL,CONTINUE Is everything still OK?
BNE EXIT             If CONTROL not 'CONTINUE', shut down
MVC TRMOP,ABRT       Assume termination with ABRT parameter
L    R4,SQLCODE        Put the SQLCODE into a register
C    R4,CODE0           Examine the SQLCODE
BZ   SYNCTERM         If zero, then CLOSE with SYNC parameter
C    R4,CODE100         See if SQLCODE was 100
BNE  DISC             If not 100, CLOSE with ABRT parameter
SYNCTERM MVC TRMOP,SYNC  Good code, terminate with SYNC parameter
DISC   DS   OH          Now build the CAF parmlist
L    R15,LIALI      Get the Language Interface address
MVC FUNCNTN,CLOSE    Get the function to call
CALL (15),(FUNCNTN,TRMOP),VL,MF=(E,CAFCALL)
BAL R14,CHEKCODE     Check the return and reason codes

***** DISCONNECT *****
CLC CONTROL,CONTINUE Is everything still OK
BNE EXIT             If CONTROL not 'CONTINUE', stop loop
L    R15,LIALI      Get the Language Interface address
MVC FUNCNTN,DISCON   Get the function to call
CALL (15),(FUNCNTN),VL,MF=(E,CAFCALL)
BAL R14,CHEKCODE     Check the return and reason codes
```

Este código de ejemplo no muestra una tarea que espera en el ECB de terminación de la conexión a Internet (Db2). Si desea realizar dicha tarea, puede codificarla utilizando la z/OS Macro WAIT para supervisar el ECB. Probablemente desee que esta tarea separe el código de muestra si se publica el ECB de terminación. Esta tarea también puede esperar en el inicio de la ECB (Db2). Este ejemplo espera en el ECB de inicio en su propio nivel de tarea.

Este código de ejemplo asume que las variables de la siguiente tabla ya están establecidas:

Tabla 14. Se establecen las variables que el código ensamblador del ejemplo anterior asume

Variable	Uso
LIALI	El punto de entrada que gestiona las solicitudes de servicio de conexión de Db2 .
LISQL	El punto de entrada que gestiona las llamadas SQL.

Tabla 14. Se establecen las variables que el código ensamblador del ejemplo anterior asume (continuación)

Variable	Uso
SSID	El identificador del subsistema de la red de área local inalámbrica (Db2).
TECB	La dirección del BCE de terminación de la Db2 .
SECB	La dirección de la ECB de inicio de Db2 .
RIBPTR	Una palabra completa que CAF establece para contener la dirección RIB.
PLAN	El nombre del plan que se utilizará en la llamada OPEN.
CONTROL	Esta variable se utiliza para detener el procesamiento debido a códigos de devolución o de motivo insatisfactorios. La subrutina CHECKCODE establece este valor.
LLAMADA	Área de parámetros en forma de lista para la macro CALL.

Ejemplo de comprobación de códigos de devolución y códigos de motivo al utilizar CAF

El siguiente código de ejemplo ilustra una forma de comprobar los códigos de retorno y el ECB de terminación de la conexión (Db2) después de cada solicitud de servicio de conexión y llamada SQL. La rutina establece la variable CONTROL para controlar el procesamiento posterior dentro del módulo.

```
*****
* CHEKCODE PSEUDOCODE
*****
*IF TECB is POSTed with the ABTERM or FORCE codes
* THEN
*   CONTROL = 'SHUTDOWN'
*   WRITE 'DB2 found FORCE or ABTERM, shutting down'
* ELSE
*   SELECT (RETCODE)          /* Termination ECB was not POSTed */
*   WHEN (0) ;                /* Look at the return code */
*   WHEN (4) ;                /* Do nothing; everything is OK */
*   WHEN ('00C10824'X)        /* Warning */
*   WHEN ('00F30002'X)        /* Look at the reason code */
*   WHEN ('00C10831'X)        /* Ready for another CAF call */
*   CONTROL = 'RESTART'       /* Start over, from the top */
*   OTHERWISE
*     WRITE 'Found unexpected R0 when R15 was 4'
*     CONTROL = 'SHUTDOWN'
* END INNER-SELECT
* WHEN (8,12)                 /* Connection failure */
*   SELECT (REASCODE)          /* Look at the reason code */
*   WHEN ('00C10831'X)         /* DB2 / CAF release level mismatch*/
*     WRITE 'Found a mismatch between DB2 and CAF release levels'
*   WHEN ('00F30002'X,         /* These mean that DB2 is down but */
*         '00F30012'X)          /* will POST SEC8 when up again */
*   DO
*     WRITE 'DB2 is unavailable. I'll tell you when it is up.'
*     WAIT SEC8               /* Wait for DB2 to come up */
*     WRITE 'DB2 is now available.'
*   END
*   ****
*   /* Insert tests for other DB2 connection failures here. */
*   /* CAF Externals Specification lists other codes you can */
*   /* receive. Handle them in whatever way is appropriate */
*   /* for your application. */
*   ****
*   OTHERWISE                  /* Found a code we're not ready for*/
*     WRITE 'Warning: DB2 connection failure. Cause unknown'
*     CALL DSNALI ('TRANSLATE',SQLCA) /* Fill in SQLCA */
*     WRITE SQLCODE and SQLERRM
*   END INNER-SELECT
* WHEN (200)
```

```

*      WRITE 'CAF found user error. See DSNTRACE data set'
*      WHEN (204)
*          WRITE 'CAF system error. See DSNTRACE data set'
*      OTHERWISE
*          CONTROL = 'SHUTDOWN'
*          WRITE 'Got an unrecognized return code'
*      END MAIN SELECT
*      IF (RETCODE > 4) THEN      /* Was there a connection problem?*/
*          CONTROL = 'SHUTDOWN'
*      END CHEKCODE

*****  

* Subroutine CHEKCODE checks return codes from DB2 and Call Attach.  

* When CHEKCODE receives control, R13 should point to the caller's  

* save area.  

*****  

CHEKCODE DS 0H
    STM R14,R12,12(R13)   Prolog
    ST  R15,RETCODE        Save the return code
    ST  R0,REASCODE        Save the reason code
    LA  R15,SAVEAREA       Get save area address
    ST  R13,4(,R15)        Chain the save areas
    ST  R15,8(,R13)        Chain the save areas
    LR  R13,R15            Put save area address in R13
* ***** HUNT FOR FORCE OR ABTERM *****
    TM  TECB,POSTBIT       See if TECB was POSTed
    BZ  DOCHECKS          Branch if TECB was not POSTed
    CLC TECBCODE(3),QUIESCE Is this "STOP DB2 MODE=FORCE"
    BE  DOCHECKS          If not QUIESCE, was FORCE or ABTERM
    MVC CONTROL,SHUTDOWN  Shutdown
    WRITE 'Found found FORCE or ABTERM, shutting down'
    B   ENDCODE           Go to the end of CHEKCODE
DOCHECKS DS 0H
    EXAMINE RETCODE AND REASCODE
* ***** HUNT FOR 0 *****
    CLC RETCODE,ZERO       Was it a zero?
    BE  ENDCODE           Nothing to do in CHEKCODE for zero
* ***** HUNT FOR 4 *****
    CLC RETCODE,FOUR       Was it a 4?
    BNE HUNT8              If not a 4, hunt eights
    CLC REASCODE,C10831    Was it a release level mismatch?
    BNE HUNT824             Branch if not an 831
    WRITE 'Found a mismatch between DB2 and CAF release levels'
    B   ENDCODE           We are done. Go to end of CHEKCODE
HUNT824 DS 0H
    NOW look for 'CAF reset' reason code
    CLC REASCODE,C10824    Was it 4? Are we ready to restart?
    BNE UNRECOG            If not 824, got unknown code
    WRITE 'CAF is now ready for more input'
    MVC CONTROL,RESTART    Indicate that we should re-CONNECT
    B   ENDCODE           We are done. Go to end of CHEKCODE
UNRECOG DS 0H
    WRITE 'Got RETCODE = 4 and an unrecognized reason code'
    MVC CONTROL,SHUTDOWN  Shutdown, serious problem
    B   ENDCODE           We are done. Go to end of CHEKCODE
* ***** HUNT FOR 8 *****
HUNT8 DS 0H
    CLC RETCODE,EIGHT      Hunt return code of 8
    BE  GOT80R12           GOT80R12
    CLC RETCODE,TWELVE     Hunt return code of 12
    BNE HUNT200
GOT80R12 DS 0H
    FOUND return code of 8 or 12
    WRITE 'Found RETCODE of 8 or 12'
    CLC REASCODE,F30002    Hunt for X'00F30002'
    BE  DB2DOWN
    CLC REASCODE,F30012    Hunt for X'00F30012'
    BE  DB2DOWN
    WRITE 'DB2 connection failure with an unrecognized REASCODE'
    CLC SQLCODE,ZERO       See if we need TRANSLATE
    BNE A4TRANS            If not blank, skip TRANSLATE
* ***** TRANSLATE unrecognized RETCODEs *****
    WRITE 'SQLCODE 0 but R15 not, so TRANSLATE to get SQLCODE'
    L   R15,LIALI           Get the Language Interface address
    CALL (15),(TRANSLAT,SQLCA),VL,MF=(E,CAFSCALL)
    C   R0,C10205           Did the TRANSLATE work?
    BNE A4TRANS            If not C10205, SQLERRM now filled in
    WRITE 'Not able to TRANSLATE the connection failure'
    B   ENDCODE           Go to end of CHEKCODE
A4TRANS DS 0H
    SQLERRM must be filled in to get here
* Note: your code should probably remove the X'FF'
* separators and format the SQLERRM feedback area.

```

```

*      Alternatively, use DB2 Sample Application DSNTIAR
*      to format a message.
        WRITE 'SQLERRM is:' SQLERRM
        B    ENDCCODE          We are done. Go to end of CHEKCODE
DB2DOWN  DS    OH           Hunt return code of 200
        WRITE 'DB2 is down and I will tell you when it comes up'
        WAIT ECB=SECB         Wait for DB2 to come up
        WRITE 'DB2 is now available'
        MVC CONTROL,RESTART   Indicate that we should re-CONNECT
        B    ENDCCODE
*      **** HUNT FOR 200 ****
HUNT200  DS    OH           Hunt return code of 200
        CLC   RETCODE,NUM200   Hunt 200
        BNE   HUNT204
        WRITE 'CAF found user error, see DSNTRACE data set'
        B    ENDCCODE          We are done. Go to end of CHEKCODE
*      **** HUNT FOR 204 ****
HUNT204  DS    OH           Hunt return code of 204
        CLC   RETCODE,NUM204   Hunt 204
        BNE   WASSAT          If not 204, got strange code
        WRITE 'CAF found system error, see DSNTRACE data set'
        B    ENDCCODE          We are done. Go to end of CHEKCODE
*      **** UNRECOGNIZED RETCODE ****
WASSAT   DS    OH
        WRITE 'Got an unrecognized RETCODE'
        MVC   CONTROL,SHUTDOWN Shutdown
        BE    ENDCCODE          We are done. Go to end of CHEKCODE
ENDCCODE DS    OH           Should we shut down?
        L     R4,RETCODE        Get a copy of the RETCODE
        C     R4,FOUR          Have a look at the RETCODE
        BNH   BYEBYE           If RETCODE <= 4 then leave CHEKCODE
        MVC   CONTROL,SHUTDOWN Shutdown
BYEBYE   DS    OH           Wrap up and leave CHEKCODE
        L     R13,4(,R13)       Point to caller's save area
        RETURN (14,12)         Return to the caller

```

Ejemplo de invocar CAF cuando no se especifica la opción del precompilador ATTACH(CAF)

Cada una de las cuatro instalaciones de adjuntos de Db2 contiene un punto de entrada llamado DSNHLI. Cuando se utiliza CAF pero no se especifica la opción del precompilador ATTACH(CAF), las sentencias SQL dan como resultado instrucciones BALR a DSNHLI en el programa. Para encontrar el punto de entrada DSNHLI correcto sin incluir DSNALI en su módulo de carga, codifique una subrutina con el punto de entrada DSNHLI que pase el control al punto de entrada DSNHLI2 en el módulo DSNALI. DSNHLI2 es exclusivo de DSNALI y se encuentra en la misma ubicación en DSNALI que DSNHLI. DSNALI utiliza direccionamiento de 31 bits. Si la aplicación que llama a esta subrutina intermedia utiliza direccionamiento de 24 bits, esta subrutina debe tener en cuenta la diferencia.

En el siguiente ejemplo, LISQL es direccionable porque la CSECT de llamada utilizó el mismo registro 12 que CSECT DSNHLI. Su aplicación también debe establecer direccionabilidad a LISQL.

```

*****+
* Subroutine DSNHLI intercepts calls to LI EP=DSNHLI
*****
DSNHLI  DS    0D
        CSECT
        STM   R14,R12,12(R13) Begin CSECT
        LA    R15,SAVEHLI   Prologue
        ST    R13,4(,R15)   Get save area address
        ST    R15,8(,R13)   Chain the save areas
        LR    R13,R15       Chain the save areas
        L     R15,LISQL     Put save area address in R13
        BASSM R14,R15      Get the address of real DSNHLI
                    Branch to DSNALI to do an SQL call
                    DSNALI is in 31-bit mode, so use
                    BASSM to assure that the addressing
                    mode is preserved.
*
*                                         Restore R13 (caller's save area addr)
*                                         Restore R14 (return address)
*                                         Restore R1-12, NOT R0 and R15 (codes)
L     R13,4(,R13)
L     R14,12(,R13)
RETURN (1,12)

```

Ejemplo de declaraciones de variables al utilizar CAF

El siguiente código de ejemplo muestra las declaraciones de algunas de las variables que se utilizaron en las subrutinas anteriores.

```
***** VARIABLES *****
SECB    DS   F           DB2 Startup ECB
TECB    DS   F           DB2 Termination ECB
LIALI   DS   F           DSNALL Entry Point address
LISQL   DS   F           DSNHLI2 Entry Point address
SSID    DS   CL4          DB2 Subsystem ID. CONNECT parameter
PLAN    DS   CL8          DB2 Plan name. OPEN parameter
TRMOP   DS   CL4          CLOSE termination option (SYNC|ABRT)
FUNCTN  DS   CL12         CAF function to be called
RIBPTR  DS   F           DB2 puts Release Info Block addr here
RETCODE  DS   F           Chekcode saves R15 here
REASCODE DS   F           Chekcode saves R0 here
CONTROL  DS   CL8          GO, SHUTDOWN, or RESTART
SAVEAREA DS  18F          Save area for CHEKCODE
***** CONSTANTS *****
SHUTDOWN DC  CL8'SHUTDOWN'      CONTROL value: Shutdown execution
RESTART   DC  CL8'RESTART'       CONTROL value: Restart execution
CONTINUE   DC  CL8'CONTINUE'     CONTROL value: Everything OK, cont
CODE0    DC  F'0'              SQLCODE of 0
CODE100   DC  F'100'             SQLCODE of 100
QUIESCE  DC  XL3'000008'        TECB postcode: STOP DB2 MODE=QUIESCE
CONNECT   DC  CL12'CONNECT'     Name of a CAF service. Must be CL12!
OPEN     DC  CL12'OPEN'          Name of a CAF service. Must be CL12!
CLOSE    DC  CL12'CLOSE'         Name of a CAF service. Must be CL12!
DISCON   DC  CL12'DISCONNECT'   Name of a CAF service. Must be CL12!
TRANSLAT  DC  CL12'TRANSLATE'   Name of a CAF service. Must be CL12!
SYNC     DC  CL4'SYNC'           Termination option (COMMIT)
ABRT     DC  CL4'ABRT'          Termination option (ROLLBACK)
***** RETURN CODES (R15) FROM CALL ATTACH ****
ZERO    DC  F'0'              0
FOUR    DC  F'4'              4
EIGHT   DC  F'8'              8
TWELVE  DC  F'12'             12 (Call Attach return code in R15)
NUM200   DC  F'200'            200 (User error)
NUM204   DC  F'204'            204 (Call Attach system error)
***** REASON CODES (R00) FROM CALL ATTACH ****
C10205   DC  XL4'00C10205'      Call attach could not TRANSLATE
C10831   DC  XL4'00C10831'      Call attach found a release mismatch
C10824   DC  XL4'00C10824'      Call attach ready for more input
F30002   DC  XL4'00F30002'      DB2 subsystem not up
F30011   DC  XL4'00F30011'      DB2 subsystem not up
F30012   DC  XL4'00F30012'      DB2 subsystem not up
F30025   DC  XL4'00F30025'      DB2 is stopping (REASCODE)
*
*      Insert more codes here as necessary for your application
*
***** SQLCA and RIB *****
EXEC SQL INCLUDE SQLCA
      DSNDRIB          Get the DB2 Release Information Block
***** CALL macro parm list *****
CAFCALL  CALL  ,(*,*,*,*,*,*),VL,MF=L
```

Invocación del recurso de conexión de servicios de recuperación de recursos

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoque RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

Antes de empezar

Antes de invocar RRSAF, realice las siguientes acciones:

- Asegúrese de que el módulo de carga de la interfaz de lenguaje RRSAF, DSNRLI, esté disponible.
- Asegúrese de que su solicitud cumple los requisitos de los programas que acceden a RRSAF.
- Asegúrese de que su aplicación cumple las características generales de entorno para conectarse a Db2.

- Asegúrese de estar familiarizado con los siguientes z/OS conceptos e instalaciones:
 - Convenciones de vinculación de macros CALL y módulos estándar
 - Opciones de direccionamiento y residencia del programa (AMODE y RMODE)
 - Creación y control de tareas; multitarea
 - Instalaciones de recuperación funcional como ESTAE, ESTAI y FRR
 - Técnicas de sincronización como WAIT/POST
 - z/OS Funciones RRS, como SRRCMIT y SRRBACK

Acerca de esta tarea

Las aplicaciones que utilizan RRSAF pueden escribirse en lenguaje ensamblador, C, COBOL, Fortran y PL/I. Al elegir un idioma para codificar su aplicación, tenga en cuenta las siguientes restricciones:

- Si utiliza z/OS macros (ATTACH, WAIT, POST, etc.), elija un lenguaje de programación que las admita.
- La función RRSAF TRANSLATE no está disponible en Fortran. Para utilizar esta función, codifíquela en una rutina escrita en otro lenguaje y, a continuación, llame a esa rutina desde Fortran.

Procedimiento

Para invocar RRSAF:

1. Realice una de las acciones siguientes:

- Invoque explícitamente RRSAF incluyendo en su programa sentencias CALL DSNRLI con las opciones adecuadas.

La primera opción es una función de conexión RRSAF, que describe la acción que desea que RRSAF realice. El efecto de cualquier función depende en parte de las funciones que el programa ya haya realizado.

Para codificar funciones RRSAF en C, COBOL, Fortran, o PL/I, siga las reglas del lenguaje individual para hacer llamadas a rutinas de lenguaje ensamblador. Especifique los parámetros de código de devolución y código de motivo en la lista de parámetros para cada llamada RRSAF.

Requisito: Para las aplicaciones C, C++ y PL/I, también debe incluir en su programa las directivas del compilador que se enumeran en la siguiente tabla, porque DSNRLI es un programa en lenguaje ensamblador.

Tabla 15. Directivas de compilación para incluir en aplicaciones C, C++ y PL/I que contengan sentencias CALL DSNRLI

Idioma	Directiva del compilador para incluir
C	#pragma linkage(dsnrl, OS)
C++	extern "OS" { int DSNRLI(char * functn, ...); }
PL/I	DCL DSNRLI ENTRY OPTIONS(ASM,INTER,RETCODE) ;

- Invoque implícitamente RRSAF incluyendo sentencias SQL o llamadas IFI en su programa tal como lo haría en cualquier programa. La instalación RRSAF establece la conexión con Db2 con los valores predeterminados para el nombre del subsistema, el nombre del plan y el ID de autorización.

Restricción: Si su programa puede realizar su primera llamada SQL desde diferentes módulos con diferentes DBRM, no puede utilizar un nombre de plan predeterminado y, por lo tanto, no puede

invocar implícitamente RRSAF. En su lugar, debe invocar explícitamente RRSAF llamando a la función CREATE THREAD.

Requisito: Si su aplicación incluye llamadas SQL e IFI, debe emitir al menos una llamada SQL antes de emitir cualquier llamada IFI. Esta acción garantiza que su aplicación utilice el plan correcto.

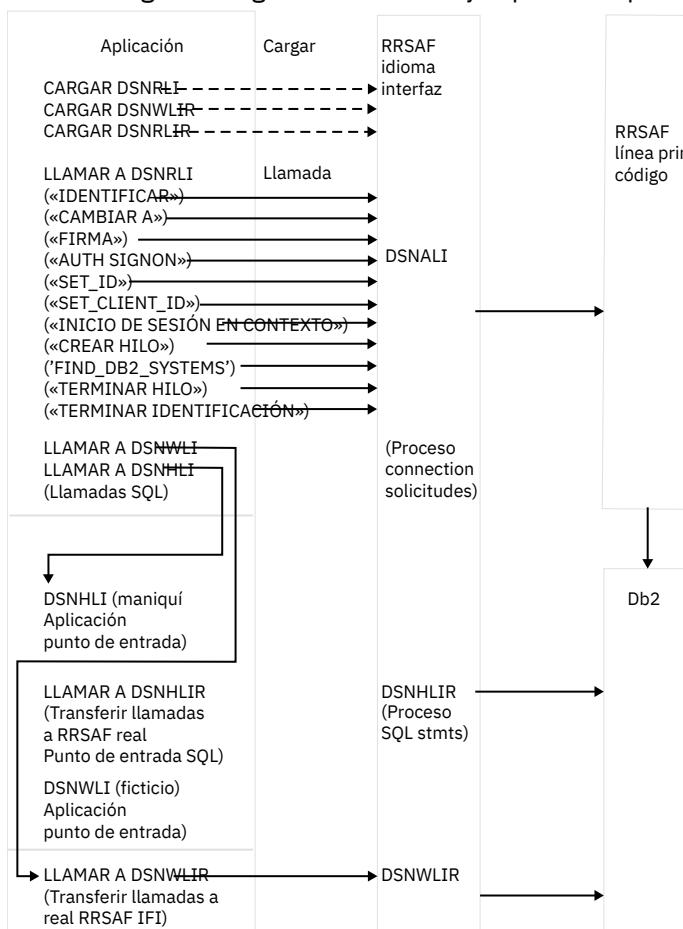
2. Si invocó implícitamente RRSAF, determine si la conexión implícita se realizó correctamente examinando el código de retorno y el código de motivo inmediatamente después de la primera instrucción SQL ejecutable dentro del programa de aplicación. Su programa puede comprobar estos códigos realizando una de las siguientes acciones:

- Examine los registros 0 y 15 directamente.
- Examine el SQLCA y, si el SQLCODE es -981, obtenga el código de retorno y el motivo del texto del mensaje. El código de devolución es el primer token y el código de motivo es el segundo token.

Si la conexión implícita se realiza correctamente, la aplicación puede examinar el SQLCODE para la primera y las siguientes sentencias SQL.

Ejemplo de configuración de RRSAF

La siguiente figura muestra un ejemplo conceptual de invocación y uso de RRSAF.



Recurso de conexión de Resource Recovery Services

Un recurso de conexión habilita programas para comunicarse con Db2. El centro de conexión de los servicios de recuperación de recursos (RRSAF) proporciona dicha conexión para los programas que se ejecutan en z/OS lotes, en primer plano TSO y en segundo plano TSO. RRSAF es una alternativa a CAF y dispone de más funcionalidades.

Un programa de aplicación que utilice RRSAF puede realizar las siguientes acciones:

- Utilice Db2 para procesar sentencias SQL, comandos o llamadas de interfaz de instrumentación (IFI).

- Coordinar las actualizaciones de Db2 con las actualizaciones realizadas por todos los demás gestores de recursos que también utilizan z/OS RRS en un z/OS sistema.
- Utilice el z/OS Sistema de autorización y un producto de seguridad externo, como RACF, para iniciar sesión en Db2 con el ID de autorización de un usuario.
- Inicie sesión en Db2 con un nuevo ID de autorización y una conexión y un plan existentes.
- Acceder a Db2 desde múltiples z/OS tareas en un espacio de direcciones.
- Cambiar un hilo de comunicación (Db2) entre z/OS tareas dentro de un único espacio de direcciones.
- Acceda al IFI (Db2).
- Ejecutar con o sin el programa de monitorización de terminales (TMP) TSO.
- Ejecutarse sin ser una subtarea del procesador de comandos DSN (o de cualquier código e Db2).
- Ejecutar por encima o por debajo de la línea de 16 MB.
- Establecer una conexión explícita con Db2, a través de una interfaz de llamada, con control sobre el estado exacto de la conexión.
- Establezca una conexión implícita con Db2 (con un identificador de subsistema predeterminado y un nombre de plan predeterminado) mediante sentencias SQL o llamadas IFI sin llamar primero a RRSAF.
- Suministrar bloques de control de eventos (ECB), para que Db2 los publique, que indiquen el inicio o la finalización.
- Interceptar códigos de retorno, códigos de motivo y códigos de error de Db2 y traducirlos en mensajes según sea necesario.

RRSAF utiliza z/OS Gestión de transacciones y servicios de recuperación de Resource Manager es (z/OS RRS).

Cualquier tarea en un espacio de direcciones puede establecer una conexión con Db2 a través de RRSAF. Cada bloque de control de tareas (TCB) solo puede tener una conexión con Db2. Una solicitud de servicio de Db2 emitida por un programa que se ejecuta en una tarea determinada está asociada a la conexión de esa tarea con Db2. La solicitud de servicio funciona independientemente de cualquier actividad de otra acción en cualquier otra tarea.

Cada tarea conectada puede ejecutar un plan. Las tareas dentro de un único espacio de direcciones pueden especificar el mismo plan, pero cada instancia de un plan se ejecuta de forma independiente de las demás. Una tarea puede finalizar su plan y ejecutar un plan diferente sin interrumpir por completo su conexión con Db2.

RRSAF no genera estructuras de tareas.

Cuando diseñe su aplicación, tenga en cuenta que el uso de múltiples conexiones simultáneas puede aumentar la posibilidad de bloqueos y de contención de recursos e Db2 .

Restricción: RRSAF no proporciona salidas de procesamiento de atención ni rutinas de recuperación funcional. Puede proporcionar cualquier atención de manipulación y recuperación funcional que necesite su aplicación, pero debe utilizar únicamente rutinas de recuperación de tipo ESTAE/ESTAI.

Un servicio de rastreo proporciona mensajes de diagnóstico que le ayudan a depurar programas y diagnosticar errores en el código RRSAF. La información de seguimiento solo está disponible en un volcado SYSABEND o SYSUDUMP.

Para confirmar el trabajo en aplicaciones RRSAF, utilice la función CPIC SRRCMIT o la sentencia COMMIT de Db2 . Para revertir el trabajo, utilice la función CPIC SRRBACK o la sentencia ROLLBACK de Db2 .

Utilice las siguientes directrices para decidir si utilizar las declaraciones de « Db2 » (comprobación de integridad de transacciones) o las funciones de CPIC para las operaciones de confirmación y reversión:

- Utilice las sentencias COMMIT y ROLLBACK de Db2 , cuando se cumplan todas las condiciones siguientes:
 - El único recurso recuperable al que accede su aplicación son los datos de Db2 , que son gestionados por una única instancia de Db2 .

Db2 Las sentencias COMMIT y ROLLBACK fallan si su aplicación RRSAF accede a recursos recuperables distintos de los datos de Db2 que son gestionados por una única instancia de Db2 .

- El espacio de direcciones desde el que se inicia el procesamiento de puntos de sincronización es el mismo que el espacio de direcciones que está conectado a Db2.
- Si su aplicación accede a otros recursos recuperables, o el procesamiento de puntos de sincronización y el acceso a Db2 se inician desde espacios de direcciones diferentes, utilice SRRCMIT y SRRBACK.

Referencia relacionada

[COMMIT declaración \(Db2 SQL\)](#)

[ROLLBACK declaración \(Db2 SQL\)](#)

Información relacionada

[Utilización de recursos protegidos \(MVS Programming: Callable Services for High-Level Languages\)](#)

Propiedades de las conexiones RRSAF

RRSAF permite que los programas se comuniquen con Db2 para procesar instrucciones SQL, comandos o llamadas IFI.

Restricción: No mezcle conexiones RRSAF con otros tipos de conexión en un único espacio de direcciones. La primera conexión que se realice desde un espacio de direcciones a Db2 determina el tipo de conexión permitida.

La conexión que RRSAF establece con Db2 tiene las propiedades básicas que se enumeran en la siguiente tabla.

Tabla 16. Propiedades de las conexiones RRSAF

Propiedad	Valor	Comentarios
Nombre de conexión	RRSAF	Puede utilizar el comando DISPLAY THREAD para enumerar las aplicaciones RRSAF que tienen el nombre de conexión RRSAF.
Tipo de conexión	RRSAF	Ninguno.

Tabla 16. Propiedades de las conexiones RRSASF (continuación)

Propiedad	Valor	Comentarios
ID de autorización	Identificadores de autorización asociados a cada conexión de Db2	<p>Una conexión debe tener un ID principal y puede tener uno o más ID secundarios. Esos identificadores se utilizan para los siguientes fines:</p> <ul style="list-style-type: none">• Validar el acceso a Db2• Verificar privilegios en objetos de Db2• Asignación de propiedad de objetos de Db2• Identificar al usuario de una conexión para auditoría, rendimiento y rastreo contable. <p>RRSAF se basa en el z/OS Sistema de Autorización (SAF) y un producto de seguridad, como RACF, para verificar y autorizar los ID de autorización. Una aplicación que se conecta a Db2 a través de RRSASF debe pasar esos identificadores a SAF para su verificación y comprobación de autorización. RRSASF recupera los identificadores de SAF.</p>
		<p>Una ubicación puede proporcionar una rutina de salida de autorización para una conexión de Db2 , para cambiar los ID de autorización e indicar si la conexión está permitida. Los valores reales que se asignan a los ID de autorización principal y secundaria pueden diferir de los valores que se proporcionan mediante una solicitud SIGNORE o AUTH SIGNORE. La rutina de salida de inicio de sesión de un sitio web (Db2) puede acceder a los ID de autorización principal y secundario y puede modificar los ID para satisfacer los requisitos de seguridad del sitio web. La rutina de salida también puede indicar si la solicitud de inicio de sesión debe ser aceptada.</p>

Tabla 16. Propiedades de las conexiones RRSAF (continuación)

Propiedad	Valor	Comentarios
Ámbito	RRSAF procesa las conexiones como si cada tarea estuviera completamente aislada. Cuando una tarea solicita una función, RRSAF pasa la función a Db2, independientemente del estado de conexión de otras tareas en el espacio de direcciones. Sin embargo, el programa de aplicación y el subsistema de e Db2 a tienen acceso al estado de conexión de múltiples tareas en un espacio de direcciones.	Ninguno.

Si una aplicación que está conectada a Db2 a través de RRSAF finaliza normalmente antes de que las funciones TERMINATE THREAD o TERMINATE IDENTIFY desasignen el plan, RRS confirma cualquier cambio realizado después del último punto de confirmación. Si la aplicación finaliza de forma anormal antes de que las funciones TERMINATE THREAD o TERMINATE IDENTIFY desasignen el plan, z/OS RRS revierte cualquier cambio realizado después del último punto de confirmación. En cualquier caso, Db2 desasigna el plan, si es necesario, y termina la conexión de la aplicación.

Si Db2 se bloquea mientras se está ejecutando una aplicación, Db2 revierte los cambios al último punto de confirmación. Si Db2 se cierra mientras procesa una solicitud de confirmación, Db2 confirma o revierte cualquier cambio en el siguiente reinicio. La acción tomada depende del estado de la solicitud de confirmación cuando se termina Db2.

Poner a disposición la interfaz de lenguaje RRSAF (DSNRLI)

Antes de poder invocar el servicio de adjuntos de los Servicios de Recuperación de Recursos (RRSAF), primero debe poner a disposición el módulo de carga de la interfaz de lenguaje RRSAF, DSNRLI.

Acerca de esta tarea

Parte de RRSAF es un módulo de carga de datos (Db2) DSNRLI, que también se conoce como módulo de interfaz de lenguaje RRSAF. DSNRLI tiene los alias DSNHLIR y DSNWLIR. El módulo tiene cinco puntos de entrada: DSNRLI, DSNHLI, DSNHLIR, DSNWLI y DSNWLIR. Estos puntos de entrada cumplen las siguientes funciones:

- El punto de entrada DSNRLI gestiona las solicitudes de servicio de conexión explícitas de Db2 .
- DSNHLI y DSNHLIR gestionan las llamadas SQL. Utilice DSNHLI si su programa de aplicación edita RRSAF mediante vínculos. Utilice DSNHLIR si su programa de aplicación carga RRSAF.
- DSNWLI y DSNWLIR gestionan las llamadas de IFI. Utilice DSNWLI si su programa de aplicación edita RRSAF mediante enlaces. Utilice DSNWLIR si su programa de aplicación carga RRSAF.

Procedimiento

Para que DSNRLI esté disponible:

1. Decida cuál de los siguientes métodos desea utilizar para que DSNRLI esté disponible:

- Emisión explícita de solicitudes LOAD cuando se ejecuta el programa.

Al cargar explícitamente el módulo DSNRLI, puede aislar el mantenimiento de su aplicación del futuro IBM mantenimiento de la interfaz de idioma. Si cambia el idioma de la interfaz, es probable que el cambio no afecte a su módulo de carga.

- Incluir el módulo DSNRLI en su módulo de carga cuando edite su programa en modo enlace.

Una desventaja de editar el enlace DSNRLI en su módulo de carga es que si IBM realiza un cambio en DSNRLI, debe volver a editar el enlace de su programa.

De forma alternativa, si utiliza conexiones explícitas a través de CALL DSNALI, puede editar enlaces de su programa con DSNULI, la interfaz de lenguaje universal.

2. Dependiendo del método que haya elegido en el paso 1, realice una de las siguientes acciones:

• **Si desea emitir explícitamente solicitudes LOAD cuando se ejecute su programa:**

En su programa, emita z/OS Cargar solicitudes de servicio para los puntos de entrada DSNRLI y DSNHLIR. Si utiliza los servicios de IFI, también debe cargar DSNWLIR. Guarde la dirección del punto de entrada que devuelve LOAD y utilícela en la macro CALL.

Indique a Db2 qué punto de entrada utilizar de una de las dos formas siguientes:

- Especifique la opción del precompilador ATTACH(RRSAF).

Esta opción hace que Db2 genere llamadas que especifiquen el punto de entrada DSNHLIR.

Restricción: No puede utilizar esta opción si su solicitud está escrita en un Fortran.

- Cree un punto de entrada ficticio llamado DSNHLI dentro de su módulo de carga.

Si no especifica la opción del precompilador ATTACH, el precompilador de Db2 genera llamadas al punto de entrada DSNHLI para cada solicitud SQL. El precompilador no conoce las diferentes funciones de adjuntar archivos (Db2) y es independiente de ellas. Cuando las llamadas generadas por el precompilador de Db2 pasan el control a DSNHLI, el código que corresponde al punto de entrada ficticio debe conservar la lista de opciones que se pasa en el registro 1 y llamar a DSNHLIR con la misma lista de opciones.

• **Si desea incluir el módulo DSNRLI en su módulo de carga al editar el enlace de su programa:**

Incluya DSNRLI en su módulo de carga durante un paso de edición de enlace. Por ejemplo, puede utilizar una instrucción de control del editor de enlaces similar a la siguiente instrucción en su JCL:

```
INCLUDE DB2LIB(DSNRLI) .
```

Al codificar esta declaración, se evita seleccionar inadvertidamente el módulo de interfaz de idioma incorrecto.

Cuando incluya el módulo DSNRLI durante la edición de enlaces, no incluya un punto de entrada DSNHLI ficticio en su programa ni especifique la opción ATTACH del precompilador. El módulo DSNRLI contiene un punto de entrada para DSNHLI, que es idéntico a DSNHLIR, y un punto de entrada para DSNWLRI, que es idéntico a DSNWLIR.

Conceptos relacionados

Ejemplos de programas para RRSAF

El recurso de conexión de servicios de recuperación de recursos (RRSAF) permite a los programas comunicarse con Db2. Puede utilizar RRSAF como alternativa a CAF.

“Interfaz de lenguaje universal (DSNULI)” en la página 121

El subcomponente de la interfaz de lenguaje universal (DSNULI) determina el entorno en tiempo de ejecución y carga y se bifurca dinámicamente al modelo de interfaz de lenguaje adecuado.

Tareas relacionadas

Disponibilidad de la interfaz de lenguaje CAF (DSNALI)

Antes de invocar el recurso de conexión de llamada (CAF), primero debe hacer que DSNALI esté disponible.

Edición de enlaces de una aplicación con DSNULI

Para crear un único módulo de carga que se pueda utilizar en más de un entorno de conexión, puede editar el enlace de su programa o procedimiento almacenado con el módulo de interfaz de lenguaje universal (DSNULI) en lugar de con uno de los módulos de interfaz de lenguaje específicos del entorno (DSNELI, DSNALI, DSNRLI, DSNCLI o DFSLI000).

Requisitos para programas que utilizan RRSAF

El recurso de conexión de servicios de recuperación de recursos (RRSAF) permite a los programas comunicarse con Db2. Antes de invocar RRSAF en su programa, asegúrese de que su programa cumple con todos los requisitos para utilizar RRSAF.

Cuando escriba programas que utilicen RRSAF, asegúrese de que cumplan los siguientes requisitos:

- El programa tiene en cuenta el tamaño del código RRSAF. El código RRSAF requiere unos 10 KB de almacenamiento virtual por espacio de direcciones y 10 KB adicionales por cada TCB que utilice RRSAF.
- Si su entorno local intercepta y reemplaza el z/OS LOAD SVC que RRSAF utiliza, debe asegurarse de que su versión de LOAD gestiona las cadenas de elementos de la lista de carga (LLE) y de entrada de directorio de contenidos (CDE) como la macro estándar z/OS Macro LOAD estándar. RRSAF utiliza z/OS SVC LOAD para cargar un módulo como parte de la inicialización después de su primera solicitud de servicio. El módulo se carga en un almacenamiento protegido contra la recuperación que tiene la clave de protección de paso de trabajo.

Puede preparar programas de aplicación para que se ejecuten en RRSAF de forma similar a como prepara aplicaciones para que se ejecuten en otros entornos, como CICS,, y TSO IMS y TSO. Puede preparar una solicitud RRSAF en el entorno por lotes o mediante el proceso de preparación del programa Db2 . Puede utilizar el sistema de preparación de programas a través de DB2I o a través de DSNH CLIST.

Tareas relacionadas

[Preparación de una aplicación para ejecutarse en Db2 for z/OS](#)

Para preparar y ejecutar aplicaciones que contengan sentencias SQL estáticas incrustadas o sentencias SQL dinámicas, debe procesar, compilar, editar vínculos y vincular las sentencias SQL.

Cómo RRSAF modifica el contenido de los registros

Si no especifica los parámetros de código de retorno y código de motivo en sus llamadas de función RRSAF o siinvoca RRSAF implícitamente, RRSAF pone un código de retorno en el registro 15 y un código de motivo en el registro 0. RRSAF conserva el contenido de los registros 2 a 14.

Si especifica los parámetros de código de devolución y código de motivo, RRSAF coloca el código de devolución en el registro 15 y en el parámetro de código de devolución para adaptarse a los lenguajes de alto nivel que admiten el procesamiento especial de códigos de devolución.

La siguiente tabla resume las convenciones de registro para las llamadas RRSAF.

Tabla 17. Registrar convenciones para llamadas RRSAF

Registrar	Uso
R1	Puntero de lista de parámetros
R13	Dirección del área de seguridad de la persona que llama
R14	Dirección de remitente de la llamada
R15	Dirección del punto de entrada RRSAF

Conexiones implícitas a RRSAF

El centro de conexión de servicios de recuperación de recursos (RRSAF) establece una conexión implícita con Db2 en determinadas situaciones. La conexión se establece si se cumplen las siguientes condiciones: el módulo de carga de la interfaz de lenguaje RRSAF (DSNRLI) está disponible, no se especifica explícitamente la función IDENTIFY en una instrucción CALL DSNRLI en el programa y la aplicación incluye instrucciones SQL o llamadas IFI.

Una conexión implícita hace que RRSAF inicie solicitudes implícitas de IDENTIFICACIÓN y CREACIÓN DE HILO a Db2. Estas solicitudes están sujetas a los mismos códigos de razón y códigos de retorno de Db2 como han especificado explícitamente las solicitudes.

Las conexiones implícitas utilizan los siguientes valores predeterminados:

Nombre de subsistema

El nombre predeterminado que se especifica en el módulo DSNHDECP. RRSAF utiliza el DSNHDECP predeterminado de la instalación, a menos que su propio módulo DSNHDECP esté en una biblioteca en una declaración STEPLIB de la concatenación JOBLIB o en la lista de enlaces. En un grupo de intercambio de datos, el nombre predeterminado del subsistema es el nombre del archivo adjunto del grupo.

Asegúrese de que conoce el nombre predeterminado y que designa el subsistema específico de Db2 que desea utilizar.

Nombre de plan

El nombre de miembro del módulo de solicitud de base de datos (DBRM) que Db2 produjo cuando precompiló el programa fuente que contiene la primera llamada SQL.

ID de autorización

El ID de usuario de 7 bytes que está asociado con el espacio de direcciones, a menos que una función autorizada haya creado un elemento de entorno de acceso (ACEE) para el espacio de direcciones. Si una función autorizada ha creado un ACEE, Db2 pasa el ID de usuario de 8 bytes desde el ACEE.

Para una solicitud de conexión implícita, su aplicación no debe especificar explícitamente ni la función IDENTIFY ni la función CREATE THREAD. Su aplicación puede ejecutar otras llamadas RRSAF explícitas después de que se haya establecido la conexión implícita. Una conexión implícita no realiza ningún procesamiento de INICIO DE SESIÓN. Su aplicación puede ejecutar la función SIGNON en cualquier punto de consistencia. Para finalizar una conexión implícita, debe utilizar las llamadas de función adecuadas.

Para solicitudes de conexión implícitas, el registro 15 contiene el código de retorno y el registro 0 contiene el código de motivo. El código de devolución y el código de motivo también se encuentran en el texto del mensaje para SQLCODE -981.

Conceptos relacionados

Resumen del comportamiento de RRSAF

El efecto de cualquier función Recurso de conexión de Recovery Resource Services (RRSAF) depende en parte de las funciones que el programa ya ha ejecutado. Debe planificar las llamadas a función RRSAF que realiza su programa para evitar errores y problemas estructurales importantes en su aplicación.

Información relacionada

-981 (códigos de la Db2)

Lista de parámetros de la sentencia CALL DSNRLI

La sentencia CALL DSNRLI invoca explícitamente RRSAF. Cuando incluye sentencias CALL DSNRLI en el programa, debe especificar todos los parámetros que van antes del parámetro de código de retorno.

En las sentencias CALL DSNRLI, no se puede omitir ninguno de los parámetros que van antes del parámetro del código de retorno codificando ceros o espacios en blanco. No existen valores predeterminados para esos parámetros para solicitudes de conexión explícitas. Los valores predeterminados se proporcionan solo para conexiones implícitas. Todos los parámetros que comienzan con el parámetro de código de retorno son opcionales.

Cuando desee utilizar el valor predeterminado para un parámetro pero especifique parámetros posteriores, codifique la instrucción CALL DSNRLI de la siguiente manera:

- Para el lenguaje C, cuando codifica sentencias CALL DSNRLI en C, debe especificar la dirección de cada parámetro, utilizando el operador "dirección de" (&), y no el parámetro en sí. Por ejemplo, para pasar el parámetro pklistptr en el "CREATE THREAD", especifique la dirección del puntero de 4 bytes a la estructura (&pklistptr):

```
fnret=dsnrlri(&crthrdfn[0], &plan[0], &collid[0], &reuse[0],  
    &retcode, &reascode, &pklistptr);
```

- Para todos los idiomas excepto el lenguaje ensamblador, código cero para ese parámetro en la instrucción CALL DSNRLI. Por ejemplo, supongamos que está codificando una llamada IDENTIFY en un

programa COBOL y desea especificar todos los parámetros excepto el parámetro de código de retorno. Puede escribir una declaración similar a la siguiente:

```
CALL 'DSNRLI' USING IDFYFN SSNM RIBPTR EIBPTR TERMECB STARTECB
      BY CONTENT ZERO BY REFERENCE REASCODE.
```

- Para el lenguaje ensamblador, codifique una coma para ese parámetro en la instrucción CALL DSNRLI. Por ejemplo, supongamos que está codificando una llamada IDENTIFY y desea especificar todos los parámetros excepto el parámetro de código de retorno. Puede escribir una declaración similar a la siguiente:

```
CALL DSNRLI, (IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,,REASCODE)
```

Para los programas ensambladores que invocan RRSAF, utilice una lista de parámetros estándar para un z/OS CALL. El registro 1 debe contener la dirección de una lista de punteros a los parámetros. Cada puntero es una dirección de 4 bytes. La última dirección debe contener el valor 1 en el bit de orden superior.

Resumen del comportamiento de RRSAF

El efecto de cualquier función Recurso de conexión de Recovery Resource Services (RRSAF) depende en parte de las funciones que el programa ya ha ejecutado. Debe planificar las llamadas a función RRSAF que realiza su programa para evitar errores y problemas estructurales importantes en su aplicación.

Las siguientes tablas resumen el comportamiento de RRSAF después de varias entradas de programas de aplicación. El contenido de cada celda de la tabla indica el resultado de llamar a la función de la primera columna para esa fila, seguida de la función del encabezado de la columna actual. Por ejemplo, si emite TERMINATE THREAD y luego IDENTIFY, RRSAF devuelve el código de motivo X'00C12201'. Utilice estas tablas para comprender el orden en el que su aplicación debe emitir llamadas RRSAF, sentencias SQL y solicitudes IFI.

La función RRSAF FIND_DB2_SYSTEMS se omite en estas tablas, porque no afecta al funcionamiento de ninguna de las demás funciones

La siguiente tabla resume el comportamiento de RRSAF cuando la siguiente llamada es a la función IDENTIFY, la función SWITCH TO, la función SIGNORE o la función CREATE THREAD.

Tabla 18. Efecto del orden de llamada cuando la siguiente llamada es IDENTIFICAR, CAMBIAR A, INICIAR SESIÓN o CREAR HILO

función Next				
función Previous	IDENTIFY	Cambiar a		
Vacio: primera llamada	IDENTIFY	X'00C12205' ¹	X'00C12204' ¹	X'00C12204' ¹
IDENTIFY	X'00 F30049'1	Cambiar a ssnm	Iniciar sesión ²	X'00C12217' ¹
Cambiar a	IDENTIFY	Cambiar a ssnm	Iniciar sesión ²	Crear hebra
INICIO DE SESIÓN, INICIO DE SESIÓN AUTOMÁTICO o INICIO DE SESIÓN EN CONTEXTO	X'00 F30049'1	Cambiar a ssnm	Iniciar sesión ²	Crear hebra
Crear hebra	X'00 F30049'1	Cambiar a ssnm	Iniciar sesión ²	X'00C12202' ¹
Terminar hebra	X'00C12201' ¹	Cambiar a ssnm	Iniciar sesión ²	Crear hebra
IFI	X'00 F30049'1	Cambiar a ssnm	Iniciar sesión ²	X'00C12202' ¹

Tabla 18. Efecto del orden de llamada cuando la siguiente llamada es IDENTIFICAR, CAMBIAR A, INICIAR SESIÓN o CREAR HILO (continuación)

función Previous	función Next			
	IDENTIFY	Cambiar a	INICIO DE SESIÓN, INICIO DE SESIÓN AUTOMÁTICO o INICIO DE SESIÓN EN CONTEXTO	Crear hebra
SQL	X'00 F30049' 1	Cambiar a ssnm	X'00F30092' 1 3	X'00C12202' 1
SRRCMIT o SRRBACK	X'00 F30049' 1	Cambiar a ssnm	Iniciar sesión 2	X'00C12202' 1

Notas:

- Los errores se identifican mediante el código de motivo de error de respuesta de red (Db2) que devuelve RRSAF.
- Signon significa la función SIGNON, la función AUTH SIGNON o la función CONTEXT SIGNON.
- Las funciones SIGNON, AUTH SIGNON o CONTEXT SIGNON no están permitidas si se solicita alguna operación SQL después de la función CREATE THREAD o después de la última solicitud SRRCMIT o SRRBACK.

La siguiente tabla resume el comportamiento de RRSAF cuando la siguiente llamada es una instrucción SQL o una llamada IFI o a la función TERMINATE THREAD, la función TERMINATE IDENTIFY o la función TRANSLATE.

Tabla 19. Efecto del orden de llamada cuando la siguiente llamada es SQL o IFI, TERMINATE THREAD, TERMINATE IDENTIFY o TRANSLATE

función Previous	función Next			
	SQL o IFI	Terminar hebra	TERMINAR IDENTIFICAR	TRANSLATE
Vacio: primera llamada	Llamada SQL o IFI4	X'00C12204' 1	X'00C12204' 1	X'00C12204' 1
IDENTIFY	Llamada SQL o IFI4	X'00C12203' 1	TERMINAR IDENTIFICAR	TRANSLATE
Cambiar a	Llamada SQL o IFI4	Terminar hebra	TERMINAR IDENTIFICAR	TRANSLATE
INICIO DE SESIÓN, INICIO DE SESIÓN AUTOMÁTICO o INICIO DE SESIÓN EN CONTEXTO	Llamada SQL o IFI4	Terminar hebra	TERMINAR IDENTIFICAR	TRANSLATE
Crear hebra	Llamada SQL o IFI4	Terminar hebra	TERMINAR IDENTIFICAR	TRANSLATE
Terminar hebra	Llamada SQL o IFI 4	X'00C12203' 1	TERMINAR IDENTIFICAR	TRANSLATE
IFI	Llamada SQL o IFI4	Terminar hebra	TERMINAR IDENTIFICAR	TRANSLATE
SQL	Llamada SQL o IFI4	X'00F30093' 1 2	X'00F30093' 1 3	TRANSLATE
SRRCMIT o SRRBACK	Llamada SQL o IFI4	Terminar hebra	TERMINAR IDENTIFICAR	TRANSLATE

Tabla 19. Efecto del orden de llamada cuando la siguiente llamada es SQL o IFI, TERMINATE THREAD, TERMINATE IDENTIFY o TRANSLATE (continuación)

función Previous	función Next			
	SQL o IFI	Terminar hebra	TERMINAR IDENTIFICAR	TRANSLATE
Notas:				
	1. Los errores se identifican mediante el código de motivo de error de respuesta de red (Db2) que devuelve RRSAF.	2. TERMINATE THREAD no está permitido si se solicitan operaciones SQL después de la función CREATE THREAD o después de la última solicitud SRRCMIT o SRRBACK.	3. TERMINATE IDENTIFY no está permitido si se solicitan operaciones SQL después de la función CREATE THREAD o después de la última solicitud SRRCMIT o SRRBACK.	4. Si utiliza una conexión implícita a RRSAF y emite llamadas SQL o IFI, RRSAF emite solicitudes implícitas IDENTIFY y CREATE THREAD. Si continúa con las declaraciones RRSAF explícitas, debe seguir el orden estándar de las llamadas RRSAF explícitas. La conexión implícita a RRSAF no causa una solicitud de INICIO DE SESIÓN implícita. Por lo tanto, es posible que tenga que emitir una solicitud de INICIO DE SESIÓN explícita para cumplir con el requisito de pedido estándar. Por ejemplo, una instrucción SQL seguida de una solicitud explícita TERMINATE THREAD da como resultado un error. Debe emitir una solicitud explícita de SIGNON antes de emitir la solicitud de TERMINATE THREAD.

Conceptos relacionados

[X'C1.....' códigos \(Códigos de Db2\)](#)
[X'F3.....' códigos \(Códigos de Db2\)](#)

Funciones de conexión RRSAF

Una función de conexión del recurso de conexión de Resource Recovery Services (RRSAF) especifica la acción que desea que realice RRSAF. Especifique estas funciones cuando invoque RRSAF a través de sentencias CALL DSNRLI.

Conceptos relacionados

[Lista de parámetros de la sentencia CALL DSNRLI](#)

La sentencia CALL DSNRLI invoca explícitamente RRSAF. Cuando incluye sentencias CALL DSNRLI en el programa, debe especificar todos los parámetros que van antes del parámetro de código de retorno.

Resumen del comportamiento de RRSAF

El efecto de cualquier función Recurso de conexión de Recovery Resource Services (RRSAF) depende en parte de las funciones que el programa ya ha ejecutado. Debe planificar las llamadas a función RRSAF que realiza su programa para evitar errores y problemas estructurales importantes en su aplicación.

Función IDENTIFICAR para RRSAF

La función IDENTIFY de RRSAF inicializa una conexión con Db2.

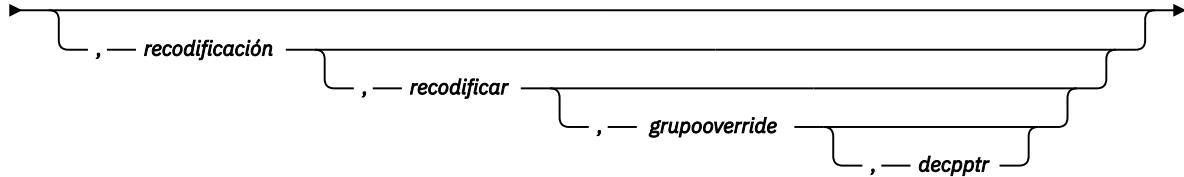
La función IDENTIFICAR establece la tarea del llamante como usuario de los servicios de Db2 . Si ninguna otra tarea en el espacio de direcciones está actualmente conectada al subsistema especificado, la función IDENTIFY también inicializa el espacio de direcciones para comunicarse con los espacios de direcciones e Db2 . La función IDENTIFY establece la autorización de memoria cruzada del espacio de direcciones a Db2 y construye bloques de control del espacio de direcciones.

El siguiente diagrama muestra la sintaxis de la función IDENTIFY.

Función IDENTIFICAR DSNRLI

```
► CALL DSNRLI ( — función — , — ss — nm, — ribptr — , — eibptr — , — termecb — , →
```

```
    ►— startecb —►
```



```
    ► ) ►
```

Los parámetros apuntan a las siguientes áreas:

función

Un área de 18 bytes que contiene IDENTIFY seguido de 10 espacios en blanco.

ssnm

Un nombre de subsistema de 4 bytes de Db2 , o un nombre de archivo adjunto de grupo o de subgrupo (si se utiliza en un grupo de intercambio de datos) al que se realiza la conexión. Si el *ssnm* tiene menos de cuatro caracteres, rellénelo a la derecha con espacios en blanco hasta alcanzar una longitud de cuatro caracteres.

ribptr

Un área de 4 bytes en la que RRSAF coloca la dirección del bloque de información de liberación (RIB) después de la llamada. Puede utilizar el RIB para determinar el nivel de liberación del subsistema de la aplicación (Db2) al que está conectada la aplicación. Puede determinar el nivel de modificación dentro del nivel de publicación examinando los campos RIBCNUMB y RIBCINFO. Si el valor del campo RIBCNUMB es mayor que cero, compruebe el campo RIBCINFO para ver los niveles de modificación.

Si el RIB no está disponible (por ejemplo, si *ssnm* nombra un subsistema que no existe), Db2 establece el área de 4 bytes a ceros.

El área a la que apunta el *cursor* está por debajo de la línea de 16 MB.

Este parámetro es necesario. Sin embargo, la solicitud no necesita hacer referencia a la información devuelta.

eibptr

Un área de 4 bytes en la que RRSAF coloca la dirección del bloque de información del entorno (EIB) después de la llamada. El BEI contiene información sobre el entorno, como el grupo de intercambio de datos, el nombre del Db2 Miembro al que se emitió la solicitud IDENTIFY y si se activan nuevas funciones en el subsistema. Si el subsistema de Db2 no está en un grupo de intercambio de datos, RRSAF establece el grupo de intercambio de datos y los nombres de los miembros en blanco. Si el EIB no está disponible (por ejemplo, si *ssnm* nombra un subsistema que no existe), RRSAF pone a cero el área de 4 bytes.

El área a la que apunta *eibptr* está por encima de la línea de 16 MB.

Este parámetro es necesario. Sin embargo, la solicitud no necesita hacer referencia a la información devuelta.

termecb

La dirección del bloque de control de eventos (ECB) de la aplicación que se utiliza para la terminación de la conexión (Db2). Db2 publica este ECB cuando el operador del sistema introduce el comando STOP DB2 o cuando Db2 finaliza de forma anormal. Especifique un valor de 0 si no desea utilizar un ECB de terminación.

El ECB se ignora cuando ya se ha detenido Db2 . El programa de aplicación debe examinar cualquier código de razón RRSAF o Db2 distinto de cero antes de emitir una solicitud WAIT en este ECB.

RRSAF introduce un código POST en el ECB para indicar el tipo de terminación, como se muestra en la siguiente tabla.

Tabla 20. Códigos postales para los tipos de terminación de la red de telefonía móvil (Db2)

Código POSTAL	Tipo de terminación
8	QUIESCE
12	FORCE
16	ABTERM

startecb

La dirección del ECB de inicio de la aplicación. Si Db2 no se ha iniciado cuando la aplicación emite la llamada IDENTIFY, Db2 publica el ECB cuando Db2 se ha iniciado. Si ya se ha iniciado Db2 , se ignora el ECB de inicio. y no se aplica al siguiente inicio de sesión de Db2 . Si no se inicia Db2 y el ECB de inicio está en cola, se ignora el ECB de terminación.

Introduzca un valor de cero si no desea utilizar un ECB de inicio. Db2 no publica más de un ECB de inicio por espacio de direcciones. El ECB que se publica está asociado con la llamada IDENTIFY más reciente desde ese espacio de direcciones. El programa de aplicación debe examinar cualquier código de razón RRSAF o Db2 distinto de cero antes de emitir una solicitud WAIT en este ECB.

retcode

Un área de 4 bytes en la que RRSAF coloca el código de retorno.

Este parámetro es opcional. Si no especifica *el código de retorno*, RRSAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF coloca un código de motivo.

Este parámetro es opcional. Si no especifica *el código de motivo*, RRSAF coloca el código de motivo en el registro 0.

Si especifica *reascode*, también debe especificar *retcode* o su valor predeterminado. Puede especificar un valor predeterminado para *retcode* especificando una coma o un cero, dependiendo del idioma.

anulación de grupo

Un área de 8 bytes que proporciona la aplicación. Este parámetro es opcional. Si no desea que se intente adjuntar un grupo, especifique 'NOGROUP'. Esta cadena indica que el nombre del subsistema especificado por *ssnm* debe utilizarse como nombre de subsistema e Db2 , incluso si *ssnm* coincide con un nombre de adjunto de grupo o de adjunto de subgrupo. Si no se proporciona *groupoverride*, *ssnm* se utiliza como nombre de archivo adjunto de grupo o de subgrupo si coincide con un nombre de archivo adjunto de grupo o de subgrupo.

Si especifica este parámetro en cualquier lenguaje excepto ensamblador, también debe especificar *los parámetros retcode y reascode*. En lenguaje ensamblador, puede omitir los parámetros *retcode* y *reascode* especificando comas como marcadores de posición.

Recomendación: Evite utilizar *el parámetro groupoverride* cuando sea posible, porque limita la capacidad de realizar el enrutamiento dinámico de la carga de trabajo en un Parallel Sysplex. Sin embargo, debe utilizar este parámetro en un entorno de intercambio de datos cuando desee conectarse a un miembro específico de un grupo de intercambio de datos, y el nombre del subsistema de ese miembro sea el mismo que el nombre de la conexión del grupo o de la conexión del subgrupo.

decpptr

Un área de 4 bytes en la que RRSAF debe poner la dirección del DSNHDECP o un módulo predeterminado de aplicación especificado por el usuario que fue cargado por el subsistema *ssnm* cuando se inició ese subsistema. Esta área de 4 bytes es un puntero de 31 bits. Si no se encuentra *ssnm*, el área de 4 bytes se establece en 0.

El área a la que apunta `decpptr` está por encima de la línea de 16 MB.

Si especifica este parámetro en cualquier lenguaje excepto ensamblador, también debe especificar los parámetros `retcode`, `reascode` y `groupoverride`. En lenguaje ensamblador, puede omitir los parámetros `retcode`, `reascode` y `groupoverride` especificando comas como marcadores de posición.

Ejemplo de llamadas a la función IDENTIFY de RRSAF

La siguiente tabla muestra una llamada IDENTIFY en cada idioma.

Tabla 21. Ejemplos de llamadas de IDENTIFICACIÓN de RRSAF

Idioma	Ejemplo de llamada
Assembler	<pre>CALL DSNRLI,(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB, RETCODE,REASCODE,GRPOVER,DECPPTR)</pre>
C1	<pre>fnret=dsnrl(&idfyfn[0],&ssnm[0], &ribptr, &eibptr, &termecb, &startecb, &retcode, &reascode,&grpovert[0],&decpptr);</pre>
COBOL	<pre>CALL 'DSNRLI' USING IDFYFN SSNM RIBPTR EIBPTR TERMECB STARTECB RETCODE REASCODE GRPOVER DECPPTR.</pre>
Fortran	<pre>CALL DSNRLI(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB, RETCODE,REASCODE,GRPOVER,DECPPTR)</pre>
PL/I1	<pre>CALL DSNRLI(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB, RETCODE,REASCODE,GRPOVER,DECPPTR);</pre>

Nota:

1. Para aplicaciones C, C++ y PL/I, debe incluir las directivas de compilación adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

Procesamiento interno para la función IDENTIFICAR

Cuando se llama a la función IDENTIFY, Db2 realiza los siguientes pasos:

1. Db2 determina si el espacio de direcciones del usuario está autorizado para conectarse a Db2. Db2 invoca el z/OS SAF y le pasa un ID de autorización principal al SAF. Esta identificación de autorización es el ID de usuario de 7 bytes asociado al espacio de direcciones, a menos que una función autorizada haya creado un ACEE para el espacio de direcciones. Si una función autorizada ha creado un ACEE, Db2 pasa el ID de usuario de 8 bytes desde el ACEE. SAF llama a un producto de seguridad externo, como RACF, para determinar si la tarea está autorizada a utilizar los siguientes elementos:
 - La clase de recurso de datos de la base de datos (Db2 , CLASS=DSNR)
 - El subsistema de la red de área local inalámbrica (Db2 , SUBSYS=ssnm)
 - Tipo de conexión RRSAF
2. Si la comprobación tiene éxito, Db2 llama a la rutina de salida de conexión de Db2 para realizar una verificación adicional y posiblemente cambiar el ID de autorización.
3. Db2 busca un contexto de confianza coincidente en la memoria caché del sistema y, a continuación, el catálogo basado en los siguientes criterios:
 - El ID de autorización principal coincide con un contexto de confianza SYSTEM AUTHID.
 - El nombre del trabajo o de la tarea iniciada coincide con el atributo JOBNNAME que está definido para el contexto de confianza identificado.

Si se define un contexto de confianza, Db2 comprueba si SECURITY LABEL está definido en el contexto de confianza. Si se define SECURITY LABEL, Db2 verifica la SECURITY LABEL con RACF mediante la solicitud RACROUTE VERIFY. Esta etiqueta de seguridad se utiliza para verificar la seguridad multinivel para SYSTEM AUTHID.

Si se define un contexto de confianza coincidente, Db2 establece la conexión como de confianza. De lo contrario, la conexión se establece sin privilegios adicionales.

4. Db2 y, a continuación, establece el nombre de conexión en RRSAF y el tipo de conexión en RRSAF.

Tareas relacionadas

Invocación del recurso de conexión de servicios de recuperación de recursos

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoque RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

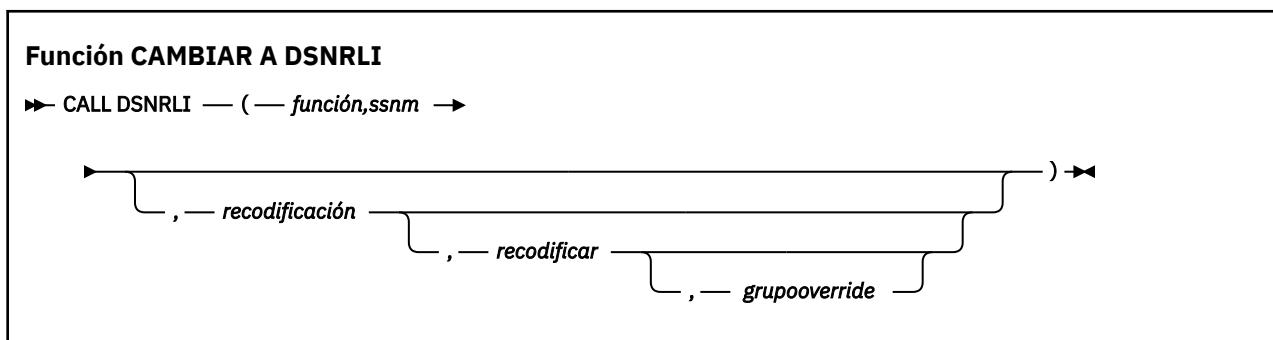
CAMBIAR A función para RRSAF

La función RRSAF SWITCH TO dirige las solicitudes RRSAF, SQL o IFI a un subsistema de Db2 especificado. Utilice la función SWITCH TO (CAMBIAR A) para establecer conexiones con múltiples subsistemas de Db2 desde una sola tarea.

La función SWITCH TO (CAMBIAR A) solo es útil después de una llamada IDENTIFY (IDENTIFICAR) exitosa. Si ha establecido una conexión con un subsistema de la red Db2 , debe realizar una llamada SWITCH TO antes de realizar una llamada IDENTIFY a otro subsistema de la red Db2 . De lo contrario, Db2 devuelve el código de devolución X'200' y el código de motivo X'00C12201'.

La primera vez que se realiza una llamada SWITCH TO a un nuevo subsistema Db2 , Db2 devuelve el código de retorno 4 y el código de motivo X'00C12205' como advertencia para indicar que la tarea actual aún no se ha identificado en el nuevo subsistema Db2 .

El siguiente diagrama muestra la sintaxis de la función SWITCH TO.



Los parámetros apuntan a las siguientes áreas:

función

Un área de 18 bytes que contiene SWITCH TO seguido de nueve espacios en blanco.

ssnm

Un nombre de subsistema de 4 bytes de Db2 , o un nombre de archivo adjunto de grupo o de subgrupo (si se utiliza en un grupo de intercambio de datos) al que se realiza la conexión. Si el *ssnm* tiene menos de cuatro caracteres, rellénelo a la derecha con espacios en blanco hasta alcanzar una longitud de cuatro caracteres.

retcode

Un área de 4 bytes en la que RRSAF coloca el código de retorno.

Este parámetro es opcional. Si no especifica el código de retorno, RRSAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF coloca el código de motivo.

Este parámetro es opcional. Si no especifica *el código de motivo*, RRSAF coloca el código de motivo en el registro 0.

Si especifica este parámetro, también debe especificar *retcode*.

anulación de grupo

Un área de 8 bytes que proporciona la aplicación. Este parámetro es opcional. Si no desea que se intente adjuntar un grupo, especifique 'NOGROUP'. Esta cadena indica que el nombre del subsistema especificado por *ssnm* debe utilizarse como nombre de subsistema e Db2 , incluso si *ssnm* coincide con un nombre de adjunto de grupo o de adjunto de subgrupo. Si no se proporciona *groupoverride*, *ssnm* se utiliza como nombre de archivo adjunto de grupo o de subgrupo si coincide con un nombre de archivo adjunto de grupo o de subgrupo.

Si especifica este parámetro en cualquier lenguaje excepto ensamblador, también debe especificar *los parámetros retcode y reascode*. En lenguaje ensamblador, puede omitir los parámetros *retcode* y *reascode* especificando comas como marcadores de posición.

Recomendación: Evite utilizar *el parámetro groupoverride* cuando sea posible, porque limita la capacidad de realizar el enrutamiento dinámico de la carga de trabajo en un Parallel Sysplex. Sin embargo, debe utilizar este parámetro en un entorno de intercambio de datos cuando desee conectarse a un miembro específico de un grupo de intercambio de datos, y el nombre del subsistema de ese miembro sea el mismo que el nombre de la conexión del grupo o de la conexión del subgrupo.

Ejemplos de llamadas SWITCH TO de RRSAF

La siguiente tabla muestra una llamada SWITCH TO en cada idioma.

Tabla 22. Ejemplos de llamadas SWITCH TO de RRSAF

Idioma	Ejemplo de llamada
Assembler	CALL DSNRLI,(SWITCHFN,SSNM,RETCODE,REASCODE,GRPOVER)
C1	fnret=dsnrlri(&switchfn[0], &ssnm[0], &retcode, &reascode,&grpover[0]);
COBOL	CALL 'DSNRLI' USING SWITCHFN RETCODE REASCODE GRPOVER.
Fortran	CALL DSNRLI(SWITCHFN,RETCODE,REASCODE,GRPOVER)
PL/I1	CALL DSNRLI(SWITCHFN,RETCODE,REASCODE,GRPOVER);

1. Para aplicaciones C, C++ y PL/I, debe incluir las directivas de compilación adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

Ejemplo de uso de la función SWITCH TO para interactuar con múltiples subsistemas de Db2

El siguiente ejemplo muestra cómo puede utilizar la función SWITCH TO para interactuar con tres subsistemas de Db2 .

```
RRSAF calls for subsystem db21:  
  IDENTIFY  
  SIGNON  
  CREATE THREAD  
Execute SQL on subsystem db21  
SWITCH TO db22  
IF retcode = 4 AND reascode = '00C12205'X THEN  
  DO;  
    RRSAF calls on subsystem db22:  
    IDENTIFY
```

```

SIGNON
CREATE THREAD
END;
Execute SQL on subsystem db22
SWITCH TO db23
IF retnode = 4 AND reascode = '00C12205'X THEN
  DO;
    RRSAF calls on subsystem db23:
    IDENTIFY
    SIGNON
    CREATE THREAD
  END;
Execute SQL on subsystem 23
SWITCH TO db21
Execute SQL on subsystem 21
SWITCH TO db22
Execute SQL on subsystem 22
SWITCH TO db21
Execute SQL on subsystem 21
SRRCMIT (to commit the UR)
SWITCH TO db23
Execute SQL on subsystem 23
SWITCH TO db22
Execute SQL on subsystem 22
SWITCH TO db21
Execute SQL on subsystem 21
SRRCMIT (to commit the UR)

```

Tareas relacionadas

[Invocación del recurso de conexión de servicios de recuperación de recursos](#)

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. invoque RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

Función SIGNON para RRSAF

La función SIGNON de RRSAF establece un ID de autorización primario y, opcionalmente, uno o más ID de autorización secundarios para una conexión.

Requisito: Su programa no necesita ser un programa autorizado para emitir la llamada SIGNON. Por esa razón, antes de emitir la llamada SIGNON, debe emitir la RACF macro de interfaz de seguridad externa RACROUTE REQUEST=VERIFY para realizar las siguientes acciones:

- Definir y llenar un ACEE para identificar al usuario del programa.
- Asociar el ACEE con el TCB del usuario.
- Verifique que el usuario esté definido para RACF y autorizado para usar la aplicación.

Por lo general, se emite una llamada SIGNON después de una llamada IDENTIFY y antes de una llamada CREATE THREAD. También puede emitir una llamada SIGNON si la aplicación se encuentra en un punto de consistencia y se cumple una de las siguientes condiciones:

- El valor de *reutilización* en la llamada CREATE THREAD era RESET.
- El valor de *reutilización* en la llamada CREATE THREAD era INITIAL, no hay cursos abiertos, el paquete o plan está vinculado con KEEPDYNAMIC(NO) y todos los registros especiales están en su estado inicial. Si existen cursos abiertos o el paquete o plan está vinculado con KEEPDYNAMIC(YES), puede emitir una llamada SIGNON solo si el ID de autorización principal no ha cambiado.

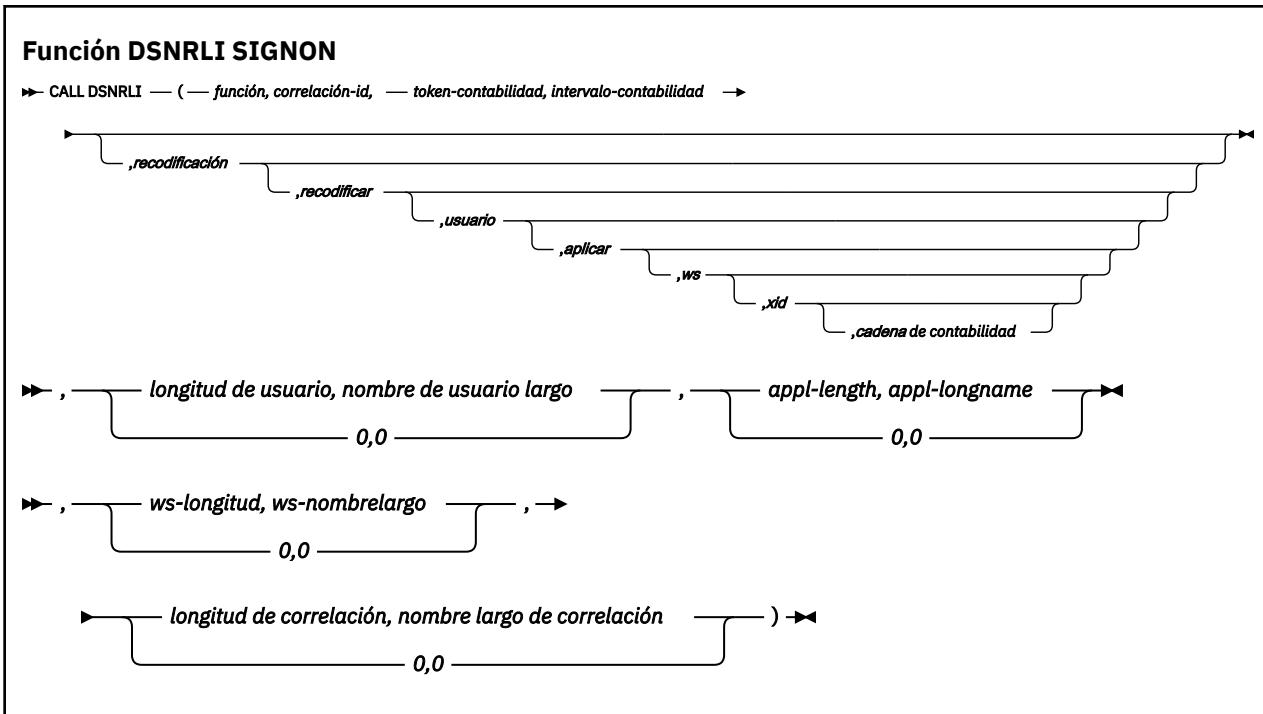
Después de emitir una llamada SIGNON, las siguientes sentencias SQL devuelven un error (SQLCODE -900) si se cumplen las dos condiciones siguientes:

- La conexión se estableció como de confianza cuando se inició.
- La identificación de autorización principal que se utilizó cuando emitió la llamada SIGNON no está autorizada a utilizar la conexión de confianza.

Si se define un contexto de confianza, Db2 comprueba si SECURITY LABEL está definido en el contexto de confianza. Si se define SECURITY LABEL, Db2 verifica la etiqueta de seguridad con RACF mediante la

solicitud RACROUTE VERIFY. Esta etiqueta de seguridad se utiliza para verificar la seguridad multinivel para SYSTEM AUTHID.

El siguiente diagrama muestra la sintaxis de la función SIGNON.



Los parámetros apuntan a las siguientes áreas:

función

Un área de 18 bytes que contiene SIGNON seguido de doce espacios en blanco.

correlation-id

Un área de 12 bytes en la que puede poner un ID de correlación e Db2 . El ID de correlación se muestra en los registros de seguimiento de contabilidad y estadísticas de Db2 . Puede utilizar el ID de correlación para correlacionar unidades de trabajo. Este símbolo aparece en el resultado del comando DISPLAY THREAD. Si no desea especificar un ID de correlación, rellene el área de 12 bytes con espacios en blanco.

contabilidad-token

Un área de 22 bytes en la que puede poner un valor para un token de contabilidad de e Db2 . Este valor se muestra en los registros de seguimiento de contabilidad y estadísticas de Db2 , en el campo QWHTOKN, que está mapeado por DSNDQWHC DSECT. Al establecer el valor del token contable, se establece el valor del registro especial CURRENT CLIENT_ACCTNG. Si el *token de contabilidad* tiene menos de 22 caracteres, debe rellenarlo a la derecha con espacios en blanco hasta alcanzar los 22 caracteres. Si no desea especificar un token de contabilidad, rellene el área de 22 bytes con espacios en blanco.

De forma alternativa, puede cambiar el valor del token de contabilidad de la aplicación (Db2) con las funciones AUTH SIGNON, CONTEXT SIGNON o SET_CLIENT_ID de RRSAF. Puede recuperar el token de contabilidad de la cuenta de cliente (Db2) con el registro especial CURRENT CLIENT_ACCTNG solo si la cadena de contabilidad DDF no está configurada.

intervalo-contable

Área de 6 bytes que especifica cuándo escribe un registro de contabilidad un Db2 .

Si especifica COMMIT en esa área, Db2 escribe un registro de contabilidad cada vez que la aplicación emite SRRCMIT. Este registro contable se escribe al final de la segunda fase de un compromiso de dos fases. Si el intervalo de contabilización es COMMIT y se emite un SRRCMIT mientras hay un cursor retenido abierto, el intervalo de contabilización abarca ese commit y termina en el siguiente punto

final de intervalo de contabilización válido (como el siguiente SRRCMIT que se emite sin cursos retenidos abiertos, finalización de la aplicación o SIGNON con un nuevo ID de autorización).

Si especifica cualquier otro valor, Db2 escribe un registro contable cuando la aplicación finaliza o cuando llama a la función SIGNON con un nuevo ID de autorización.

retcode

Un área de 4 bytes en la que RRSAF coloca el código de retorno.

Este parámetro es opcional. Si no especifica *el código de retorno*, RRSAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF coloca el código de motivo.

Este parámetro es opcional. Si no especifica *el código de motivo*, RRSAF coloca el código de motivo en el registro 0.

Si especifica este parámetro, también debe especificar *retcode*.

USER

Un área de 16 bytes que contiene el ID de usuario del usuario cliente. Puede utilizar este parámetro para proporcionar la identidad del usuario cliente con fines contables y de supervisión. Db2 muestra este ID de usuario en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el ID de usuario se establece el valor del registro especial CURRENT CLIENT_USERID. Si *el usuario* tiene menos de 16 caracteres, debe llenar los espacios en blanco a la derecha hasta alcanzar los 16 caracteres.

Este parámetro es opcional. Si especifica *user*, también debe especificar *retcode* y *reascode*. Si no especifica *usuario*, no se asociará ningún ID de usuario a la conexión.

aplicación

Un área de 32 bytes que contiene el nombre de la aplicación o transacción de la aplicación del usuario. Puede utilizar este parámetro para proporcionar la identidad del usuario cliente con fines contables y de supervisión. Db2 muestra el nombre de la aplicación en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el nombre de la aplicación se establece el valor del registro especial CURRENT CLIENT_APPLNAME. Si *la solicitud* tiene menos de 32 caracteres, debe llenarla a la derecha con espacios en blanco hasta alcanzar los 32 caracteres.

Este parámetro es opcional. Si especifica *appl*, también debe especificar *retcode*, *reascode* y *user*. Si no especifica *appl*, no se asociará ninguna aplicación o transacción con la conexión.

ws

Un área de 18 bytes que contiene el nombre de la estación de trabajo del usuario cliente. Puede utilizar este parámetro para proporcionar la identidad del usuario cliente con fines contables y de supervisión. Db2 muestra el nombre de la estación de trabajo en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el nombre de la estación de trabajo se establece el valor del registro especial CURRENT CLIENT_WRKSTNNAME. Si *ws* tiene menos de 18 caracteres, debe llenarlo a la derecha con espacios en blanco hasta alcanzar los 18 caracteres.

Este campo es opcional. Si especifica *ws*, también debe especificar *retcode*, *reascode*, *user* y *appl*. Si no especifica *ws*, no se asociará ningún nombre de estación de trabajo a la conexión.

xid

Un área de 4 bytes que indica si el hilo forma parte de una transacción global. Un hilo de comunicación (Db2) que forma parte de una transacción global puede compartir bloqueos con otros hilos de comunicación (Db2) que forman parte de la misma transacción global y pueden acceder a los mismos datos y modificarlos. Una transacción global existe hasta que uno de los hilos que forma parte de la transacción global se confirma o se revierte.

Puede especificar uno de los siguientes valores para *xid* :

0

Indica que el hilo no forma parte de una transacción global. El valor 0 debe especificarse como un entero binario.

1

Indica que el hilo es parte de una transacción global y que Db2 debe recuperar el ID de transacción global de RRS. Si ya existe un ID de transacción global para la tarea, el hilo pasa a formar parte de la transacción global asociada. De lo contrario, RRS genera un nuevo ID de transacción global. El valor 1 debe especificarse como un entero binario. De forma alternativa, si desea que Db2 devuelva el ID de transacción global generado al llamante, especifique una dirección en lugar de 1.

dirección

La dirección de 4 bytes de un área en la que se introduce un ID de transacción global para el hilo. Si el ID de transacción global ya existe, el hilo pasa a formar parte de la transacción global asociada. De lo contrario, RRS crea una nueva transacción global con el ID que usted especifique.

Alternativamente, si desea que Db2 genere y devuelva un ID de transacción global, pase la dirección de un ID de transacción global nulo configurando *el campo de formato ID* del ID de transacción global en binario -1 ('FFFFFFF'X). Db2 y luego reemplaza el contenido del área con el ID de transacción generado. El área en la dirección especificada debe estar en almacenamiento grabable y tener una longitud de al menos 140 bytes para acomodar el mayor valor posible de ID de transacción.

La siguiente tabla muestra el formato de un ID de transacción global.

Tabla 23. Formato de un ID de transacción global creado por el usuario

Descripción del campo	Longitud en bytes	Tipo de datos
ID de formato	4	Entero
Longitud del ID de transacción global (1-64)	4	Entero
Longitud del calificador de sucursal (1-64)	4	Entero
ID de transacción global	de 1 a 64	Carácter
Calificador de rama	de 0 a 64	Carácter

cadena de contabilidad

Un campo de un byte de longitud y un área de 255 bytes en los que puede poner un valor para una cadena de contabilidad de Db2 . Este valor se coloca en los registros de seguimiento contable DDF en el campo QMDASQLI, que está mapeado por DSNDQMDA DSECT. Si *la cadena de contabilidad* tiene menos de 255 caracteres, debe rellenarla a la derecha con ceros hasta una longitud de 255 bytes. Los 256 bytes completos se asignan mediante DSNDQMDA DSECT.

Este parámetro es opcional. Si especifica *accounting-string*, también debe especificar *retcode*, *reascode*, *user*, *appl* y *xid*. Si no especifica *una cadena de contabilidad*, no se asociará ninguna cadena de contabilidad a la conexión.

También puede cambiar el valor de la cadena de contabilidad con las funciones RRSAF AUTH SIGNON, CONTEXT SIGNON o SET_CLIENT_ID.

Puede recuperar la parte del sufijo DDF de la cadena de contabilidad con el registro especial CURRENT CLIENT_ACCTNG. La parte del sufijo de *la cadena de contabilidad* puede contener un máximo de 200 caracteres. El campo QMDASFLN contiene la longitud del sufijo de contabilidad y el campo QMDASUFX contiene el valor del sufijo de contabilidad. Si se establece la cadena de contabilidad DDF, no se puede consultar el token de contabilidad con el registro especial CURRENT CLIENT_ACCTNG.

Los siguientes parámetros son opcionales y posicionales. Estos parámetros anulan los valores especificados anteriormente en la lista de parámetros. Para proporcionar un valor para una longitud, par de valores, debe proporcionar un valor o especificar una longitud 0 para los parámetros anteriores en la lista de parámetros.

longitud del usuario, nombre largo del usuario

Un par de parámetros que consta de una longitud entera de 2 bytes y un área de cadena de 128 bytes. Los parámetros se separan mediante una coma. Puede proporcionar el ID de usuario del usuario cliente con fines contables y de supervisión en *user-longname*. Db2 muestra este ID de usuario en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el ID de usuario se establece el valor del registro especial CURRENT CLIENT_USERID. Los espacios en blanco al final del *nombre de usuario* se truncan y se actualiza la longitud en *longitud de usuario*.

Estos parámetros son opcionales, para especificarlos también debe especificar un valor para *accounting-string*. Un valor de 0 en la *longitud del usuario* omite el procesamiento del *nombre largo del usuario*.

Importante: Estos parámetros anulan cualquier valor que se proporcione en *user*.

longitud-aplicación, nombre-largo-aplicación

Un par de parámetros que consta de una longitud entera de 2 bytes y un área de cadena de 255 bytes. Una coma separa los parameters. You Puede proporcionar el nombre de la aplicación o transacción del usuario del cliente para fines de contabilidad y monitoreo en *appl-longname*. Db2 muestra este nombre de aplicación en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el nombre de la aplicación se establece el valor del registro especial CURRENT CLIENT_APPLNAME. Los espacios en blanco al final de *appl-longname* se truncan y se actualiza la longitud en *appl-length*.

Estos parámetros son opcionales, para especificarlos también debe especificar un valor para *user-length*, *user-longname*. Un valor de 0 en *appl-length* omite el procesamiento de *appl-longname*.

Importante: Estos parámetros anulan cualquier valor que se proporcione en la *aplicación*.

longitud de ws, nombre largo de ws

Un par de parámetros que consta de una longitud entera de 2 bytes y un área de cadena de 255 bytes. Los parámetros se separan mediante una coma. Puede proporcionar el nombre de la estación de trabajo del usuario cliente con fines contables y de supervisión en *ws-longname*. Db2 muestra el nombre de esta estación de trabajo en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el nombre de la estación de trabajo se establece el valor del registro especial CURRENT CLIENT_WRKSTNNNAME. Los espacios en blanco al final de *ws-longname* se truncan y se actualiza la longitud en *ws-length*.

Estos parámetros son opcionales, para especificarlos también debe especificar un valor para *appl-length*, *appl-longname*. Un valor de 0 en *ws-length* omite el procesamiento de *ws-longname*.

Importante: Estos parámetros anulan cualquier valor que se proporcione en *ws*.

longitud de correlación, nombre largo de correlación

Un par de parámetros que consta de una longitud entera de 2 bytes y un área de cadena de 255 bytes. Los parámetros se separan mediante una coma. Puede proporcionar un valor único para correlacionar los nombres de sus procesos de negocio con los hilos de correlación (Db2) en *correlation-longname*. Db2 muestra este token de correlación en el resultado del comando DISPLAY THREAD DETAIL. El registro especial CURRENT CLIENT_CORR_TOKEN contiene el token de correlación de cliente. Los espacios en blanco finales en *correlación-longname* se truncan y se actualiza la longitud en *correlación-longitud*.

Estos parámetros son opcionales, para especificarlos también debe especificar un valor para *ws-length*, *ws-longname*. Un valor de 0 en la *longitud de correlación* omite el procesamiento de la *longitud de correlación*.

También puede cambiar el valor del token de correlación de cliente con la función RRSAF AUTH SIGNON y la función SET_CLIENT_ID.

Ejemplo de llamadas de RRSAF SIGNON

La siguiente tabla muestra una llamada de INICIO DE SESIÓN en cada idioma.

Tabla 24. Ejemplos de llamadas RRSAF SIGNON

Idioma	Ejemplo de llamada
ensamblador	CALL DSNRLI,(SGNONFN,CORRID,ACCTTKN,ACCTINT, RETCODE,REASCODE,USERID,APPLNAME,WSNAME,XIDPTR)
C1	fnret=dsnrl1(&sgnonfn[0], &corrid[0], &accttkn[0], &acctint[0], &retcode, &reascode, &userid[0], &applname[0], &wsname[0], &xidptr);
COBOL	CALL 'DSNRLI' USING SGNONFN CORRID ACCTTKN ACCTINT RETCODE REASCODE USERID APPLNAME WSNAME XIDPTR.
Fortran	CALL DSNRLI(SGNONFN,CORRID,ACCTTKN,ACCTINT, RETCODE,REASCODE,USERID,APPLNAME,WSNAME,XIDPTR)
PL/I	CALL DSNRLI(SGNONFM,CORRID,ACCTTKN,ACCTINT, RETCODE,REASCODE,USERID,APPLNAME,WSNAME,XIDPTR);

Nota:

1. Para aplicaciones C, C++ y PL/I, debe incluir las directivas de compilación adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

El siguiente ejemplo muestra una llamada SIGNON en C¹ con todos los parámetros pasados. Los parámetros que son números se pasan como enteros y las cadenas como matrices de caracteres. En este ejemplo, si `&useridlen` es mayor que 0, entonces el valor del registro especial CURRENT_CLIENT_USERID es el valor que se almacena en `&luserid[0]`.

```
fnret=dsnrl1(&sgnonfn[0],&corrid[0],&accttkn[0],&acctint[0],&retcode,&reascode,  
&userid[0],&applname[0],&wsname[0],&xidptr,&lacctngid[0],  
&useridlen,&luserid[0],&applidlen,&lapplid[0],&wsidlen,&lwsid[0],  
&corrtkidlen,&lcorrtkid[0]);
```

Nota:

1. Para aplicaciones C, debe incluir las directivas de compilador adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

Tareas relacionadas

[Invocación del recurso de conexión de servicios de recuperación de recursos](#)

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoque RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

Referencia relacionada

[RACROUTE REQUEST=VERIFY \(formato estándar\) \(Referencia de macro RACROUTE de Security Server\)](#)

Función SIGNON para RRSAF

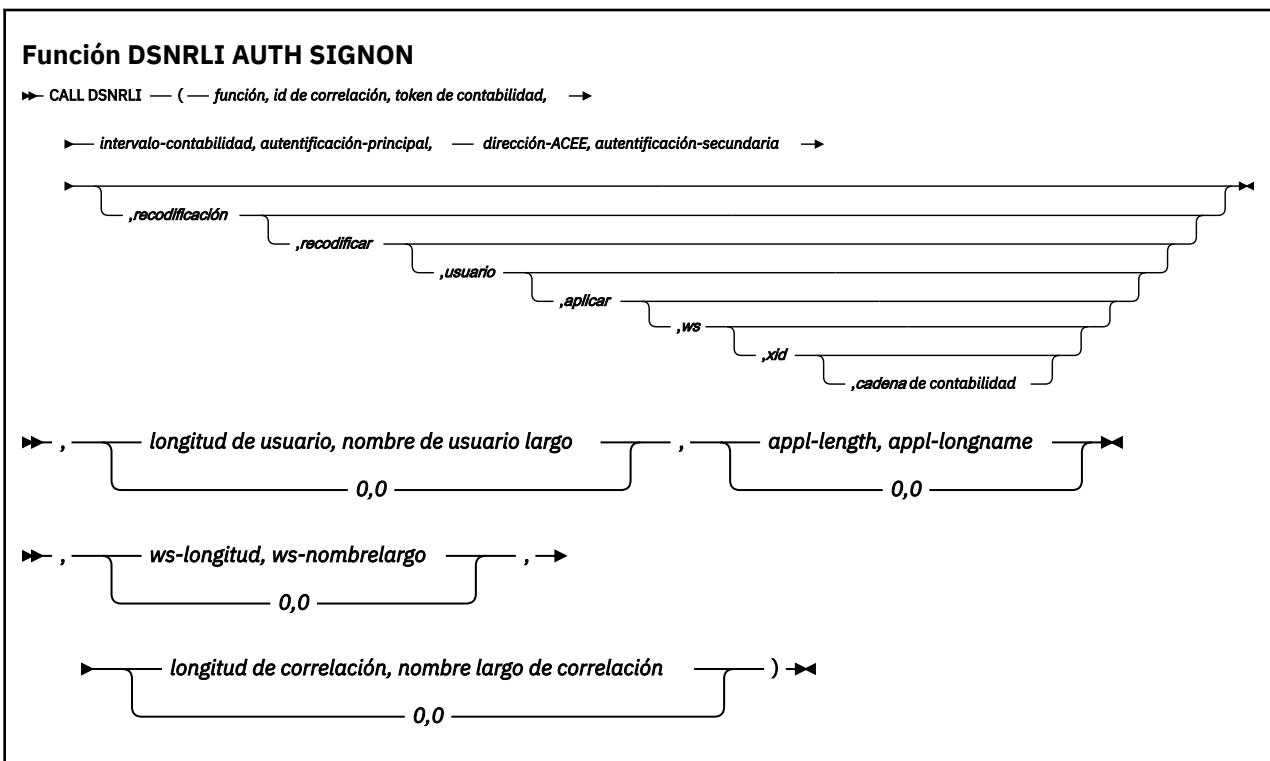
La función RRSAF AUTH SIGNON permite a un programa de autorización de APF pasar un ID a Db2.

Un programa autorizado por APF puede pasar a Db2 un ID de autorización principal y, opcionalmente, uno o más ID de autorización secundarios, o un ACEE que se utiliza para la comprobación de la autorización. Estas identificaciones se asocian entonces a la conexión.

Por lo general, se emite una llamada AUTH SIGNON después de una llamada IDENTIFY y antes de una llamada CREATE THREAD. También puede emitir una llamada AUTH SIGNON si la aplicación se encuentra en un punto de consistencia y se cumple una de las siguientes condiciones:

- El valor de *reutilización* en la llamada CREATE THREAD era RESET.
- El valor de *reutilización* en la llamada CREATE THREAD era INITIAL, no hay cursorios abiertos, el paquete o plan está vinculado con KEEPDYNAMIC(NO) y todos los registros especiales están en su estado inicial. Si existen cursorios abiertos o el paquete o plan está vinculado con KEEPDYNAMIC(SÍ), se permite una llamada SIGNON solo si el ID de autorización principal no ha cambiado.

El siguiente diagrama muestra la sintaxis de la función AUTH SIGNON.



Los parámetros apuntan a las siguientes áreas:

función

Un área de 18 bytes que contiene AUTH SIGNON seguido de siete espacios en blanco.

correlation-id

Un área de 12 bytes en la que puede poner un ID de correlación e Db2 . El ID de correlación se muestra en los registros de seguimiento de contabilidad y estadísticas de Db2 . Puede utilizar el ID de correlación para correlacionar unidades de trabajo. Este símbolo aparece en el resultado del comando DISPLAY THREAD. Si no desea especificar un ID de correlación, rellene el área de 12 bytes con espacios en blanco.

contabilidad-token

Un área de 22 bytes en la que puede poner un valor para un token de contabilidad de e Db2 . Este valor se muestra en los registros de seguimiento de contabilidad y estadísticas de Db2 , en el campo QWHCTOKN, que está mapeado por DSNDQWHC DSECT. Al establecer el valor del token contable, se establece el valor del registro especial CURRENT CLIENT_ACCTNG. Si el *token de contabilidad* tiene menos de 22 caracteres, debe rellenarlo a la derecha con espacios en blanco hasta alcanzar los 22 caracteres. Si no desea especificar un token de contabilidad, rellene el área de 22 bytes con espacios en blanco.

También puede cambiar el valor del token de contabilidad de la aplicación (Db2) con las funciones RRSAF SIGNON, CONTEXT SIGNON o SET_CLIENT_ID. Puede recuperar el token de contabilidad de la cuenta de cliente (Db2) con el registro especial CURRENT CLIENT_ACCTNG solo si la cadena de contabilidad DDF no está configurada.

intervalo-contable

Un área de 6 bytes que especifica cuándo escribe un registro de contabilidad Db2 .

Si especifica COMMIT en esa área, Db2 escribe un registro de contabilidad cada vez que la aplicación emite SRRCMIT. Este registro contable se escribe al final de la segunda fase de un compromiso de dos fases. Si el intervalo de contabilización es COMMIT y se emite un SRRCMIT mientras hay un cursor retenido abierto, el intervalo de contabilización abarca ese commit y termina en el siguiente punto final de intervalo de contabilización válido (como el siguiente SRRCMIT que se emite sin cursos retenidos abiertos, finalización de la aplicación o SIGNON con un nuevo ID de autorización).

Si especifica cualquier otro valor, Db2 escribe un registro contable cuando la aplicación finaliza o cuando llama a la función SIGNON con un nuevo ID de autorización.

primary-authid

Un área de 8 bytes en la que puede poner un ID de autorización principal. Si no va a pasar el ID de autorización a Db2 explícitamente, ponga X'00' o un espacio en blanco en el primer byte del área.

Dirección ACEE

La dirección de 4 bytes de un ACEE que usted pasa a Db2. Si no desea proporcionar una ACEE, especifique 0 en este campo.

autentificación-secundaria

Un área de 8 bytes en la que puede poner un ID de autorización secundario. Si no pasa el ID de autorización a Db2 explícitamente, ponga X'00' o un espacio en blanco en el primer byte del área. Si introduce un ID de autorización secundario, también debe introducir un ID de autorización principal.

retcode

Un área de 4 bytes en la que RRSAF coloca el código de retorno.

Este parámetro es opcional. Si no especifica *el código de retorno*, RRSAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF coloca el código de motivo.

Este parámetro es opcional. Si no especifica *el código de motivo*, RRSAF coloca el código de motivo en el registro 0.

Si especifica *reascoder*, también debe especificar *retcode*.

USER

Un área de 16 bytes que contiene el ID de usuario del usuario cliente. Puede utilizar este parámetro para proporcionar la identidad del usuario cliente con fines contables y de supervisión. Db2 muestra este ID de usuario en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el ID de usuario se establece el valor del registro especial CURRENT CLIENT_USERID. Si *el usuario* tiene menos de 16 caracteres, debe llenar los espacios en blanco a la derecha hasta alcanzar los 16 caracteres.

Este parámetro es opcional. Si especifica *user*, también debe especificar *retcode* y *reascode*. Si no especifica este parámetro, no se asociará ningún ID de usuario a la conexión.

aplicación

Un área de 32 bytes que contiene el nombre de la aplicación o transacción de la aplicación del usuario. Puede utilizar este parámetro para proporcionar la identidad del usuario cliente con fines contables y de supervisión. Db2 muestra el nombre de la aplicación en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el nombre de la aplicación se establece el valor del registro especial CURRENT CLIENT_APPLNAME. Si *la solicitud* tiene menos de 32 caracteres, debe llenarla a la derecha con espacios en blanco hasta alcanzar los 32 caracteres.

Este parámetro es opcional. Si especifica *appl*, también debe especificar *retcode*, *reascode* y *user*. Si no especifica este parámetro, no se asociará ninguna aplicación o transacción a la conexión.

ws

Un área de 18 bytes que contiene el nombre de la estación de trabajo del usuario cliente. Puede utilizar este parámetro para proporcionar la identidad del usuario cliente con fines contables y de supervisión. Db2 muestra el nombre de la estación de trabajo en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el nombre de la estación de trabajo se establece el valor del registro especial CURRENT CLIENT_WRKSTNNNAME. Si *ws* tiene menos de 18 caracteres, debe rellenarlo a la derecha con espacios en blanco hasta alcanzar los 18 caracteres.

Este parámetro es opcional. Si especifica *ws*, también debe especificar *retcode*, *reascode*, *user* y *appl*. Si no especifica este parámetro, no se asociará ningún nombre de estación de trabajo a la conexión.

También puede cambiar el valor del nombre de la estación de trabajo con las funciones RRSAF SIGNON, CONTEXT SIGNON o SET_CLIENT_ID. Puede recuperar el nombre de la estación de trabajo con el registro especial CURRENT CLIENT_WRKSTNNNAME.

xid

Un área de 4 bytes que indica si el hilo forma parte de una transacción global. Un hilo de comunicación (Db2) que forma parte de una transacción global puede compartir bloqueos con otros hilos de comunicación (Db2) que forman parte de la misma transacción global y pueden acceder a los mismos datos y modificarlos. Una transacción global existe hasta que uno de los hilos que forma parte de la transacción global se confirma o se revierte.

Puede especificar uno de los siguientes valores para *xid* :

0

Indica que el hilo no forma parte de una transacción global. El valor 0 debe especificarse como un entero binario.

1

Indica que el hilo es parte de una transacción global y que Db2 debe recuperar el ID de transacción global de RRS. Si ya existe un ID de transacción global para la tarea, el hilo pasa a formar parte de la transacción global asociada. De lo contrario, RRS genera un nuevo ID de transacción global. El valor 1 debe especificarse como un entero binario. De forma alternativa, si desea que Db2 devuelva el ID de transacción global generado al llamante, especifique una dirección en lugar de 1.

dirección

La dirección de 4 bytes de un área en la que se introduce un ID de transacción global para el hilo. Si el ID de transacción global ya existe, el hilo pasa a formar parte de la transacción global asociada. De lo contrario, RRS crea una nueva transacción global con el ID que usted especifique.

Alternativamente, si desea que Db2 genere y devuelva un ID de transacción global, pase la dirección de un ID de transacción global nulo configurando el campo de formato *ID* del ID de transacción global a binario -1 ('FFFFFFFX'). Db2 y, a continuación, sustituye el contenido del área por el ID de transacción generado. El área en la dirección especificada debe estar en almacenamiento grabable y tener una longitud de al menos 140 bytes para acomodar el mayor valor posible de ID de transacción.

El formato de un ID de transacción global se muestra en la descripción de la función RRSAF SIGNON.

cadena de contabilidad

Un campo de 1 byte de longitud y un área de 255 bytes en la que puede poner un valor para una cadena de contabilidad de e Db2 . Este valor se coloca en los registros de seguimiento contable DDF en el campo QMDASQLI, que está mapeado por DSNDQMDA DSECT. Si la *cadena de contabilidad* tiene menos de 255 caracteres, debe rellenarla a la derecha con ceros hasta una longitud de 255 bytes. Los 256 bytes completos se asignan mediante DSNDQMDA DSECT.

Este parámetro es opcional. Si especifica esta *cadena de contabilidad*, también debe especificar *retcode*, *reascode*, *user*, *appl* y *xid*. Si no especifica este parámetro, no se asociará ninguna cadena de contabilidad a la conexión.

También puede cambiar el valor de la cadena de contabilidad con las funciones RRSAF AUTH SIGNON, CONTEXT SIGNON o SET_CLIENT_ID.

Puede recuperar la parte del sufijo DDF de la cadena de contabilidad con el registro especial CURRENT CLIENT_ACCTNG. La parte del sufijo de *la cadena de contabilidad* puede contener un máximo de 200 caracteres. El campo QMDASFLN contiene la longitud del sufijo de contabilidad y el campo QMDASUFX contiene el valor del sufijo de contabilidad. Si se establece la cadena de contabilidad DDF, no se puede consultar el token de contabilidad con el registro especial CURRENT CLIENT_ACCTNG.

Los siguientes parámetros son opcionales y posicionales. Estos parámetros anulan los valores especificados anteriormente en la lista de parámetros. Para proporcionar un valor para una longitud, par de valores, debe proporcionar un valor o especificar una longitud 0 para los parámetros anteriores en la lista de parámetros.

longitud del usuario, nombre largo del usuario

Un par de parámetros que consta de una longitud entera de 2 bytes y un área de cadena de 128 bytes. Los parámetros se separan mediante una coma. Puede proporcionar el ID de usuario del usuario cliente con fines contables y de supervisión en *user-longname*. Db2 muestra este ID de usuario en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el ID de usuario se establece el valor del registro especial CURRENT CLIENT_USERID. Los espacios finales en *user-longname* se truncan y se actualiza la longitud en *user-length*.

Estos parámetros son opcionales; para especificarlos, también debe especificar un valor para *accounting-string*. Un valor de 0 en *la longitud del usuario* omite el procesamiento del *nombre largo del usuario*.

Importante: Estos parámetros anulan cualquier valor que se proporcione en *user*.

longitud-aplicación, nombre-largo-aplicación

Un par de parámetros que consta de una longitud entera de 2 bytes y un área de cadena de 255 bytes. Los parámetros se separan mediante una coma. Puede proporcionar el nombre de la aplicación o transacción del usuario cliente con fines contables y de supervisión en *appl-longname*. Db2 muestra este nombre de aplicación en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el nombre de la aplicación se establece el valor del registro especial CURRENT CLIENT_APPLNAME. Los espacios en blanco al final de *appl-longname* se truncan y se actualiza la longitud en *appl-length*.

Estos parámetros son opcionales, para especificarlos también debe especificar un valor para *user-length*, *user-longname*. Un valor de 0 en *appl-length* omite el procesamiento de *appl-longname*.

Importante: Estos parámetros anulan cualquier valor que se proporcione en *appl*.

longitud de ws, nombre largo de ws

Un par de parámetros que consta de una longitud entera de 2 bytes y un área de cadena de 255 bytes. Los parámetros se separan mediante una coma. Puede proporcionar el nombre de la estación de trabajo del usuario cliente con fines contables y de supervisión en *ws-longname*. Db2 muestra el nombre de esta estación de trabajo en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el nombre de la estación de trabajo se establece el valor del registro especial CURRENT CLIENT_WRKSTNNNAME. Los espacios en blanco al final de *ws-longname* se truncan y se actualiza la longitud en *ws-length*.

Estos parámetros son opcionales, para especificarlos también debe especificar un valor para *appl-length*, *appl-longname*. Un valor de 0 en *ws-length* omite el procesamiento de *ws-longname*.

Importante: Estos parámetros anulan cualquier valor que se proporcione en *ws*.

longitud de correlación, nombre largo de correlación

Un par de parámetros que consta de una longitud entera de 2 bytes y un área de cadena de 255 bytes. Los parámetros se separan mediante una coma. Puede proporcionar un valor único para correlacionar los nombres de sus procesos de negocio con los hilos de correlación (Db2) en *correlation-longname*. Db2 muestra este token de correlación en el resultado del comando DISPLAY THREAD DETAIL. El registro especial CURRENT CLIENT_CORR_TOKEN contiene el token de correlación de cliente. Los espacios en blanco finales en *correlación-longname* se truncan y se actualiza la longitud en *correlación-longitud*.

Estos parámetros son opcionales, para especificarlos también debe especificar un valor para *ws-length*, *ws-longname*. Un valor de 0 en la *longitud de correlación* omite el procesamiento de la *longitud de correlación*.

También puede cambiar el valor del token de correlación de cliente con la función RRSAF AUTH SIGNON y la función SET_CLIENT_ID.

Ejemplo de llamadas de inicio de sesión con AUTH RRSAF

La siguiente tabla muestra una llamada AUTH SIGNON en cada idioma.

Tabla 25. Ejemplos de llamadas de RRSAF AUTH SIGNON

Idioma	Ejemplo de llamada
Assembler	CALL DSNRLI,(ASGNONFN,CORRID,ACCTTKN,ACCTINT,PAUTHID,ACEEPR, SAUTHID,RETCODE,REASCODE,USERID,APPLNAME,WSNAME,XIDPTR)
C1	fnret=dsnrl(&asgnonfn[0], &corrid[0], &acctkn[0], &acctint[0], &pauthid[0], &aceepr, &sauthid[0], &retcode, &reascode, &userid[0], &applname[0], &wsname[0], &xidptr);
COBOL	CALL 'DSNRLI' USING ASGNONFN CORRID ACCTTKN ACCTINT PAUTHID ACEEPR SAUTHID RETCODE REASCODE USERID APPLNAME WSNAME XIDPTR.
Fortran	CALL DSNRLI(ASGNONFN,CORRID,ACCTTKN,ACCTINT,PAUTHID,ACEEPR, SAUTHID,RETCODE,REASCODE,USERID, APPLNAME,WSNAME,XIDPTR)
PL/I1	CALL DSNRLI(ASGNONFN,CORRID,ACCTTKN,ACCTINT,PAUTHID,ACEEPR, SAUTHID,RETCODE,REASCODE,USERID, APPLNAME,WSNAME,XIDPTR);

Nota:

1. Para aplicaciones C, C++ y PL/I, debe incluir las directivas de compilación adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

El siguiente ejemplo muestra una llamada AUTH SIGNON en C¹ con todos los parámetros pasados. Los parámetros que son números se pasan como enteros y las cadenas como matrices de caracteres. En este ejemplo, si *&useridlen* es mayor que 0, entonces el valor del registro especial CURRENT CLIENT_USERID es el valor que se almacena en *&luserid[0]*.

```
fnret = dsnrl(&authsgnfn[0],&corrid[0],&acctkn[0],&acctint[0],&pauthid[0],  
&aceepr,&sauthid[0],&retcode,&reascode,&userid[0],&applname[0],  
&wsname[0],&xidptr,&lacctngid[0],&useridlen,&luserid[0],&applidlen,  
&lapplid[0],&wsidlen,&lwsid[0],&corrtkidlen,&lcorrtkid[0]);
```

Nota:

1. Para aplicaciones C, debe incluir las directivas de compilador adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

Tareas relacionadas

[Invocación del recurso de conexión de servicios de recuperación de recursos](#)

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoque RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

Referencia relacionada

[Función SIGNON para RRSAF](#)

La función SIGNON de RRSAF establece un ID de autorización primario y, opcionalmente, uno o más ID de autorización secundarios para una conexión.

CONTEXTO Función SIGNON para RRSAF

La función RRSAF CONTEXT SIGNON establece un ID de autorización principal y uno o más ID de autorización secundarios para una conexión.

Requisito: Antes de invocar CONTEXT SIGNON, debe haber llamado a la función de servicios de contexto RRS Set Context Data (CTXSDTA) para almacenar un ID de autorización principal y, opcionalmente, la dirección de un ACEE en los datos de contexto cuya clave de contexto suministra como entrada a CONTEXT SIGNON.

La función CONTEXT SIGNON utiliza la clave de contexto para recuperar el ID de autorización principal a partir de los datos asociados al contexto RRS actual. Db2 utiliza la función de servicios de contexto RRS Recuperar datos de contexto (CTXRDTA) para recuperar datos de contexto que contienen el ID de autorización y la dirección ACEE. Los datos de contexto deben tener el siguiente formato:

Número de versión

Un área de 4 bytes que contiene el número de versión de los datos de contexto. Establezca esta área en 1.

Nombre del producto del servidor

Un área de 8 bytes que contiene el nombre del producto servidor que establece los datos de contexto.

ALET

Un área de 4 bytes que puede contener un valor ALET. Db2 no hace referencia a esta área.

Dirección de ACEE

Un área de 4 bytes que contiene una dirección ACEE o 0 si no se proporciona una ACEE. Db2 requiere que el ACEE esté en el espacio de dirección de la tarea.

Si pasa una dirección ACEE, la función CONTEXT SIGNON utiliza el valor en ACEEGRPN como ID de autorización secundaria si la longitud del nombre de grupo (ACEEGRPL) no es 0.

primary-authid

Un área de 8 bytes que contiene el ID de autorización principal que se utilizará. Si el ID de autorización tiene menos de 8 bytes de longitud, rellénelo a la derecha con caracteres en blanco hasta alcanzar los 8 bytes.

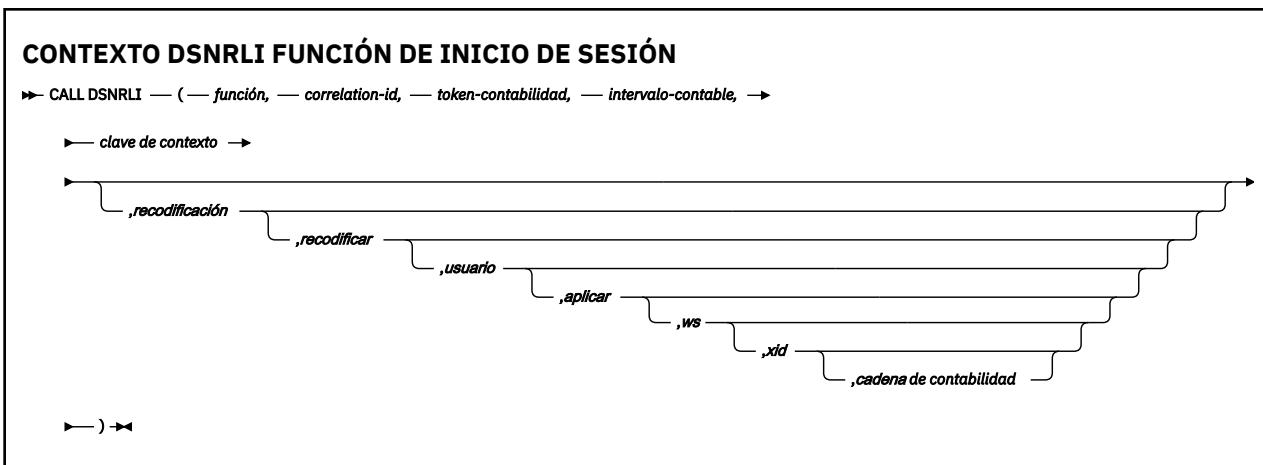
Si el nuevo ID de autorización principal no es diferente del ID de autorización principal actual (que se estableció cuando se invocó la función IDENTIFY o en una invocación anterior de SIGNON), Db2 invoca solo la salida de inicio de sesión. Si el valor ha cambiado, Db2 establece un nuevo ID de autorización principal y un nuevo ID de autorización SQL y, a continuación, invoca la salida de inicio de sesión.

Por lo general, se emite una llamada CONTEXT SIGNON después de una llamada IDENTIFY y antes de una llamada CREATE THREAD. También puede emitir una llamada CONTEXT SIGNON si la aplicación se encuentra en un punto de consistencia y se cumple una de las siguientes condiciones:

- El valor de *reutilización* en la llamada CREATE THREAD era RESET.

- El valor de *reutilización* en la llamada CREATE THREAD era INITIAL, no hay cursos abiertos, el paquete o plan está vinculado con KEEPDYNAMIC(NO), y todos los registros especiales están en su estado inicial. Si existen cursos abiertos o el paquete o plan está vinculado con KEEPDYNAMIC (SÍ), se permite una llamada SIGNON solo si el ID de autorización principal no ha cambiado.

El siguiente diagrama muestra la sintaxis de la función CONTEXT SIGNON.



Los parámetros apuntan a las siguientes áreas:

función

Un área de 18 bytes que contiene CONTEXT SIGNON seguido de cuatro espacios en blanco.

correlation-id

Un área de 12 bytes en la que puede poner un ID de correlación e Db2 . El ID de correlación se muestra en los registros de seguimiento de contabilidad y estadísticas de Db2 . Puede utilizar el ID de correlación para correlacionar unidades de trabajo. Este símbolo aparece en el resultado del comando DISPLAY THREAD. Si no desea especificar un ID de correlación, rellene el área de 12 bytes con espacios en blanco.

contabilidad-token

Un área de 22 bytes en la que puede poner un valor para un token de contabilidad de e Db2 . Este valor se muestra en los registros de seguimiento de contabilidad y estadísticas de Db2 , en el campo QWHCTOKN, que está mapeado por DSNDQWHC DSECT. Al establecer el valor del token contable, se establece el valor del registro especial CURRENT CLIENT_ACCTNG. Si el *token de contabilidad* tiene menos de 22 caracteres, debe llenarlo a la derecha con espacios en blanco hasta alcanzar los 22 caracteres. Si no desea especificar un token de contabilidad, rellene el área de 22 bytes con espacios en blanco.

También puede cambiar el valor del token de contabilidad de la aplicación web (Db2) con las funciones RRSAF SIGNON, AUTH SIGNON o SET_CLIENT_ID. Puede recuperar el token de contabilidad de la cuenta de cliente (Db2) con el registro especial CURRENT CLIENT_ACCTNG solo si la cadena de contabilidad DDF no está configurada.

intervalo-contable

Un área de 6 bytes que especifica cuándo escribe un registro de contabilidad Db2 .

Si especifica COMMIT en esa área, Db2 escribe un registro de contabilidad cada vez que la aplicación emite SRRCMIT. Este registro contable se escribe al final de la segunda fase de un compromiso de dos fases. Si el intervalo de contabilización es COMMIT y se emite un SRRCMIT mientras hay un cursor retenido abierto, el intervalo de contabilización abarca ese commit y termina en el siguiente punto final de intervalo de contabilización válido (como el siguiente SRRCMIT que se emite sin cursos retenidos abiertos, finalización de la aplicación o SIGNON con un nuevo ID de autorización).

Si especifica cualquier otro valor, Db2 escribe un registro contable cuando la aplicación finaliza o cuando llama a la función SIGNON con un nuevo ID de autorización.

clave de contexto

Un área de 32 bytes en la que se coloca la clave de contexto especificada al llamar al servicio RRS Set Context Data (CTXSDTA) para guardar el ID de autorización principal y una dirección ACEE opcional.

retcode

Un área de 4 bytes en la que RRSAF coloca el código de retorno.

Este parámetro es opcional. Si no especifica *el código de retorno*, RRSAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF coloca el código de motivo.

Este parámetro es opcional. Si no especifica *el código de motivo*, RRSAF coloca el código de motivo en el registro 0.

Si especifica *reascode*, también debe especificar *retcode*.

USER

Un área de 16 bytes que contiene el ID de usuario del usuario cliente. Puede utilizar este parámetro para proporcionar la identidad del usuario cliente con fines contables y de supervisión. Db2 muestra este ID de usuario en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el ID de usuario se establece el valor del registro especial CURRENT CLIENT_USERID. Si *el usuario* tiene menos de 16 caracteres, debe llenar los espacios en blanco a la derecha hasta alcanzar los 16 caracteres.

Este parámetro es opcional. Si especifica *user*, también debe especificar *retcode* y *reascode*. Si no especifica *usuario*, no se asociará ningún ID de usuario a la conexión.

aplicación

Un área de 32 bytes que contiene el nombre de la aplicación o transacción de la aplicación del usuario. Puede utilizar este parámetro para proporcionar la identidad del usuario cliente con fines contables y de supervisión. Db2 muestra el nombre de la aplicación en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el nombre de la aplicación se establece el valor del registro especial CURRENT CLIENT_APPLNAME. Si *la solicitud* tiene menos de 32 caracteres, debe llenarla a la derecha con espacios en blanco hasta alcanzar los 32 caracteres.

Este parámetro es opcional. Si especifica *appl*, también debe especificar *retcode*, *reascode* y *user*. Si no especifica *appl*, no se asociará ninguna aplicación o transacción con la conexión.

ws

Un área de 18 bytes que contiene el nombre de la estación de trabajo del usuario cliente. Puede utilizar este parámetro para proporcionar la identidad del usuario cliente con fines contables y de supervisión. Db2 muestra el nombre de la estación de trabajo en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Al establecer el nombre de la estación de trabajo se establece el valor del registro especial CURRENT CLIENT_WRKSTNNNAME. Si *ws* tiene menos de 18 caracteres, debe llenarlo a la derecha con espacios en blanco hasta alcanzar los 18 caracteres.

Este parámetro es opcional. Si especifica *ws*, también debe especificar *retcode*, *reascode*, *user* y *appl*. Si no especifica *ws*, no se asociará ningún nombre de estación de trabajo a la conexión.

También puede cambiar el valor del nombre de la estación de trabajo con las funciones RRSAF SIGNON, AUTH SIGNON o SET_CLIENT_ID. Puede recuperar el nombre de la estación de trabajo con el registro especial CLIENT_WRKSTNNNAME.

xid

Un área de 4 bytes que indica si el hilo forma parte de una transacción global. Un hilo de comunicación (Db2) que forma parte de una transacción global puede compartir bloqueos con otros hilos de comunicación (Db2) que forman parte de la misma transacción global y pueden acceder a los mismos datos y modificarlos. Una transacción global existe hasta que uno de los hilos que forma parte de la transacción global se confirma o se revierte.

Puede especificar uno de los siguientes valores para *xid* :

0

Indica que el hilo no forma parte de una transacción global. El valor 0 debe especificarse como un entero binario.

1

Indica que el hilo es parte de una transacción global y que Db2 debe recuperar el ID de transacción global de RRS. Si ya existe un ID de transacción global para la tarea, el hilo pasa a formar parte de la transacción global asociada. De lo contrario, RRS genera un nuevo ID de transacción global. El valor 1 debe especificarse como un entero binario. De forma alternativa, si desea que Db2 devuelva el ID de transacción global generado al llamante, especifique una dirección en lugar de 1.

dirección

La dirección de 4 bytes de un área en la que se introduce un ID de transacción global para el hilo. Si el ID de transacción global ya existe, el hilo pasa a formar parte de la transacción global asociada. De lo contrario, RRS crea una nueva transacción global con el ID que usted especifique.

Alternativamente, si desea que Db2 genere y devuelva un ID de transacción global, pase la dirección de un ID de transacción global nulo configurando *el campo de formato ID* del ID de transacción global en binario -1 ('FFFFFFF'X). Db2 y luego reemplaza el contenido del área con el ID de transacción generado. El área en la dirección especificada debe estar en almacenamiento grabable y tener una longitud de al menos 140 bytes para acomodar el mayor valor posible de ID de transacción.

El formato de un ID de transacción global se muestra en la descripción de la función RRSAF SIGNON.

cadena de contabilidad

Un campo de un byte de longitud y un área de 255 bytes en el que puede poner un valor para una cadena de contabilidad de Db2 . Este valor se coloca en los registros de seguimiento contable DDF en el campo QMDASQLI, que está mapeado por DSNDQMDA DSECT. Si *la cadena de contabilidad* tiene menos de 255 caracteres, debe llenarla a la derecha con ceros hasta una longitud de 255 bytes. Los 256 bytes completos se asignan mediante DSNDQMDA DSECT.

Este parámetro es opcional. Si especifica esta *cadena de contabilidad*, también debe especificar *retcode*, *reascode*, *user*, *appl* y *xid*. Si no especifica este parámetro, no se asociará ninguna cadena de contabilidad a la conexión.

También puede cambiar el valor de la cadena de contabilidad con las funciones RRSAF AUTH SIGNON, CONTEXT SIGNON o SET_CLIENT_ID.

Puede recuperar la parte del sufijo DDF de la cadena de contabilidad con el registro especial CURRENT CLIENT_ACCTNG. La parte del sufijo de *la cadena de contabilidad* puede contener un máximo de 200 caracteres. El campo QMDASFLN contiene la longitud del sufijo de contabilidad y el campo QMDASUFX contiene el valor del sufijo de contabilidad. Si se establece la cadena de contabilidad DDF, no se puede consultar el token de contabilidad con el registro especial CURRENT CLIENT_ACCTNG.

Ejemplo de llamadas de inicio de sesión de RRSAF CONTEXT

La siguiente tabla muestra una llamada de CONTEXT SIGNON en cada idioma.

Tabla 26. Ejemplos de llamadas de inicio de sesión de RRSAF CONTEXT

Idioma	Ejemplo de llamada
Assembler	CALL DSNRLI,(CSGNONFN,CORRID,ACCTTKN,ACCTINT,CTXKEY, RETCODE,REASCODE,USERID,APPLNAME,WSNAME,XIDPTR)
C1	fnret=dsnrlri(&csgnonfn[0], &corrid[0], &accttkn[0], &acctint[0], &ctxkey[0], &retcode, &reascode, &userid[0], &aplname[0], &wsname[0], &xidptr);

Tabla 26. Ejemplos de llamadas de inicio de sesión de RRSAF CONTEXT (continuación)

Idioma	Ejemplo de llamada
COBOL	<pre>CALL 'DSNRLI' USING CSGNONFN CORRID ACCTTKN ACCTINT CTXTKEY RETCODE REASCODE USERID APPLNAME WSNAME XIDPTR.</pre>
Fortran	<pre>CALL DSNRLI(CSGNONFN,CORRID,ACCTTKN,ACCTINT,CTXKEY, RETCODE,REASCODE, USERID,APPLNAME, WSNAME,XIDPTR)</pre>
PL/I1	<pre>CALL DSNRLI(CSGNONFN,CORRID,ACCTTKN,ACCTINT,CTXKEY, RETCODE,REASCODE,USERID,APPLNAME, WSNAME,XIDPTR);</pre>

Nota:

1. Para aplicaciones C, C++ y PL/I, debe incluir las directivas de compilación adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

Tareas relacionadas

Invocación del recurso de conexión de servicios de recuperación de recursos

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoca RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

Referencia relacionada

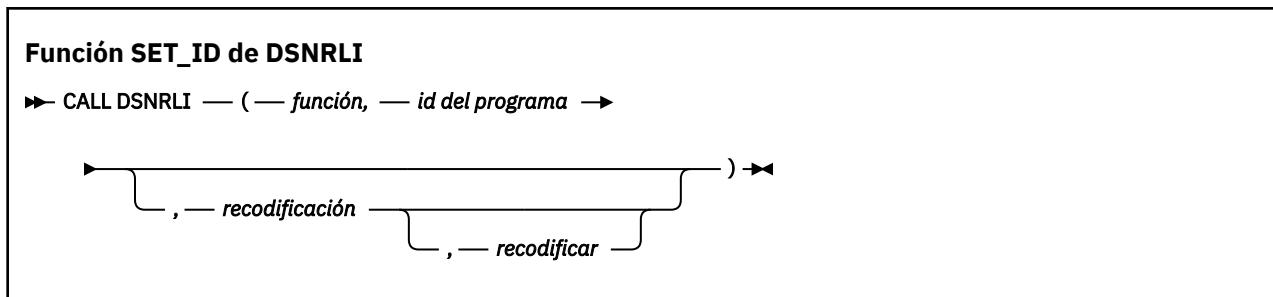
Función SIGNON para RRSAF

La función SIGNON de RRSAF establece un ID de autorización primario y, opcionalmente, uno o más ID de autorización secundarios para una conexión.

Función SET_ID para RRSAF

La función RRSAF SET_ID establece un nuevo valor para el ID del programa cliente que puede utilizarse para identificar al usuario. La función pasa esta información a Db2 cuando se procesa la siguiente solicitud SQL.

El siguiente diagrama muestra la sintaxis de la función SET_ID.



función

Un área de 18 bytes que contiene SET_ID seguido de 12 espacios en blanco.

id-programa

Un área de 80 bytes que contiene la cadena proporcionada por el autor de la llamada que se pasará a Db2. Si el *identificador del programa* tiene menos de 80 caracteres, debe rellenarlo con espacios en blanco a la derecha hasta alcanzar una longitud de 80 caracteres.

Db2 coloca el contenido de *program-id* en los registros IFCID 316, junto con otras estadísticas, para que pueda identificar qué programa está asociado con una instrucción SQL concreta.

retcode

Un área de 4 bytes en la que RRSAF coloca el código de retorno.

Este parámetro es opcional. Si no especifica *el código de devolución*, RRSAF coloca el código de devolución en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF coloca el código de motivo.

Este parámetro es opcional. Si no especifica *el código de motivo*, RRSAF coloca el código de motivo en el registro 0.

Si especifica *reascode*, también debe especificar *retcode*.

Ejemplo de llamadas RRSAF SET_ID

La siguiente tabla muestra una llamada SET_ID en cada idioma.

Tabla 27. Ejemplos de llamadas RRSAF SET_ID

Idioma	Ejemplo de llamada
Assembler	CALL DSNRLI,(SETIDFN,PROGID,RETCode,REASCODE)
C1	fnret=dsnrl(&setidfn[0], &progid[0], &retcode, &reascode);
COBOL	CALL 'DSNRLI' USING SETIDFN PROGID RETCODE REASCODE.
Fortran	CALL DSNRLI(SETIDFN,PROGID,RETCode,REASCODE)
PL/I1	CALL DSNRLI(SETIDFN,PROGID,RETCode,REASCODE);

Nota:

1. Para aplicaciones C, C++ y PL/I, debe incluir las directivas de compilación adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

Tareas relacionadas

Invocación del recurso de conexión de servicios de recuperación de recursos

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoca RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

Función SET_CLIENT_ID para RRSAF

La función RRSAF SET_CLIENT_ID establece valores nuevos para el ID de usuario del cliente, el nombre del programa de aplicación, el nombre de la estación de trabajo, el token contable, la serie de contabilidad del cliente DDF, la señal de correlación y el nombre largo. La función pasa esta información a Db2 cuando se procesa la siguiente solicitud SQL.

Estos valores pueden utilizarse para identificar al usuario final. El programa de llamada define el contenido de estos parámetros. Db2 coloca los valores de los parámetros en la salida del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 .

El siguiente diagrama muestra la sintaxis de la función SET_CLIENT_ID.

Función SET_CLIENT_ID de DSNRLI

```
► CALL DSNRLI — ( — función — , t , usuario , →  
          ↘oken-contabilidad 0 ↗ , 0 ↗  
          ↗ aplicación ↗ , ws ↗ , ►  
          ↗ 0 ↗ , ↗ 0 ↗  
          ↗ código de retorno ↗ , c ↗ , cadena-contable ↗ , token corr ↗ , nombre largo ↗  
          ↗ 0 ↗ , ↗ 0 ↗ , ↗ 0 ↗ , ↗ 0 ↗ , ↗ 0 ↗  
          ► ) ►
```

Los parámetros apuntan a las siguientes áreas:

función

Un área de 18 bytes que contiene SET_CLIENT_ID seguido de 5 espacios en blanco.

contabilidad-token

Un área de 22 bytes en la que puede poner un valor para un token de contabilidad de e Db2 .

Este valor se coloca en los registros de seguimiento de contabilidad y estadísticas de la base de datos (Db2) en el campo QWHCTOKN, que está mapeado por DSNDQWHD DSECT. Si el *token de contabilidad* tiene menos de 22 caracteres, debe rellenarlo a la derecha con espacios en blanco hasta alcanzar los 22 caracteres.

Puede omitir este parámetro especificando un valor de 0 en la lista de parámetros.

Alternativamente, puede cambiar el valor del token de contabilidad de la solicitud de registro (Db2) con las funciones SIGNON, AUTH SIGNON o CONTEXT SIGNON de RRSASF. Puede recuperar el token de contabilidad de la cuenta de cliente (Db2) con el registro especial CURRENT CLIENT_ACCTNG solo si la cadena de contabilidad DDF no está configurada.

USER

Un área de 16 bytes o 128 bytes que contiene el ID de usuario del usuario final del cliente. Puede utilizar este parámetro para proporcionar la identidad del usuario final del cliente con fines contables y de supervisión. Db2 coloca este ID de usuario en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Si el *usuario* tiene menos de 16 caracteres, debe llenar los espacios en blanco a la derecha hasta alcanzar los 16 caracteres.

Puede omitir este parámetro especificando un valor de 0 en la lista de parámetros.

Si se especifica el parámetro *de nombre largo*, la longitud máxima del *parámetro de usuario* es de 128 bytes. Si el *usuario* tiene menos de 128 caracteres, debe llenar los espacios en blanco a la derecha hasta alcanzar los 128 caracteres.

También puede cambiar el valor del ID de usuario del cliente con las funciones RRSASF SIGNON, AUTH SIGNON o CONTEXT SIGNON. Puede recuperar el ID de usuario del cliente con el registro especial CLIENT_USERID.

aplicación

Un área de 32 bytes o 255 bytes que contiene el nombre de la aplicación o transacción de la aplicación del usuario final. Puede utilizar este parámetro para proporcionar la identidad del usuario final del cliente con fines contables y de supervisión. Db2 coloca el nombre de la aplicación en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Si el *nombre* tiene menos de 32 caracteres, debe llenarlo a la derecha con espacios en blanco hasta alcanzar los 32 caracteres.

Puede omitir este parámetro especificando un valor de 0 en la lista de parámetros.

Si se especifica el parámetro *de nombre largo*, la longitud máxima *del parámetro appl* es de 255 bytes. Si la *solicitud* tiene menos de 255 caracteres, debe llenarla a la derecha con espacios en blanco hasta alcanzar los 255 caracteres.

También puede cambiar el valor del nombre de la aplicación con las funciones RRSAF SIGNON, AUTH SIGNON o CONTEXT SIGNON. Puede recuperar el nombre de la aplicación con el registro especial CLIENT_APPLNAME.

ws

Un área de 18 bytes o 255 bytes que contiene el nombre de la estación de trabajo del usuario final cliente. Puede utilizar este parámetro para proporcionar la identidad del usuario final del cliente con fines contables y de supervisión. Db2 coloca este nombre de estación de trabajo en el resultado del comando DISPLAY THREAD y en los registros de seguimiento de contabilidad y estadísticas de Db2 . Si ws tiene menos de 18 caracteres, debe rellenarlo a la derecha con espacios en blanco hasta alcanzar una longitud de 18 caracteres.

Puede omitir este parámetro especificando un valor de 0 en la lista de parámetros.

Si se especifica el parámetro *de nombre largo*, la longitud máxima *del parámetro ws* es de 255 bytes. Si *el mensaje* tiene menos de 255 caracteres, debe rellenarlo a la derecha con espacios en blanco hasta alcanzar los 255 caracteres.

También puede cambiar el valor del nombre de la estación de trabajo con las funciones RRSAF SIGNON, AUTH SIGNON o CONTEXT SIGNON. Puede recuperar el nombre de la estación de trabajo con el registro especial CLIENT_WRKSTNNNAME.

retcode

Un área de 4 bytes en la que RRSAF coloca el código de retorno.

Puede omitir este parámetro especificando un valor de 0 en la lista de parámetros.

Este parámetro es opcional. Si no especifica *el código de retorno*, RRSAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF coloca el código de motivo.

Puede omitir este parámetro especificando un valor de 0 en la lista de parámetros.

Este parámetro es opcional. Si no especifica *el código de reasignación*, RRSAF coloca el código de motivo en el registro 0.

Si especifica *reascode*, también debe especificar *retcode*.

cadena de contabilidad

Un campo de un byte de longitud y un área de 255 bytes en los que puede poner un valor para una cadena de contabilidad de Db2 . Este valor se coloca en los registros de seguimiento contable DDF en el campo QMDASUFX, que está mapeado por DSNDQMDA DSECT. Si *la cadena de contabilidad* tiene menos de 255 caracteres, debe rellenarla a la derecha con ceros hasta una longitud de 255 bytes. Los 256 bytes completos se asignan mediante DSNDQMDA DSECT.

Puede omitir este parámetro especificando un valor de 0 en la lista de parámetros.

Este parámetro es opcional. Si especifica esta *cadena de contabilidad*, también debe especificar *retcode*, *reascode*, *user* y *appl*. Si no especifica este parámetro, no se asociará ninguna cadena de contabilidad a la conexión.

También puede cambiar el valor de la cadena de contabilidad con las funciones RRSAF AUTH SIGNON, CONTEXT SIGNON o SET_CLIENT_ID.

Puede recuperar la parte del sufijo DDF de la cadena de contabilidad con el registro especial CURRENT CLIENT_ACCTNG. La parte del sufijo de *la cadena de contabilidad* puede contener un máximo de 200 caracteres. El campo QMDASFLN contiene la longitud del sufijo de contabilidad y el campo QMDASUFX contiene el valor del sufijo de contabilidad. Si se establece la cadena de contabilidad DDF, no se puede consultar el token de contabilidad con el registro especial CURRENT CLIENT_ACCTNG.

corr-token

Un área de 255 bytes donde se especifica un token de correlación de cliente. Puede especificar un valor único para correlacionar su proceso de negocio dentro de Db2 y toda su empresa. El valor de

corr-token se muestra mediante el comando DISPLAY THREAD DETAIL. El registro especial CURRENT CLIENT_CORR_TOKEN contiene el token de correlación de cliente. Si el *token de corrección* tiene menos de 255 caracteres, debe rellenarlo a la derecha con espacios en blanco hasta alcanzar una longitud de 255 bytes.

Puede omitir este parámetro especificando un valor de 0 en la lista de parámetros. Si especifica *corr-token*, también debe especificar *long-name*.

También puede cambiar el valor del token de correlación del cliente con la función RRSAF SIGNON.

nombre largo

Un área de 8 bytes que contiene el valor LONGNAME.

Este parámetro opcional se utiliza para indicar a la función RRSAF que los parámetros de entrada *user*, *appl*, *ws*, *accounting-string* y *corr-token* pueden aceptar longitudes mayores. No puede asociar selectivamente el parámetro *nombre largo* con ningún parámetro individual.

Ejemplo de llamadas RRSAF SET_CLIENT_ID

La siguiente tabla muestra una llamada SET_CLIENT_ID en cada idioma.

Tabla 28. Ejemplos de llamadas RRSAF SET_CLIENT_ID

Idioma	Ejemplo de llamada
Assembler	CALL DSNRLI,(SECLIDFN,ACCT,USER,APPL,WS,RETCODE,REASCODE, ACCOUNTINGSTRING,CORRTOKEN,LONGNAME)
C1	fnret=dsnrl1(&seclidfn[0], &acct[0], &user[0], &appl[0], &ws[0], &retcode, &reascode, &accountingstring[0], &corrtoken[0], &longname[0]);
COBOL	CALL 'DSNRLI' USING SECLIDFN ACCT USER APPL WS RETCODE REASCODE ACCOUNTING-STRING CORR-TOKEN LONG-NAME.
Fortran	CALL DSNRLI(SECLIDFN,ACCT,USER,APPL,WS,RETCODE,REASCODE, ACCOUNTINGSTRING,CORRTOKEN,LONGNAME)
PL/I1	CALL DSNRLI(SECLIDFN,ACCT,USER,APPL,WS,RETCODE,REASCODE, ACCOUNTINGSTRING,CORRTOKEN,LONGNAME);

Nota:

1. Para aplicaciones C, C++ y PL/I, debe incluir las directivas de compilación adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

Tareas relacionadas

Invocación del recurso de conexión de servicios de recuperación de recursos

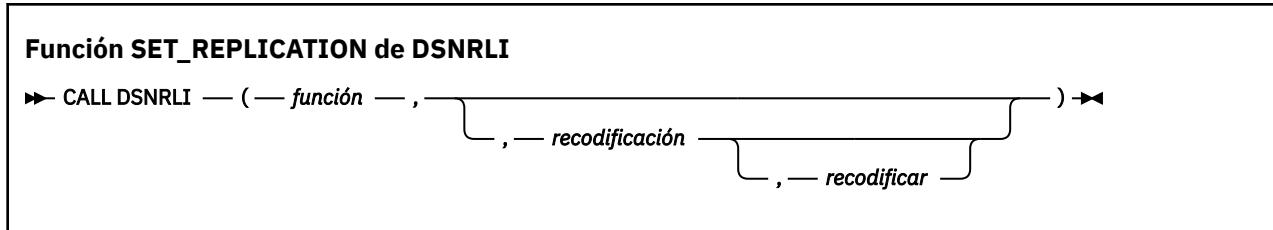
El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoca RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

Función SET_REPLICATION para RRSAF

La función RRSAF SET_REPLICATION permite a un programa autorizado APF identificar Db2 como un programa de réplica.

La llamada a la función SET_REPLICATION es opcional. Si no lo llama, Db2 tratará la solicitud con normalidad. La función SET_REPLICATION permite a la aplicación realizar operaciones de inserción, actualización y eliminación, y luego el espacio de tabla o la base de datos se inicia el acceso RREPL.

El siguiente diagrama muestra la sintaxis de la función SET REPLICATION.



Los parámetros apuntan a las siguientes áreas:

función

Un área de 18 bytes que contiene SET_REPLICATION.

retcode

Un área de 4 bytes en la que RRSAF coloca el código de retorno.

Este parámetro es opcional. Si no especifica *el código de retorno*, RRSAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF coloca un código de motivo.

Este parámetro es opcional. Si no especifica *el código de reasignación*, RRSAF coloca el código de motivo en el registro 0.

Si especifica *reascode*, también debe especificar *retcode*.

Tareas relacionadas

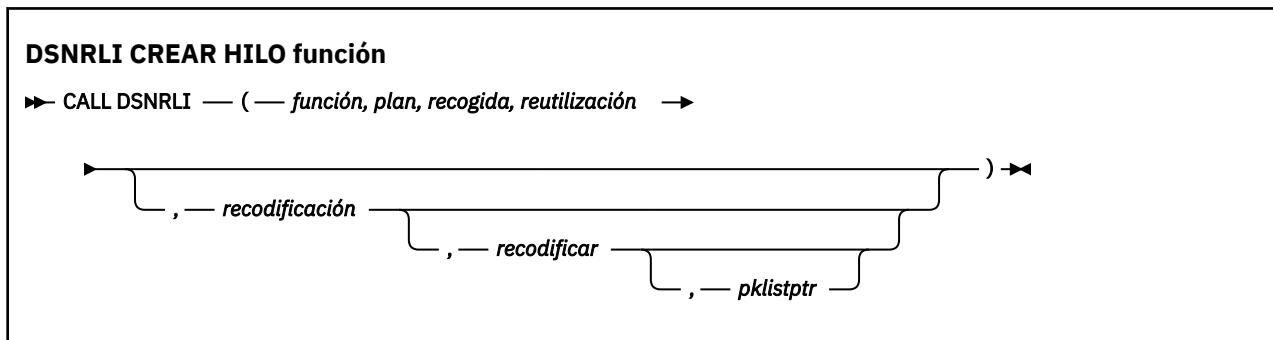
[Invocación del recurso de conexión de servicios de recuperación de recursos](#)

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoque RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

Función CREATE THREAD para RRSAF

La función RRSAF CREATE THREAD asigna los recursos de E/S (Db2) necesarios para que una aplicación emita solicitudes SQL o IFI. Esta función debe completarse antes de que la aplicación pueda ejecutar sentencias SQL o solicitudes IFI.

El siguiente diagrama muestra la sintaxis de la función CREATE THREAD.



Los parámetros apuntan a las siguientes áreas:

función

Un área de 18 bytes que contiene CREATE THREAD seguido de cinco espacios en blanco.

plan

Un nombre de plan de 8 bytes (Db2). RRSAF asigna el plan designado.

Si proporciona un nombre de colección en lugar de un nombre de plan, especifique el carácter de interrogación (?) en el primer byte de este campo. Db2 y luego asigna un plan especial llamado? RRSAF y utiliza el valor que usted especifique para *el cobro*. Cuando Db2 asigna un plan llamado? RRSAF, Db2 comprueba la autorización para ejecutar el paquete de la misma manera que comprueba la autorización para ejecutar un paquete de un solicitante que no sea Db2 for z/OS.

Si no proporciona un nombre de colección en el campo *de colección*, debe introducir un nombre de plan válido en este campo.

colección

Un área de 18 bytes en la que se introduce un nombre de colección. Db2 utiliza los nombres de la colección para localizar un paquete asociado a la primera instrucción SQL del programa.

Cuando proporcione un nombre de colección y ponga el carácter de interrogación (?) en *el campo del plan*, Db2 asigna un plan llamado?RRSAF y una lista de paquetes que contenga las dos entradas siguientes:

- El nombre de la colección especificado.
- Una entrada que contenga * para la ubicación, el nombre de la colección y el nombre del paquete. (Esta entrada permite a la aplicación acceder a ubicaciones remotas y a paquetes de colecciones distintas de la colección predeterminada especificada en el momento de crear el hilo)

La aplicación puede utilizar la instrucción SET CURRENT PACKAGESET para cambiar el ID de colección que utiliza Db2 para localizar un paquete.

Si proporciona un nombre de plan en el campo *del plan*, Db2 ignora el valor del *campo de recopilación*.

reutilizar

Un área de 8 bytes que controla la acción que realiza Db2 si se emite una llamada SIGNON después de una llamada CREATE THREAD. Especifique uno de los siguientes valores en este campo:

RESET

Liberá cualquier cursor retenido y reinicializa los registros especiales

INITIAL

No permite la llamada de INICIO DE SESIÓN

Este parámetro es necesario. Si el área de 8 bytes no contiene ni RESET ni INITIAL, el valor predeterminado es INITIAL.

retcode

Un área de 4 bytes en la que RRSAF coloca el código de retorno.

Este parámetro es opcional. Si no especifica *el código de retorno*, RRSAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF coloca el código de motivo.

Este parámetro es opcional. Si no especifica *el código de motivo*, RRSAF coloca el código de motivo en el registro 0.

Si especifica *reascode*, también debe especificar *retcode*.

pklistptr

Un campo de 4 bytes que contiene un puntero a un área de datos proporcionada por el usuario que contiene una lista de ID de colección. Un ID de colección es un identificador SQL de 1 a 128 letras, dígitos o el carácter de subrayado que identifica una colección de paquetes. La longitud del área de datos es de un máximo de 2050 bytes. El área de datos contiene un campo de longitud de 2 bytes, seguido de hasta 2048 bytes de entradas de ID de colección, separadas por comas.

Cuando especifique *pklistptr* y el carácter de interrogación (?) en el campo del plan, Db2 asigna un plan especial llamado?RRSAF y una lista de paquetes que contenga las siguientes entradas:

- Los nombres de colección que especifique en el área de datos a la que apunta *pklistptr*
- Una entrada que contenga * para la ubicación, el ID de la colección y el nombre del paquete

Si también especifica *la recogida*, Db2 ignora ese valor.

Cada entrada de colección debe tener el formato *collection-ID. *, *.collection-ID. *, o **. iD de colección* y debe seguir las convenciones de nomenclatura para un ID de colección, como se describe en la descripción de las opciones ENLAZAR y VUELVA A ENLAZAR.

Db2 utiliza los nombres de la colección para localizar un paquete asociado a la primera instrucción SQL del programa. La entrada que contiene *** permite que la aplicación acceda a ubicaciones remotas y a paquetes de acceso en colecciones distintas de la colección predeterminada que se especifica en el momento de crear el hilo.

La aplicación puede utilizar la instrucción SET CURRENT PACKAGESET para cambiar el ID de colección que utiliza Db2 para localizar un paquete.

Este parámetro es opcional. Si especifica este parámetro, también debe especificar *retcode* y *reascode*.

Si proporciona un nombre de plan en el campo *del plan*, Db2 ignora *el valor pklistptr*.

Recomendación: El uso de una lista de paquetes puede tener un impacto negativo en el rendimiento. Para un mejor rendimiento, especifique una lista de paquetes corta.

Ejemplo de llamadas RRSAF CREATE THREAD

La siguiente tabla muestra una llamada CREATE THREAD en cada idioma.

Tabla 29. Ejemplos de llamadas RRSAF CREATE THREAD

Idioma	Ejemplo de llamada
Assembler	CALL DSNRLI,(CRTHRDFN,PLAN,COLLID,REUSE,RETCODE,REASCODE,PKLISTPTR)
C1	fnret=dsnrl(&crthrdfn[0], &plan[0], &collid[0], &reuse[0], &retcode, &reascode, &pklistptr);
COBOL	CALL 'DSNRLI' USING CRTHRDFN PLAN COLLID REUSE RETCODE REASCODE PKLSTPTR.
Fortran	CALL DSNRLI(CRTHRDFN,PLAN,COLLID,REUSE,RETCODE,REASCODE,PKLSTPTR)
PL/I1	CALL DSNRLI(CRTHRDFN,PLAN,COLLID,REUSE,RETCODE,REASCODE,PKLSTPTR);

Nota:

1. Para aplicaciones C, C++ y PL/I, debe incluir las directivas de compilación adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

Tareas relacionadas

[Invocación del recurso de conexión de servicios de recuperación de recursos](#)

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoca RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

[Autorización del acceso a planes o paquetes mediante aplicaciones \(Managing Security\)](#)

Referencia relacionada

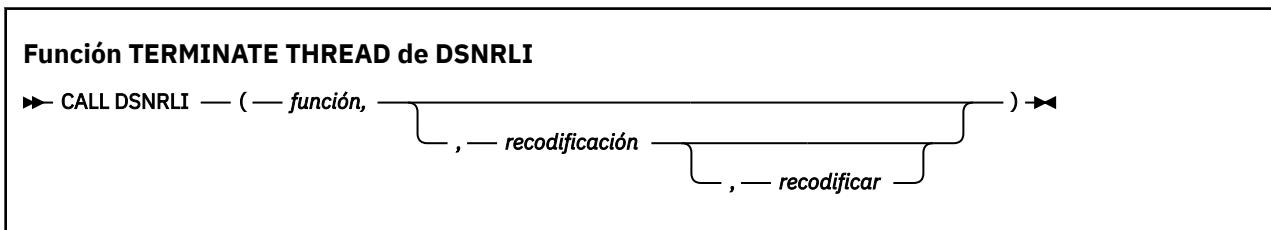
Opciones BIND y REBIND para paquetes, planes y servicios (comandos Db2)

Función TERMINATE THREAD para RRSAF

La función RRSAF TERMINATE THREAD desasigna recursos de memoria compartida (Db2) que están asociados con un plan y que fueron asignados previamente para una aplicación por la función CREATE THREAD. A continuación, puede utilizar la función CREAR HILO para asignar otro plan con la misma conexión.

Si llama a la función TERMINATE THREAD y la aplicación no está en un punto de consistencia, RRSAF devuelve el código de razón X'00C12211'.

El siguiente diagrama muestra la sintaxis de la función TERMINATE THREAD.



Los parámetros apuntan a las siguientes áreas:

función

Un área de 18 bytes que contiene TERMINATE THREAD seguido de dos espacios en blanco.

retcode

Un área de 4 bytes en la que RRSAF coloca el código de retorno.

Este parámetro es opcional. Si no especifica *el código de retorno*, RRSAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF coloca el código de motivo.

Este parámetro es opcional. Si no especifica *el código de motivo*, RRSAF lo coloca en el registro 0.

Si especifica *reascode*, también debe especificar *retcode*.

Ejemplo de llamadas RRSAF TERMINATE THREAD

La siguiente tabla muestra una llamada TERMINATE THREAD en cada idioma.

Tabla 30. Ejemplos de llamadas RRSAF TERMINATE THREAD

Idioma	Ejemplo de llamada
Assembler	CALL DSNRLI,(TRMTHDFN,RETCODE,REASCODE)
C ¹	fnret=dsnrl(&trmthdfn[0], &retcode, &reascode);
COBOL	CALL 'DSNRLI' USING TRMTHDFN RETCODE REASCODE.
Fortran	CALL DSNRLI(TRMTHDFN,RETCODE,REASCODE)
PL/I1	CALL DSNRLI(TRMTHDFN,RETCODE,REASCODE);

Nota:

1. Para aplicaciones C, C++ y PL/I, debe incluir las directivas de compilador adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

Tareas relacionadas

Invocación del recurso de conexión de servicios de recuperación de recursos

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoque RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

TERMINAR función IDENTIFICAR para RRSAF

La función RRSAF TERMINATE IDENTIFY finaliza una conexión con Db2. La llamada a la función TERMINATE IDENTIFY es opcional. Si no lo llama, Db2 realiza las mismas funciones cuando finaliza la tarea.

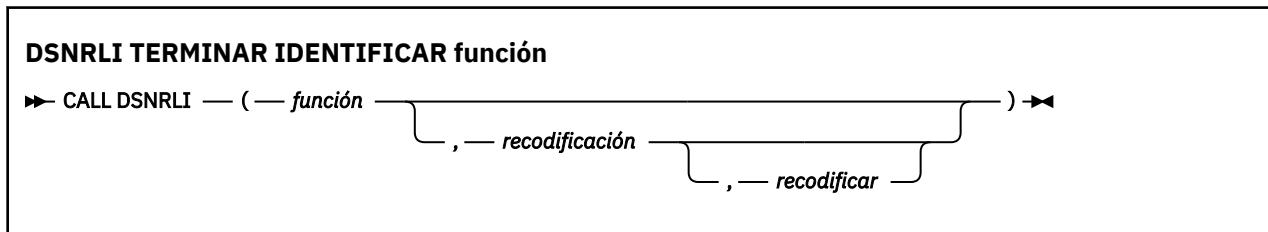
Si se produce una terminación de la aplicación (Db2), esta debe emitir TERMINATE IDENTIFY para restablecer los bloques de control RRSAF. Esta acción garantiza que las futuras solicitudes de conexión de la tarea se realicen correctamente cuando se reinicie Db2.

La función TERMINATE IDENTIFY elimina la conexión de la tarea de llamada a Db2. Si ninguna otra tarea en el espacio de direcciones tiene una conexión activa con Db2, Db2 también elimina las estructuras de bloques de control que se crearon para el espacio de direcciones y elimina la autorización de memoria cruzada.

Si la aplicación no está en un punto de consistencia cuando se llama a la función TERMINATE IDENTIFY, RRSAF devuelve el código de razón X'00C12211'.

Si la aplicación asignó un plan y usted llama a la función TERMINATE IDENTIFY sin llamar primero a la función TERMINATE THREAD, Db2 desasigna el plan antes de terminar la conexión.

El siguiente diagrama muestra la sintaxis de la función TERMINATE IDENTIFY.



Los parámetros apuntan a las siguientes áreas:

función

Un área de 18 bytes que contiene TERMINATE IDENTIFY.

retcode

Un área de 4 bytes en la que RRSAF coloca el código de retorno.

Este parámetro es opcional. Si no especifica *el código de retorno*, RRSAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF coloca el código de motivo.

Este parámetro es opcional. Si no especifica *el código de razón*, RRSAF coloca el código de razón en el registro 0.

Si especifica reascode, también debe especificar retcode.

Ejemplo de llamadas RRSAF TERMINATE IDENTIFY

La siguiente tabla muestra una llamada TERMINATE IDENTIFY en cada idioma.

Tabla 31. Ejemplos de llamadas RRSAF TERMINATE IDENTIFY

Idioma	Ejemplo de llamada
Assembler	CALL DSNRLI,(TMIDFYFN, RETCODE, REASCODE)
C ¹	fnret=dsnrl(&tmidfyfn[0], &retcode, &reascode);
COBOL	CALL 'DSNRLI' USING TMIDFYFN RETCODE REASCODE.
Fortran	CALL DSNRLI(TMIDFYFN, RETCODE, REASCODE)
PL/I1	CALL DSNRLI(TMIDFYFN, RETCODE, REASCODE);

Nota:

1. Para aplicaciones C, C++ y PL/I, debe incluir las directivas de compilación adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

Tareas relacionadas

[Invocación del recurso de conexión de servicios de recuperación de recursos](#)

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoque RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

Función TRANSLATE para RRSAF

La función TRANSLATE de RRSAF convierte un código de razón hexadecimal para un error de Db2 en un código SQL entero firmado y un mensaje de error imprimible. El código SQL y el texto del mensaje se colocan en las variables de host SQLCODE y SQLSTATE o en campos relacionados de SQLCA.

Tenga en cuenta las siguientes reglas y recomendaciones sobre cuándo utilizar y no utilizar la función TRADUCIR:

- No puede llamar a la función TRANSLATE desde el idioma e Fortran.
- Llame a la función TRANSLATE solo después de una operación IDENTIFY exitosa. Para los errores que se producen durante las solicitudes SQL o IFI, la función TRANSLATE se ejecuta automáticamente.
- La función TRANSLATE traduce códigos que empiezan por X'00F3', pero no traduce códigos de motivo RRSAF que empiezan por X'00C1'.

Si recibe el código de motivo de error X'00F30040' (recurso no disponible) después de una solicitud OPEN, la función TRANSLATE devuelve el nombre del objeto de base de datos no disponible en los últimos 44 caracteres del campo SQLERRM.

Si la función TRANSLATE no reconoce el código de motivo de error, devuelve SQLCODE -924 (SQLSTATE '58006') y coloca una copia imprimible del código de función original de la función Db2 y los códigos de motivo de error y de retorno en el campo SQLERRM. El contenido de los registros 0 y 15 no cambia, a menos que falle TRANSLATE. En este caso, el registro 0 se establece en X'00C12204' y el registro 15 se establece en 200.

El siguiente diagrama muestra la sintaxis de la función TRANSLATE.

Función DSNRLI TRANSLATE

```
► CALL DSNRLI — ( — función, sqlca — , — recodificación — , — recodificar — ) ►
```

Los parámetros apuntan a las siguientes áreas:

función

Un área de 18 bytes que contiene la palabra TRANSLATE seguida de nueve espacios en blanco.

sqlca

El área de comunicación SQL (SQLCA) del programa.

retcode

Un área de 4 bytes en la que RRSAF coloca el código de retorno.

Este parámetro es opcional. Si no especifica *el código de retorno*, RRSAF coloca el código de retorno en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF coloca el código de motivo.

Este parámetro es opcional. Si no especifica *el código de motivo*, RRSAF coloca el código de motivo en el registro 0.

Si especifica *reascode*, también debe especificar *retcode*.

Ejemplo de llamadas RRSAF TRANSLATE

La siguiente tabla muestra una llamada de TRANSLATE en cada idioma.

Tabla 32. Ejemplos de llamadas RRSAF TRANSLATE

Idioma	Ejemplo de llamada
Assembler	CALL DSNRLI,(XLATFN,SQLCA,RETCODE,REASCODE)
C1	fnret=dsnrlili(&connfn[0], &sqlca, &retcode, &reascode);
COBOL	CALL 'DSNRLI' USING XLATFN SQLCA RETCODE REASCODE.
PL/I1	CALL DSNRLI(XLATFN,SQLCA,RETCODE,REASCODE);

Nota:

1. Para aplicaciones C, C++ y PL/I, debe incluir las directivas de compilación adecuadas, ya que DSNRLI es un programa en lenguaje ensamblador. Estas directivas del compilador se describen en las instrucciones para invocar RRSAF.

Tareas relacionadas

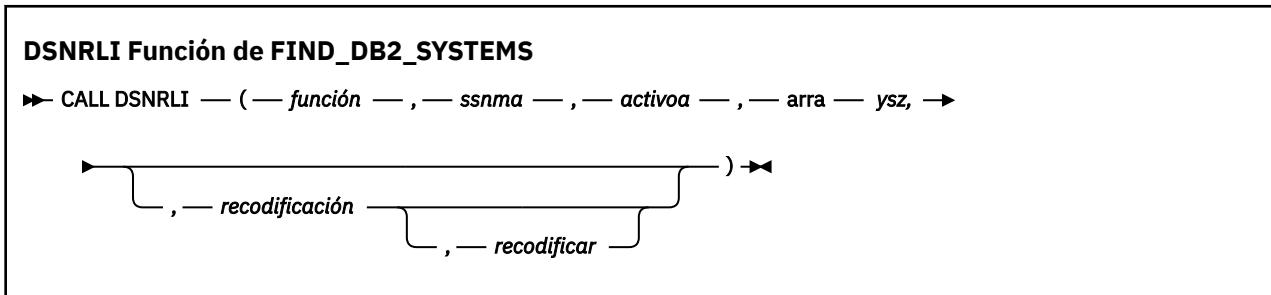
Invocación del recurso de conexión de servicios de recuperación de recursos

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoque RRSAF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSAF tiene más funciones que CAF.

FIND_DB2_SYSTEMS función para RRSAF

La función RRSAF (FIND_DB2_SYSTEMS) identifica todos los subsistemas de Db2 s activos en un z/OS LPAR.

El siguiente diagrama muestra la sintaxis de la función " FIND_DB2_SYSTEMS ".



Los parámetros apuntan a las siguientes áreas:

función

Un área de 18 bytes que contiene " FIND_DB2_SYSTEMS " seguido de dos espacios en blanco.

ssnma

Un área de almacenamiento para una matriz de cadenas de caracteres de 4 bytes en la que RRSAF coloca los nombres de todos los subsistemas de radio (Db2 , SSID) que están definidos para el LPAR actual. Debe proporcionar el área de almacenamiento. Si la matriz es mayor que el número de subsistemas de Db2 , RRSAF devuelve el valor " ' ' " (cuatro espacios en blanco) en todos los miembros de la matriz no utilizados.

activoa

Un área de almacenamiento para una matriz de valores de 4 bytes en la que RRSAF devuelve una indicación de si un subsistema definido está activo. Cada valor se representa como un entero fijo de 31 bits. El valor 1 significa que el subsistema está activo. El valor 0 significa que el subsistema no está activo. El tamaño de esta matriz debe ser el mismo que el de la matriz *ssnma*. Si la matriz es mayor que el número de subsistemas de Db2 , RRSAF devuelve el valor -1 en todos los miembros de la matriz no utilizados.

La información en la matriz *activoa* es la información que está disponible en el momento en que la solicitó y podría cambiar en cualquier momento.

arraysz

Un área de 4 bytes, representada como un entero fijo de 31 bits, que especifica el número de entradas para las matrices *ssnma* y *activea*. Si el número de entradas de matriz es insuficiente para contener todos los subsistemas definidos en el LPAR actual, RRSAF utiliza todas las entradas disponibles y devuelve el código de retorno 4.

retcode

Un área de 4 bytes en la que RRSAF debe colocar el código de retorno para esta llamada a la función " FIND_DB2_SYSTEMS ".

Este parámetro es opcional. Si no vuelve a codificar, RRSAF coloca el código de devolución en el registro 15 y el código de motivo en el registro 0.

reascode

Un área de 4 bytes en la que RRSAF debe colocar el código de motivo de esta llamada a la función " FIND_DB2_SYSTEMS ".

Este parámetro es opcional. Si no especifica el código de razón, RRSAF coloca el código de razón en el registro 0.

Ejemplos de valores que devuelve la función " FIND_DB2_SYSTEMS "

Suponga que se definen dos subsistemas en la LPAR actual. El subsistema DB2A está activo y el subsistema DB2B está detenido. Supongamos que invoca RRSASF con la función FIND_DB2_SYSTEMS y un valor de 3 para arraysz. Las matrices ssnma y activea se establecen con los siguientes valores:

Tabla 33. Valores de ejemplo devueltos en las matrices ssnma y activea

Número de elemento de matriz	Valores en matriz ssnma	Valores en matriz activa
1	DB2A	1
2	DB2B	0
3	(cuatro espacios en blanco)	-1

Tareas relacionadas

[Invocación del recurso de conexión de servicios de recuperación de recursos](#)

El recurso de conexión de servicios de recuperación de recursos (RRSAF) habilita el programa para que se comunique con Db2. Invoca RRSASF como alternativa a invocar CAF o al usar procedimientos almacenados que se ejecutan en un espacio de direcciones establecido por WLM. RRSASF tiene más funciones que CAF.

Códigos de retorno y códigos de razón de RRSASF

Si se especifican parámetros de código de retorno y de código de razón en una llamada de función de recurso de conexión de servicios de recuperación de recursos (RRSAF), RRSASF devuelve el código de retorno y el código de razón en esos parámetros. Si no se especifican estos parámetros ni se invoca RRSASF de forma implícita, RRSASF coloca el código de retorno en el registro 15 y el código de razón en el registro 0.

Cuando el código de motivo comience por X'00F3 ', excepto X'00F30006 ', puede utilizar la función RRSASF TRANSLATE para obtener el texto del mensaje de error que se puede imprimir y mostrar.

Para las llamadas SQL, RRSASF devuelve códigos de retorno SQL estándar en el SQLCA. RRSASF devuelve los códigos de devolución IFI y los códigos de motivo en el área de comunicación de la instalación de instrumentación (IFCA).

La siguiente tabla enumera los códigos de devolución de RRSASF.

Tabla 34. Códigos de devolución RRSASF

Código de retorno	Explicación
0	La llamada se ha realizado satisfactoriamente.
4	La información de estado está disponible. Consulte el código de motivo para obtener más detalles.
>4	La llamada ha fallado. Consulte el código de motivo para obtener más detalles.

Referencia relacionada

[Función TRANSLATE para RRSASF](#)

La función TRANSLATE de RRSASF convierte un código de razón hexadecimal para un error de Db2 en un código SQL entero firmado y un mensaje de error imprimible. El código SQL y el texto del mensaje se colocan en las variables de host SQLCODE y SQLSTATE o en campos relacionados de SQLCA.

Ejemplos de escenarios RRSASF

Una o más tareas pueden utilizar el servicio de conexión de los Servicios de recuperación de recursos (RRSAF) para conectarse a Db2. Esta conexión puede establecerse de forma implícita o explícita. Para conexiones explícitas, una tarea llama a una o más de las funciones de conexión RRSASF.

Una sola tarea

El siguiente pseudocódigo de ejemplo ilustra una única tarea que se ejecuta en un espacio de direcciones que se conecta explícitamente a Db2 a través de RRSAF. z/OS Los controles RRS confirman el procesamiento cuando la tarea finaliza con normalidad.

```
IDENTIFY  
SIGNON  
CREATE THREAD  
SQL or IFI  
:  
TERMINATE IDENTIFY
```

Múltiples tareas

En el siguiente escenario, varias tareas en un espacio de direcciones se conectan explícitamente a Db2 a través de RRSAF. La tarea 1 no ejecuta instrucciones SQL ni realiza llamadas IFI. Su propósito es supervisar los ECB de terminación y arranque de e Db2 , y comprobar el nivel de liberación de e Db2 .

TASK 1	TASK 2	TASK 3	TASK n
IDENTIFY	IDENTIFY	IDENTIFY	IDENTIFY
SIGNON	SIGNON	SIGNON	SIGNON
CREATE THREAD	CREATE THREAD	CREATE THREAD	CREATE THREAD
SQL	SQL	SQL	SQL
...
SRRCMIT	SRRCMIT	SRRCMIT	SRRCMIT
SQL	SQL	SQL	SQL
...
SRRCMIT	SRRCMIT	SRRCMIT	SRRCMIT
...
TERMINATE IDENTIFY

Reutilización de un hilo e Db2

El siguiente pseudocódigo de ejemplo muestra un hilo de conversación (Db2) que es reutilizado por otro usuario en un punto de consistencia. Cuando la aplicación llama a la función SIGNON para el usuario B, Db2 reutiliza el plan asignado por la función CREATE THREAD para el usuario A.

```
IDENTIFY  
SIGNON user A  
CREATE THREAD  
SQL  
:  
SRRCMIT  
SIGNON user B  
SQL  
:  
SRRCMIT
```

Cambiar hilos de conversación de Db2 s entre tareas

El siguiente escenario muestra cómo puede cambiar los hilos para cuatro usuarios (A, B, C y D) entre dos tareas (1 y 2).

Task 1	Task 2
CTXBEGC (create context a) CTXSWCH(a,0)	CTXBEGC (create context b) CTXSWCH(b,0)
IDENTIFY	IDENTIFY
SIGNON user A	SIGNON user B
CREATE THREAD (Plan A)	CREATE THREAD (plan B)
SQL	SQL
...	...
CTXSWCH(0,a)	CTXSWCH(0,b)
CTXBEGC (create context c) CTXSWCH(c,0)	CTXBEGC (create context d) CTXSWCH(d,0)

```

IDENTIFY
SIGNON user C
CREATE THREAD (plan C)
SQL
...
CTXSWCH(b,c)
SQL (plan B)
...
IDENTIFY
SIGNON user D
CREATE THREAD (plan D)
SQL
...
CTXSWCH(0,d)
...
CTXSWCH(a,0)
SQL (plan A)

```

Las aplicaciones realizan los siguientes pasos:

- La tarea 1 crea el contexto a, cambia de contexto para que el contexto a esté activo para la tarea 1 y llama a la función IDENTIFY para inicializar una conexión con un subsistema. Una tarea siempre debe llamar a la función IDENTIFY antes de que pueda producirse un cambio de contexto. Una vez completada la operación IDENTIFY, la tarea 1 asigna un hilo al usuario A y realiza operaciones SQL.

Al mismo tiempo, la tarea 2 crea el contexto b, cambia de contexto para que el contexto b esté activo para la tarea 2, llama a la función IDENTIFY para inicializar una conexión con el subsistema, asigna un hilo al usuario B y realiza operaciones SQL.

Cuando se completan las operaciones SQL, ambas tareas realizan operaciones de cambio de contexto RRS. Estas operaciones desconectan cada hilo de ejecución (Db2) de la tarea bajo la cual se estaba ejecutando.

- La tarea 1 crea entonces el contexto c, llama a la función IDENTIFY para inicializar una conexión con el subsistema, cambia de contexto para que el contexto c esté activo para la tarea 1, asigna un hilo para el usuario C y realiza operaciones SQL para el usuario C.

La tarea 2 realiza las mismas operaciones para el usuario D.

- Cuando las operaciones SQL para el usuario C se completan, la tarea 1 realiza una operación de cambio de contexto para llevar a cabo las siguientes acciones:

- Cambie el hilo del usuario C fuera de la tarea 1.
- Cambie el hilo del usuario B a la tarea 1.

Para que una operación de cambio de contexto asocie una tarea con un hilo de ejecución (Db2), el hilo de ejecución (Db2) debe haber realizado previamente una operación de IDENTIFICACIÓN (IDENTIFY). Por lo tanto, antes de que el hilo del usuario B pueda asociarse a la tarea 1, la tarea 1 debe haber realizado una operación IDENTIFICAR.

- La tarea 2 realiza dos operaciones de cambio de contexto para llevar a cabo las siguientes acciones:
 - Disociar el hilo del usuario D de la tarea 2.
 - Asociar el hilo para el usuario A con la tarea 2.

Ejemplos de programas para RRSAF

El recurso de conexión de servicios de recuperación de recursos (RRSAF) permite a los programas comunicarse con Db2. Puede utilizar RRSAF como alternativa a CAF.

Ejemplo de JCL para invocar RRSAF

El siguiente ejemplo de JCL muestra cómo utilizar RRSAF en un entorno por lotes. La instrucción DSNRRSAF DD inicia el seguimiento RRSAF. Utilice ese informe de diagnóstico solo si está diagnosticando un problema.

```

//jobname      JOB      z/OS_jobcard_information
//RRSJCL       EXEC    PGM=RRS_application_program
//STEPLIB      DD      DSN=application_load_library
//                  DD      DSN=DB2_load_library
//
//SYSPRINT     DD      SYSOUT=*
//DSNRRSAF     DD      DUMMY
//SYSUDUMP     DD      SYSOUT=*

```

Ejemplo de carga y eliminación de la interfaz de idioma RRSAF

El siguiente segmento de código muestra cómo una aplicación carga los puntos de entrada DSNRLI y DSNHLIR de la interfaz de lenguaje RRSAF. Almacenar los puntos de entrada en las variables LIRLI y LISQL garantiza que la aplicación cargue los puntos de entrada solo una vez. Elimine los módulos cargados cuando la aplicación ya no necesite acceder a Db2.

```
***** GET LANGUAGE INTERFACE ENTRY ADDRESSES
LOAD EP=DSNRLI      Load the RRSAF service request EP
ST  R0,LIRLI        Save this for RRSAF service requests
LOAD EP=DSNHLIR     Load the RRSAF SQL call Entry Point
ST  R0,LISQL        Save this for SQL calls
*
*: Insert connection service requests and SQL calls here
.
DELETE EP=DSNRLI    Correctly maintain use count
DELETE EP=DSNHLIR   Correctly maintain use count
```

Ejemplo de uso del punto de entrada ficticio DSNHLI para RRSAF

Cada una de las instalaciones de conexión de Db2 contiene un punto de entrada llamado DSNHLI. Cuando se utiliza RRSAF pero no se especifica la opción del precompilador ATTACH(RRSAF), el precompilador genera instrucciones BALR a DSNHLI para las sentencias SQL en el programa. Para encontrar el punto de entrada DSNHLI correcto sin incluir DSNRLI en su módulo de carga, codifique una subrutina, con el punto de entrada DSNHLI, que pase el control al punto de entrada DSNHLIR en el módulo DSNRLI. DSNHLIR es exclusivo de DSNRLI y se encuentra en la misma ubicación que DSNHLI en DSNRLI. DSNRLI utiliza direccionamiento de 31 bits. Si la aplicación que llama a esta subrutina intermedia utiliza direccionamiento de 24 bits, la subrutina intermedia debe tener en cuenta la diferencia.

En el siguiente ejemplo, LISQL es direccionable porque la CSECT de llamada utilizó el mismo registro 12 que CSECT DSNHLI. Su aplicación también debe establecer direccionabilidad a LISQL.

```
*****
* Subroutine DSNHLI intercepts calls to LI EP=DSNHLI
*****
DS 0D
DSNHLI CSECT      Begin CSECT
STM R14,R12,12(R13) Prologue
LA  R15,SAVEHLI   Get save area address
ST  R13,4(,R15)   Chain the save areas
ST  R15,8(,R13)   Chain the save areas
LR  R13,R15      Put save area address in R13
L   R15,LISQL    Get the address of real DSNHLI
BASSM R14,R15   Branch to DSNRLI to do an SQL call
*               DSNRLI is in 31-bit mode, so use
*               BASSM to assure that the addressing
*               mode is preserved.
*               Restore R13 (caller's save area addr)
*               Restore R14 (return address)
*               Restore R1-12, NOT R0 and R15 (codes)
L   R13,4(,R13)
L   R14,12(,R13)
RETURN (1,12)
```

Ejemplo de conexión a Db2 con RRSAF

Este ejemplo utiliza las variables que se declaran en el siguiente código.

```
***** VARIABLES SET BY APPLICATION *****
LIRLI  DS F          DSNRLI entry point address
LISQL   DS F          DSNHLIR entry point address
SSNM    DS CL4        DB2 subsystem name for IDENTIFY
CORRID  DS CL12       Correlation ID for SIGNON
ACCTTKN DS CL22      Accounting token for SIGNON
ACCTINT DS CL6       Accounting interval for SIGNON
PLAN    DS CL8        DB2 plan name for CREATE THREAD
COLLID  DS CL18       Collection ID for CREATE THREAD. If
*           PLAN contains a plan name, not used.
REUSE   DS CL8        Controls SIGNON after CREATE THREAD
CONTROL  DS CL8       Action that application takes based
*           on return code from RRSAF
***** VARIABLES SET BY DB2 *****
STARTECB DS F         DB2 startup ECB
```

```

TERMECB DS F DB2 termination ECB
EIBPTR DS F Address of environment info block
RIBPTR DS F Address of release info block
***** CONSTANTS *****
CONTINUE DC CL8'CONTINUE' CONTROL value: Everything OK
IDFYFN DC CL18'IDENTIFY' ' Name of RRSASF service
SGNONFN DC CL18'SIGNON' ' Name of RRSASF service
CRTHRDFN DC CL18'CREATE THREAD' ' Name of RRSASF service
TRMTHDFN DC CL18'TERMINATE THREAD' ' Name of RRSASF service
TMIDFYFN DC CL18'TERMINATE IDENTIFY' ' Name of RRSASF service
***** SQLCA and RIB *****
      EXEC SQL INCLUDE SQLCA
      DSNDRIB          Map the DB2 Release Information Block
***** Parameter list for RRSASF calls *****
RRSAFCLL CALL ,(*,*,*,*,*,*),VL,MF=L

```

El siguiente código de ejemplo muestra cómo emitir solicitudes para las funciones RRSASF IDENTIFY, SIGNON, CREATE THREAD, TERMINATE THREAD y TERMINATE IDENTIFY. Este ejemplo no muestra una tarea que espera en el ECB de terminación de la conexión a Internet (Db2). Puede codificar dicha tarea y utilizar la z/OS Macro WAIT para supervisar el ECB. La tarea que espera en el ECB de terminación debe separar el código de muestra si se publica el ECB de terminación. Esa tarea también puede esperar en el inicio de Db2 ECB. Este ejemplo espera en el ECB de inicio en su propio nivel de tarea.

```

***** IDENTIFY *****
L   R15,LIRLI      Get the Language Interface address
CALL (15),(IDFYFN,SSNM,RIBPTR,EIBPTR,TERMECB,STARTECB),VL,MF=X
(E,RRSAFCLL)
BAL R14,CHEKCODE   Call a routine (not shown) to check
                   return and reason codes
* CLC CONTROL,CONTINUE Is everything still OK
* BNE EXIT          If CONTROL not 'CONTINUE', stop loop
* USING R8,RIB       Prepare to access the RIB
* L   R8,RIBPTR     Access RIB to get DB2 release level
* CLC RIBREL,RIBR999 DB2 V10 or later?
* BE  USERELX       If RIBREL = '999', use RIBRELX
* WRITE 'The current DB2 release level is' RIBREL
* B   SIGNON        Continue with signon USERELX
* WRITE 'The current DB2 release level is' RIBRELX
***** SIGNON *****
SIGNON L   R15,LIRLI      Get the Language Interface address
CALL (15),(SGNONFN,CORRID,ACCTTKN,ACCTINT),VL,MF=(E,RRSAFCLL)
BAL R14,CHEKCODE   Check the return and reason codes
***** CREATE THREAD *****
L   R15,LIRLI      Get the Language Interface address
CALL (15),(CRTHRDFN,PLAN,COLLID,REUSE),VL,MF=(E,RRSAFCLL)
BAL R14,CHEKCODE   Check the return and reason codes
***** SQL *****
* Insert your SQL calls here. The DB2 Precompiler
* generates calls to entry point DSNHLI. You should
* code a dummy entry point of that name to intercept
* all SQL calls. A dummy DSNHLI is shown in the following
* section.
***** TERMINATE THREAD *****
CLC CONTROL,CONTINUE Is everything still OK?
BNE EXIT          If CONTROL not 'CONTINUE', shut down
L   R15,LIRLI      Get the Language Interface address
CALL (15),(TRMTHDFN),VL,MF=(E,RRSAFCLL)
BAL R14,CHEKCODE   Check the return and reason codes
***** TERMINATE IDENTIFY *****
CLC CONTROL,CONTINUE Is everything still OK?
BNE EXIT          If CONTROL not 'CONTINUE', stop loop
L   R15,LIRLI      Get the Language Interface address
CALL (15),(TMIDFYFN),VL,MF=(E,RRSAFCLL)
BAL R14,CHEKCODE   Check the return and reason codes

```

Interfaz de lenguaje universal (DSNULI)

El subcomponente de la interfaz de lenguaje universal (DSNULI) determina el entorno en tiempo de ejecución y carga y se bifurca dinámicamente al modelo de interfaz de lenguaje adecuado.

La siguiente figura muestra la estructura general de DSNULI y un programa que lo utiliza:

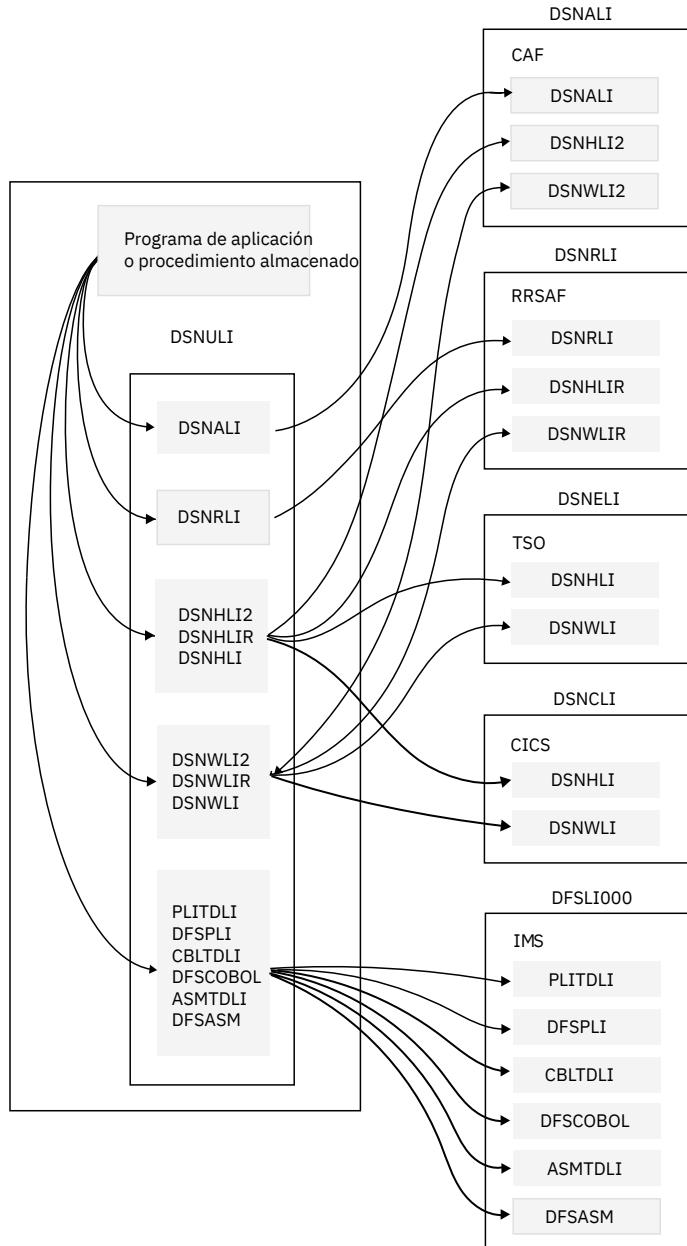


Figura 2. Programa de aplicación o procedimiento almacenado vinculado con DSNULI

El módulo de carga de la interfaz de lenguaje universal (Db2, DSNULI) es el módulo de la interfaz de lenguaje universal. DSNULI no tiene alias.

DSNULI tiene los siguientes puntos de entrada:

DSNALI

Para Db2 explícito Llame a las solicitudes de servicio de conexión de Attach Facility

DSNRLI

Para Db2 explícito Servicios de recuperación de recursos Adjuntar solicitudes de servicio de conexión de instalaciones

DSNCLI

Para editar enlaces con CICS

DSNHLI

Para llamadas SQL genéricas desde aplicaciones diseñadas para ejecutarse en cualquier entorno.

DSNHLI2

Para llamadas SQL explícitas a través de la función Call Attachment Facility

DSNHЛИR

Para llamadas SQL explícitas a través del Servicio de recuperación de recursos adjuntos

DSNWLI

Para llamadas IFI genéricas desde aplicaciones diseñadas para ejecutarse en cualquier entorno.

DSNWLI2

Para llamadas IFI explícitas a través del servicio de adjuntar llamadas.

DSNWLR

Para llamadas IFI explícitas a través del Servicio de recuperación de recursos adjunto

PLITDLI

Para PL/I específico IMS acceso a bases de datos

DFSPLI

Para PL/I específico IMS acceso a bases de datos

CBLTDLI

Para COBOL específico IMS acceso a bases de datos

DFSCOBOL

Para COBOL específico IMS acceso a bases de datos

ASMTDLI

Para ensambladores específicos IMS acceso a bases de datos

DFSASM

Para ensambladores específicos IMS acceso a bases de datos

DSNULI carga dinámicamente y se ramifica al módulo de interfaz de idioma apropiado, que se basa en el nombre del punto de entrada (para puntos de entrada específicos de adjuntos), o en el entorno actual (para los puntos de entrada genéricos DSNHLI y DSNWLI).

Tareas relacionadasEdición de enlaces de una aplicación con DSNULI

Para crear un único módulo de carga que se pueda utilizar en más de un entorno de conexión, puede editar el enlace de su programa o procedimiento almacenado con el módulo de interfaz de lenguaje universal (DSNULI) en lugar de con uno de los módulos de interfaz de lenguaje específicos del entorno (DSNELI, DSNALI, DSNRLI, DSNCLI o DFSLI000).

Edición de enlaces de una aplicación con DSNULI

Para crear un único módulo de carga que se pueda utilizar en más de un entorno de conexión, puede editar el enlace de su programa o procedimiento almacenado con el módulo de interfaz de lenguaje universal (DSNULI) en lugar de con uno de los módulos de interfaz de lenguaje específicos del entorno (DSNELI, DSNALI, DSNRLI, DSNCLI o DFSLI000).

Acerca de esta tarea

DSNULI debe editarse mediante vínculos con las aplicaciones TSO, CAF, RRSAF (incluidos los procedimientos almacenados), CICS aplicaciones y aplicaciones de IMS. DSNULI determina el entorno de ejecución, luego carga dinámicamente y se ramifica al módulo de interfaz de lenguaje apropiado (DSNELI, DSNALI, DSNRLI, DSNCLI o DFSLI000).

Se aplican las siguientes consideraciones:

- Si el requisito principal es el máximo rendimiento, enlace-edite con DSNELI, DSNALI, DSNRLI, DSNCLI o DFSLI000 en lugar de DSNULI. Si el requisito principal es mantener una única copia de un módulo de carga, enlace-edición con DSNULI.
- Si se requiere la funcionalidad de conexión implícita CAF, edite el enlace de su aplicación con DSNALI en lugar de con DSNULI. DSNULI utiliza de forma predeterminada conexiones implícitas RRSAF si no se ha establecido un entorno de adjuntos al entrar en DSNHLI. Los entornos de adjuntos se establecen llamando inicialmente a DSNRLI o DSNALI, o ejecutando una aplicación SQL bajo el procesador de comandos TSO o bajo CICS o IMS.

- DSNULI no eliminará explícitamente los DSNELI, DSNALI, DSNRLI o DSNCLI cargados. Si una aplicación no puede tolerar que estos módulos se eliminan solo al finalizar la tarea, utilice DSNELI, DSNALI, DSNRLI o DSNCLI en lugar de DSNULI.
- DSNULI se envía con los atributos de vinculación AMODE(31) y RMODE(ANY) y debe introducirse en AMODE(31).

Procedimiento

Puede incluir DSNULI cuando edite el enlace de su módulo de carga. Por ejemplo, puede utilizar una instrucción de control del editor de enlaces como esta en su JCL:

```
INCLUDE SYSLIB(DSNULI)
```

Resultados

Al codificar esta declaración, se evita la vinculación a uno de los módulos de interfaz de lenguaje específicos del entorno.

Controlar la CICS función de adjuntar desde una aplicación

Utilice la CICS función de adjuntar para acceder a Db2 desde CICS programas de aplicación.

Acerca de esta tarea

Puede iniciar y detener la CICS función de adjuntar desde un programa de aplicación.

Procedimiento

Para controlar la CICS función de adjuntar:

1. Para iniciar la CICS función de adjuntar, realice una de las siguientes acciones:

- Incluya la siguiente declaración en su solicitud:

```
EXEC CICS LINK PROGRAM('DSN2COM0')
```

- Utilice la interfaz de programación del sistema SET DB2CONN para el CICS Servidor de transacciones.

2. Para detener la CICS función de adjuntar, realice una de las siguientes acciones:

- Incluya la siguiente declaración en su solicitud:

```
EXEC CICS LINK PROGRAM('DSN2COM2')
```

- Utilice la interfaz de programación del sistema SET DB2CONN para el CICS Servidor de transacciones.

Información relacionada

[SET DB2CONN \(CICS Transaction Server for z/OS \)](#)

Detectar si el CICS funcionamiento del servicio de adjuntos

Antes de ejecutar instrucciones SQL en un CICS programa, debe determinar si la CICS función de adjunto está disponible. No es necesario que realice esta prueba si el CICS función de conexión está iniciada y está utilizando el modo de espera.

Acerca de esta tarea

Cuando se ejecuta una instrucción SQL y el CICS función de adjuntar está en modo de espera, el adjunto emite SQLCODE -923 con un código de razón que indica que Db2 no está disponible.

Procedimiento

Utilice el comando INQUIRE EXITPROGRAM para el CICS Servidor de transacciones de su aplicación.

El siguiente ejemplo muestra cómo utilizar este comando. En este ejemplo, el comando INQUIRE EXITPROGRAM comprueba si el administrador de recursos para SQL, DSNCSQL, está en funcionamiento. CICS devuelve los resultados en el campo EIBRESP del bloque de interfaz EXEC (EIB) y en el campo cuyo nombre es el argumento del parámetro CONNECTST (en este caso, STST). Si el valor EIBRESP indica que el comando se completó normalmente y el valor STST indica que el administrador de recursos está disponible, entonces puede ejecutar instrucciones SQL.

```
STST      DS      F
ENTNAME   DS      CL8
EXITPROG  DS      CL8
:
MVC      ENTNAME,=CL8'DSNCSQL'
MVC      EXITPROG,=CL8'DSN2EXT1'
EXEC    CICS INQUIRE EXITPROGRAM(EXITPROG)
        ENTRYNAME(ENTNAME) CONNECTST(STST) NOHANDLE
        CLC     EIBRESP,DFHRESP(NORMAL)
        BNE     NOTREADY
        CLC     STST,DFHVALUE(CONNECTED)
        BNE     NOTREADY
UPNREADY DS      OH
        attach is up
NOTREADY DS      OH
        attach is not up yet
```

Si utiliza el comando INQUIRE EXITPROGRAM para evitar que AEY9 falle y el CICS función de adjuntos no funciona, puede producirse el efecto de desagüe de tormentas. *El efecto de drenaje de tormentas* es una condición que se produce cuando un sistema sigue recibiendo trabajo, aunque ese sistema esté inactivo.

Conceptos relacionados

[Efecto "storm-drain" \(Db2 Installation and Migration\)](#)

Información relacionada

[INQUIRE EXITPROGRAM \(CICS Transaction Server for z/OS \)](#)

[-923 \(Db2 Codes\)](#)

Mejorar la reutilización de hilo en CICS aplicaciones

Por lo general, se recomienda que las transacciones reutilicen subprocessos, ya que la creación de cada subprocesso conlleva un alto coste de procesamiento.

Procedimiento

Cierre todos los cursos que se declaran con la opción WITH HOLD antes de cada punto de sincronización.

Db2 no los cierra automáticamente. Un hilo para una aplicación que contiene un cursor abierto no se puede reutilizar. Debe cerrar todos los cursos inmediatamente después de haberlos utilizado.

Conceptos relacionados

[Cursos retenidos y no retenidos](#)

Un cursor retenido no se cierra después de una operación de confirmación. Un cursor no retenido se cierra después de una operación de confirmación. Especifica si desea que un cursor esté retenido o no incluyendo u omitiendo la cláusula WITH HOLD cuando declara el cursor.

Capítulo 3. Db2 Programación de SQL

Puede utilizar el lenguaje SQL para escribir una instrucción que describa lo que quiere hacer con los datos de una base de datos y en qué condiciones quiere hacerlo.

El Lenguaje de Consulta Estructurado (SQL) es un lenguaje estandarizado basado en el modelo relacional de datos que se utiliza para definir y manipular datos en una base de datos relacional. Las sentencias SQL pueden estar contenidas en funciones definidas por el usuario, procedimientos definidos por el usuario o desencadenadores, incrustadas en programas de lenguaje de alto nivel, preparadas y ejecutadas dinámicamente o ejecutadas de forma interactiva.

Para obtener información sobre SQL incrustado, consulte [Capítulo 4, “Programación de SQL incorporado”, en la página 487](#).

Creación y modificación de objetos de interfaz de usuario (Db2) desde programas de aplicación

Su programa de aplicación puede crear y manipular objetos de la base de datos (Db2), como tablas, vistas, desencadenadores, tipos distintos, funciones definidas por el usuario y procedimientos almacenados. Debe tener las autorizaciones adecuadas para crear dichos objetos.

Creación de tablas a partir de programas de aplicación

La creación de una tabla proporciona un lugar más lógico para almacenar datos relacionados en un subsistema Db2.

Procedimiento

Utilice una instrucción CREATE TABLE que incluya los siguientes elementos:

- Nombre de la tabla. Consulte [Directrices para nombres de tabla \(Db2 Administration Guide\)](#).
- Una lista de las columnas que componen la tabla. Separe cada descripción de columna de la siguiente con una coma y encierre entre paréntesis la lista completa de descripciones de columnas.

Para cada columna, especifique la siguiente información:

- El nombre de la columna (por ejemplo, SERIAL). Consulte [Nombres de columna \(Db2 SQL\)](#).
- El atributo de tipo de datos y longitud (por ejemplo, CHAR(8)). Consulte [Tipos de datos de columnas \(Introducción a Db2 para z/OS\)](#).
- Opcionalmente, especifique un valor predeterminado o una restricción sobre el valor. Puede utilizar los valores siguientes:

Palabra clave	Resultado
NO NULL	Especifica que la columna no puede contener valores nulos
UNIQUE	El valor de cada fila debe ser único y la columna no puede contener valores nulos.
DEFAULT	La columna tiene uno de los siguientes valores predeterminados asignados por el sistema de gestión de contenidos (Db2): <ul style="list-style-type: none">- Para las columnas numéricas, 0 (cero) es el valor predeterminado.

Palabra clave	Resultado
	<ul style="list-style-type: none"> - Para cadenas de longitud fija de caracteres o gráficos, el valor predeterminado es blanco. - Para cadenas binarias de longitud fija, un conjunto de ceros hexadecimales es el valor predeterminado. - Para cadenas de longitud variable, incluidas las cadenas LOB, la cadena vacía (una cadena de longitud cero) es el valor predeterminado. - Para las columnas de fecha y hora, el valor actual del registro especial asociado es el valor predeterminado.
Valor POR DEFECTO	<p>El valor predeterminado se especifica como uno de los siguientes valores:</p> <ul style="list-style-type: none"> - Una constante - Nulo - SESSION_USER, que especifica el valor del registro especial SESSION_USER en el momento en que se necesita un valor predeterminado para la columna - CURRENT_SQLID, que especifica el valor del registro especial CURRENT_SQLID en el momento en que se necesita un valor predeterminado para la columna - El nombre de una función de conversión que convierte un valor predeterminado (de un tipo de datos integrado) al tipo distinto de una columna

- Opcionalmente, especifique el método de partición para los datos de la tabla. Db2 utiliza particiones basadas en el tamaño de forma predeterminada si no se especifica cómo particionar los datos al crear la tabla. Para obtener más información, consulte [Particionamiento de datos en tablas de Db2 \(Db2 Administration Guide\)](#).
- Opcionalmente, una restricción referencial o una restricción de verificación. Para obtener más información, consulte “[Restricciones de comprobación](#)” en la página 137 y “[Restricciones referenciales](#)” en la página 139.

Ejemplo

Por ejemplo, la siguiente instrucción SQL crea una tabla llamada PRODUCT:

```
CREATE TABLE PRODUCT
(SERIAL      CHAR(8)      NOT NULL,
DESCRIPTION  VARCHAR(60)   DEFAULT '',
MFGCOST     DECIMAL(8,2),
MFGDEPT    CHAR(3),
MARKUP      SMALLINT,
SALESDEPT  CHAR(3),
CURDATE    DATE         DEFAULT );
```

Conceptos relacionados

[Tablas de Db2 \(Introducción a Db2 para z/OS\)](#)

Tareas relacionadas

[Creación de tablas base \(Db2 Administration Guide\)](#)

Referencia relacionada

CREATE TABLE declaración (Db2 SQL)

Información relacionada

Lección 1.2: Creación de una tabla (Introducción a Db2 para z/OS)

Tipos de datos de columnas

Al crear una tabla de Db2, define cada columna para que tenga un tipo de datos específico. El tipo de datos de una columna determina qué puede y qué no puede hacer con la columna.

Cuando realice operaciones en columnas, los datos deben ser compatibles con el tipo de datos de la columna a la que se hace referencia. Por ejemplo, no puede insertar datos de caracteres, como un apellido, en una columna cuyo tipo de datos sea numérico. Del mismo modo, no puede comparar columnas que contengan tipos de datos incompatibles.

El tipo de datos de una columna puede ser un tipo distinto, que es un tipo de datos definido por el usuario, o un tipo de datos integrado de Oracle (Db2). Como se muestra en la siguiente figura, los tipos de datos integrados de Db2 tienen cuatro categorías generales: fecha y hora, cadena, numérico e identificador de fila (ROWID).

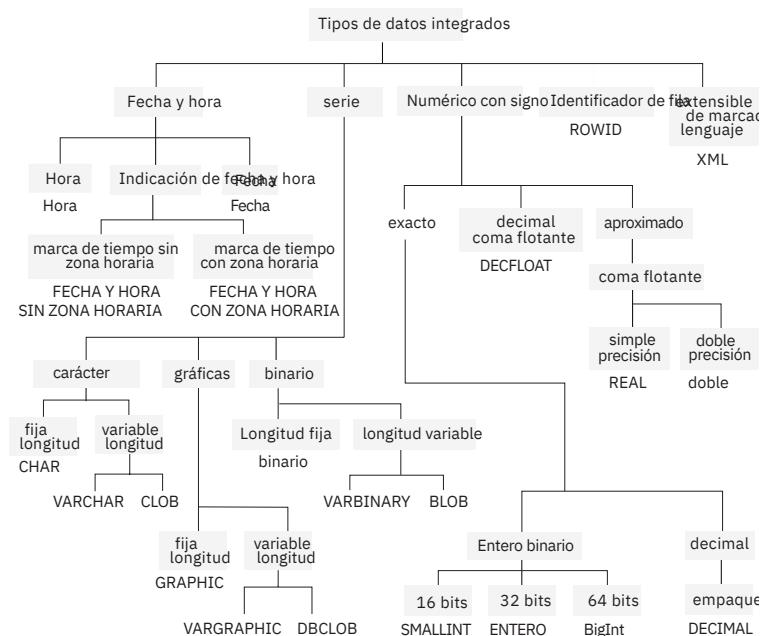


Figura 3. Db2 Tipos de datos incorporados

Resumen de tipos de datos en Db2 for z/OS

Tabla 35. Intervalos o longitudes y recuentos de bytes de columnas por tipo de datos

Tipo de datos	Rango o longitud	Número de bytes
SMALLINT	de -32768 a +32767	2
ENTERO	de -2147483648 a +2147483647	4
BIGINT	-9223372036854775808 a +9223372036854775807	8
FLOTAR (n)	aproximadamente de -7.2E+75 a 7.2E+75	Si n está en el rango de 1 a 21, el recuento de bytes es 4. Si n está en el rango 22-53, el recuento de bytes es 8.

Tabla 35. Intervalos o longitudes y recuentos de bytes de columnas por tipo de datos (continuación)

Tipo de datos	Rango o longitud	Número de bytes
DECIMAL	1 - 10^{31} a $10^{31} - 1$	ENTERO ($p/2$) + 1, donde p es la precisión
DECFLOAT(16)	10^{-383} a 10^{+384}	9
DECFLOAT(34)	10^{-6143} a 10^{+6144}	17
CHAR (n)	el atributo de longitud debe estar en el rango de 1 a 255	n
VARCHAR (n)	el atributo de longitud debe estar en el rango 1-32704	$n+2$
CLOB	el atributo de longitud debe estar en el rango de 1 a 2147483647, ambos inclusive	6
En línea CLOB	el atributo de longitud debe estar en el rango de 1 a 2147483647, ambos inclusive; la longitud en línea debe estar en el rango de 0 a 32680, ambos inclusive	6 + recuento de bytes en línea
GRAPHIC (n)	el atributo de longitud debe estar en el rango de 1 a 127, ambos inclusive	$2n$
VARGRAPHIC (n)	el atributo de longitud debe estar en el rango de 1 a 16352	$2n+2$
DBCLOB	el atributo de longitud debe estar en el rango de 1 a 1 073 741 823, ambos inclusive la longitud en línea debe estar en el rango de 0 a 32680, ambos inclusive	6
En línea DBCLOB	el atributo de longitud debe estar en el rango de 1 a 1 073 741 823, ambos inclusive; la longitud en línea debe estar en el rango de 0 a 16340	6 + (número de caracteres en línea * 2)
BINARIO(n)	el atributo de longitud debe estar en el rango de 1 a 255, ambos inclusive	n
VARBINARIO(n)	el atributo de longitud debe estar en el rango 1-32704	$n+2$
BLOB	el atributo de longitud debe estar en el rango 1-32704	6
BLOB en línea	el atributo de longitud debe estar en el rango 1-32704	6 + recuento de bytes en línea
Fecha	ver el enlace	4
Hora	ver el enlace	3
FECHA Y HORA (p) SIN ZONA HORARIA	ver el enlace	ENTERO(($p+1$)/2) + 7 donde p es la precisión

Tabla 35. Intervalos o longitudes y recuentos de bytes de columnas por tipo de datos (continuación)

Tipo de datos	Rango o longitud	Número de bytes
FECHA Y HORA (<i>p</i>) CON ZONA HORARIA	ver el enlace	ENTERO((<i>p</i> +1)/2) + 9 donde <i>p</i> es la precisión
ROWID	ver el enlace	19
tipo diferenciado	ver el tipo de fuente	La longitud del tipo de datos de origen en el que se basó el tipo distinto
XML	ver el enlace	6 - Si la columna no puede contener varias versiones de un documento XML. 14 - Si la columna puede contener varias versiones de un documento XML. Para obtener más información, consulte Db2 Cómo utiliza XML (Db2 Programming for XML) Tiffany & Co. (Programming for XML).

Conceptos relacionados

[Asignación y comparación \(Db2 SQL\)](#)

[Conversiones entre tipos de datos \(Db2 SQL\)](#)

[Normas para tipos de datos de resultados \(Db2 SQL\)](#)

[Tipos diferenciados](#)

Un *tipo distinto* es un tipo de datos definido por el usuario que comparte su representación interna con un tipo de datos incorporado (su *tipo de origen*), pero se considera un tipo de datos independiente e incompatible para la mayoría de las operaciones.

[Tipos de datos en Db2 for z/OS \(Db2 SQL\)](#)

Almacenamiento de datos LOB en tablas de Db2

Db2 maneja los datos LOB de manera diferente a otros tipos de datos. Como resultado, a veces es necesario realizar acciones adicionales al definir columnas LOB e insertar los datos LOB.

Antes de empezar

Db2 a veces crea implícitamente el espacio de tabla LOB, la tabla auxiliar y el índice en la tabla auxiliar para cada columna LOB en una tabla o partición. Para obtener más información, consulte [Creación implícita de espacio en la tabla LOB \(Guía de administración de Db2\)](#).

Si Db2 no crea implícitamente los espacios de tabla LOB, las tablas auxiliares y los índices en las tablas auxiliares, debe crear estos objetos emitiendo instrucciones CREATE TABLESPACE, CREATE AUXILIARY TABLE y CREATE INDEX.

Acerca de esta tarea

Los *objetos grandes* y *LOB* se refieren a objetos de almacenamiento de objetos (Db2) que puede utilizar para almacenar grandes cantidades de datos. Un LOB es una cadena de caracteres de longitud variable que puede contener hasta 2 GB (1 byte de datos). Db2 admite los siguientes tipos de datos LOB:

Objeto grande binario (BLOB)

Utilice un BLOB para almacenar datos binarios como imágenes, voz y medios mixtos.

Objeto grande de caracteres (CLOB)

Utilice un CLOB para almacenar datos SBCS o datos de caracteres mixtos, como documentos.

objeto grande de caracteres de doble byte (DBCLOB)

Utilice un DBCLOB para almacenar datos que consistan únicamente en datos DBCS.

Para obtener más información sobre los tipos de datos LOB, consulte [Objetos grandes \(LOB\) \(Db2 SQL\)](#).

Puede utilizar Db2 para almacenar datos LOB, pero estos datos se almacenan de forma diferente a otros tipos de datos.

Aunque una tabla puede tener una columna LOB, los datos LOB reales se almacenan en otra tabla, llamada *tabla auxiliar*. Esta tabla auxiliar existe en un espacio de tabla independiente llamado *espacio de tabla LOB*. Debe existir una tabla auxiliar para cada columna LOB. La tabla con la columna LOB se denomina tabla base. La tabla base tiene una columna ROWID que utiliza Db2 para localizar los datos en la tabla auxiliar. La tabla auxiliar debe tener exactamente un índice.

Procedimiento

Para almacenar datos LOB en Db2, siga estos pasos:

1. Opcional: Defina como máximo una columna ROWID cuando cree o modifique la tabla, incluso si la tabla va a tener varias columnas LOB. Si no crea una columna ROWID antes de definir una columna LOB, e Db2 o crea implícitamente una columna ROWID con el atributo IMPLICITLY HIDDEN y la añade como última columna de la tabla.

Si agrega una columna ROWID después de agregar una columna LOB, la tabla tendrá dos columnas ROWID: la columna creada implícitamente y la columna creada explícitamente. En este caso, Db2 garantiza que los valores de las dos columnas ROWID sean siempre idénticos.

2. Defina una o más columnas del tipo de columna LOB apropiado emitiendo una instrucción CREATE TABLE o una o más instrucciones ALTER TABLE.
3. Cree espacios de tabla y tablas auxiliares para los datos LOB, a menos que Db2 los crea implícitamente para usted. Para obtener más información, consulte [Creación implícita de espacio en la tabla LOB \(Guía de administración de Db2\)](#).

Debe crear un espacio de tabla LOB para cada partición de tabla y una tabla auxiliar para cada columna LOB. Por ejemplo, si su tabla base tiene tres particiones, debe crear tres espacios de tabla LOB y tres tablas auxiliares para cada columna LOB. Utilice las siguientes instrucciones para crear estos objetos: [CREATE LOB TABLESPACE \(Db2 SQL\)](#) y [CREATE AUXILIARY TABLE declaración \(Db2 SQL\)](#).

El conjunto de privilegios debe incluir los siguientes privilegios:

- El privilegio USE en el grupo de almacenamiento intermedio y el grupo de almacenamiento que utilizan los objetos LOB
 - Si el espacio de la tabla base se crea explícitamente, también se requiere CREATETS en la base de datos que contiene la tabla (DSNDB04 si la base de datos se crea implícitamente)
4. Cree un índice para cada tabla auxiliar utilizando la instrucción CREATE INDEX. Cada tabla auxiliar debe tener exactamente un índice en el que cada entrada de índice haga referencia a un LOB.
 5. Inserte los datos LOB en Db2 utilizando una de las siguientes técnicas:
 - Si la longitud total de una columna LOB y la fila de la tabla base es inferior a 32 KB, utilice la utilidad LOAD y especifique la tabla base.
 - De lo contrario, utilice instrucciones INSERT, UPDATE o MERGE y especifique la tabla base. Si utiliza la instrucción INSERT, asegúrese de que su aplicación tiene suficiente almacenamiento disponible para contener todo el valor que se va a poner en la columna LOB.

Ejemplo

Supongamos que desea añadir un currículum para cada empleado a la tabla de empleados. Los currículos de los empleados no superan los 5 MB de tamaño. Debido a que los currículos de los empleados contienen caracteres de un solo byte, puede definir los currículos como CLOBs en Db2. Por lo tanto, debe agregar una columna de tipo de datos CLOB con una longitud de 5 MB a la tabla de empleados. Si desea definir una columna ROWID explícitamente, debe definirla antes de definir la columna CLOB.

Primero, ejecute una instrucción ALTER TABLE para añadir la columna ROWID y, a continuación, ejecute otra instrucción ALTER TABLE para añadir la columna CLOB. Las siguientes declaraciones crean estas columnas:

```
ALTER TABLE EMP
  ADD ROW_ID ROWID NOT NULL GENERATED ALWAYS;
COMMIT;
ALTER TABLE EMP
  ADD EMP_RESUME CLOB(5M);
COMMIT;
```

Si ha creado explícitamente el espacio de tabla para esta tabla y el registro especial CURRENT RULES no está establecido en STD, entonces necesita definir un espacio de tabla LOB y una tabla auxiliar para contener los currículums de los empleados. También debe definir un índice en la tabla auxiliar. Debe definir el espacio de tablas de LOB en la misma base de datos que la tabla base asociada. Las siguientes declaraciones crean estos objetos:

```
CREATE LOB TABLESPACE RESUMETS
  IN DSN8D12A
  LOG NO
COMMIT;
CREATE AUXILIARY TABLE EMP_RESUME_TAB
  IN DSN8D12A.RESUMETS
  STORES DSN8C10.EMP
  COLUMN EMP_RESUME;
CREATE UNIQUE INDEX XEMP_RESUME
  ON EMP_RESUME_TAB;
COMMIT;
```

A continuación, puede cargar los currículos de sus empleados en Db2. En su aplicación, puede definir una variable de host para contener el currículum, copiar los datos del currículum de un archivo en la variable de host y, a continuación, ejecutar una instrucción UPDATE para copiar los datos en un Db2. Aunque los datos LOB se almacenan en la tabla auxiliar, su instrucción UPDATE especifica el nombre de la tabla base. El siguiente código declara una variable de host para almacenar el currículum en lenguaje C:

```
SQL TYPE is CLOB (5M) resumedata;
```

La siguiente instrucción UPDATE copia los datos en un Db2:

```
UPDATE EMP SET EMP_RESUME=:resumedata
  WHERE EMPNO=:employeenum;
```

En esta declaración, employeenum es una variable de host que identifica al empleado asociado a un currículum.

Conceptos relacionados

[Objetos grandes \(LOB\) \(Db2 SQL\)](#)

Tareas relacionadas

[Creación de objetos grandes \(Introducción a Db2 for z/OS\)](#)

Referencia relacionada

[CREATE TABLE declaración \(Db2 SQL\)](#)

[CREATE AUXILIARY TABLE declaración \(Db2 SQL\)](#)

[CREATE LOB TABLESPACE \(Db2 SQL\)](#)

[CREATE INDEX declaración \(Db2 SQL\)](#)

Columnas de identidad

Una columna de identidad contiene un valor numérico exclusivo para cada fila de la tabla. Db2 puede generar automáticamente valores numéricos secuenciales para esta columna cuando las filas se insertan en la tabla. Así, las columnas de identidad son ideales para los valores de clave primaria, como los números de empleado o los números de producto.

Uso de columnas de identidad como claves

Si define una columna con el atributo AS IDENTITY y con los atributos GENERATED ALWAYS y NO CYCLE, Db2 genera automáticamente un número secuencial que aumenta o disminuye de forma monótona para el valor de esa columna cuando se inserta una nueva fila en la tabla. Sin embargo, para que Db2 garantice que los valores de la columna de identidad son únicos, debe definir un índice único en esa columna.

Puede utilizar columnas de identidad para claves primarias que suelen ser números secuenciales únicos, por ejemplo, números de pedido o números de empleado. De este modo, puede evitar los problemas de concurrencia que pueden producirse cuando una aplicación genera su propio contador único fuera de la base de datos.

Recomendación: Establezca los valores de las claves externas en las tablas dependientes después de cargar la tabla principal. Si utiliza una columna de identidad como clave principal en una estructura de integridad referencial, cargar datos en esa estructura podría ser bastante complicado. Los valores de la columna de identidad no se conocen hasta que se carga la tabla porque la columna se define como GENERADA SIEMPRE.

Es posible que haya lagunas en los valores de la columna de identidad por las siguientes razones:

- Si otras aplicaciones están insertando valores en la misma columna de identidad
- Si Db2 finaliza de forma anormal antes de asignar todos los valores almacenados en caché
- Si su aplicación revierte una transacción que inserta valores de identidad

Definición de una columna de identidad

Puede definir una columna de identidad como GENERADA POR DEFECTO o GENERADA SIEMPRE:

- Si define la columna como GENERATED BY DEFAULT, puede insertar un valor, y Db2 proporciona un valor predeterminado si no proporciona uno.
- Si define la columna como GENERATED ALWAYS (Generada siempre), Db2 genera siempre un valor para la columna y no puede insertar datos en esa columna. Si desea que los valores sean únicos, debe definir la columna de identidad con GENERATED ALWAYS y NO CYCLE y definir un índice único en esa columna.

Los valores que genera Db2 para una columna de identidad dependen de cómo se defina la columna. La opción START WITH (Comenzar con) determina el primer valor que genera Db2 . Los valores avanzan por el valor INCREMENT BY en orden ascendente o descendente.

Las opciones MINVALUE y MAXVALUE determinan los valores mínimo y máximo que genera Db2 . Sin embargo, la opción CYCLE (CICLO) o NO CYCLE (SIN CICLO) determina si Db2 envuelve los valores cuando ha generado todos los valores entre el valor START WITH (INICIAR CON) y MAXVALUE (VALOR MÁXIMO) si los valores son ascendentes, o entre el valor START WITH (INICIAR CON) y MINVALUE (VALOR MÍNIMO) si los valores son descendentes. MINVALUE y MAXVALUE no limitan un valor START WITH o RESTART WITH.

Ejemplo: Uso de GENERATED ALWAYS y CYCLE

Supongamos que la tabla T1 se define con GENERATED ALWAYS y CYCLE:

```
CREATE TABLE T1
(CHARCOL1 CHAR(1),
 IDENTCOL1 SMALLINT GENERATED ALWAYS AS IDENTITY
  (START WITH -1,
   INCREMENT BY 1,
   CYCLE,
   MINVALUE -3,
   MAXVALUE 3));
```

Ahora suponga que ejecuta la siguiente instrucción INSERT ocho veces:

```
INSERT INTO T1 (CHARCOL1) VALUES ('A');
```

Cuando Db2 genera valores para IDENTCOL1, comienza con -1 y se incrementa en 1 hasta alcanzar el MAXVALUE de 3 en el quinto INSERT. Db2 Para generar el valor para el sexto INSERT, el programador vuelve a MINVALUE, que es -3. T1 se ve así después de que se ejecuten las ocho instrucciones INSERT:

CHARCOL1	IDENTCOL1
A	-1
A	0
A	1
A	2
A	3
A	-3
A	-2
A	-1

El valor de IDENTCOL1 para el octavo INSERT repite el valor de IDENTCOL1 para el primer INSERT.

Ejemplo: EMPEZAR CON o REINICIAR CON valores fuera del rango para el ciclo

Las opciones MINVALUE y MAXVALUE no limitan el valor START WITH. Es decir, la cláusula START WITH puede utilizarse para iniciar la generación de valores fuera del rango que se utiliza para los ciclos. Sin embargo, el siguiente valor generado después del valor START WITH especificado es MNVALUE para una columna de identidad ascendente o MAXVALUE para una columna de identidad descendente. Lo mismo ocurre si modifica la columna de identidad y especifica un valor RESTART WITH.

Considera el ejemplo anterior (T1) y supón que modificas la tabla con una instrucción que especifique las siguientes palabras clave.

```
ALTER TABLE T1  
ALTER COLUMN IDENTCOL1 SET GENERATED ALWAYS RESTART WITH 99;
```

Ahora suponga que ejecuta la siguiente instrucción INSERT tres veces:

```
INSERT INTO T1 (CHARCOL1) VALUES ('B');
```

Cuando Db2 genera el valor IDENTCOL1, comienza con 99. Sin embargo, para el siguiente valor generado, Db2 vuelve a pasar por MINVALUE, que es -3. T1 se ve así después de que se ejecuten las tres sentencias INSERT:

CHARCOL1	IDENTCOL1
A	-1
A	0
A	1
A	2
A	3
A	-3
A	-2
A	-1
B	99
B	-3
B	-2

Columnas de identidad como claves primarias

La instrucción SELECT from INSERT le permite insertar una fila en una tabla principal con su clave primaria definida como una columna de identidad generada por el lenguaje de definición de datos (Db2) y recuperar el valor de la clave primaria o principal. A continuación, puede utilizar este valor generado como clave externa en una tabla dependiente.

Además, puede utilizar la función IDENTITY_VAL_LOCAL para devolver el valor asignado más recientemente a una columna de identidad.

Ejemplo: Uso de SELECT desde INSERT

Supongamos que una tabla EMPLOYEE y una tabla DEPARTMENT se definen de la siguiente manera:

```
CREATE TABLE EMPLOYEE
  (EMPNO      INTEGER GENERATED ALWAYS AS IDENTITY
   PRIMARY KEY NOT NULL,
   NAME        CHAR(30) NOT NULL,
   SALARY      DECIMAL(7,2) NOT NULL,
   WORKDEPT    SMALLINT);

CREATE TABLE DEPARTMENT
  (DEPTNO     SMALLINT NOT NULL PRIMARY KEY,
   DEPTNAME   VARCHAR(30),
   MGRNO      INTEGER NOT NULL,
   CONSTRAINT REF_EMPNO FOREIGN KEY (MGRNO)
   REFERENCES EMPLOYEE (EMPNO) ON DELETE RESTRICT);

ALTER TABLE EMPLOYEE ADD
  CONSTRAINT REF_DEPTNO FOREIGN KEY (WORKDEPT)
  REFERENCES DEPARTMENT (DEPTNO) ON DELETE SET NULL;
```

Cuando inserta un nuevo empleado en la tabla EMPLOYEE, para recuperar el valor de la columna EMPNO, puede utilizar la siguiente instrucción SELECT from INSERT:

```
EXEC SQL
  SELECT EMPNO INTO :hv_empno
  FROM FINAL TABLE (INSERT INTO EMPLOYEE (NAME, SALARY, WORKDEPT)
    VALUES ('New Employee', 75000.00, 11));
```

La sentencia SELECT devuelve el valor de identidad generado por el sistema de gestión de bases de datos (Db2) para la columna EMPNO en la variable de host :hv_empno.

A continuación, puede utilizar el valor de :hv_empno para actualizar la columna MGRNO de la tabla DEPARTAMENTO con el nuevo empleado como director de departamento:

```
EXEC SQL
  UPDATE DEPARTMENT
    SET MGRNO = :hv_empno
  WHERE DEPTNO = 11;
```

Conceptos relacionados

[Reglas para insertar datos en una columna de identidad](#)

Una columna de identidad contiene un valor numérico único para cada fila de la tabla. Si puede insertar datos en una columna de identidad y cómo se insertan esos datos depende de cómo se define la columna.

Tareas relacionadas

[Selección de valores al insertar datos](#)

Cuando inserta filas en una tabla, también puede seleccionar valores de las filas insertadas al mismo tiempo.

Referencia relacionada

[Función escalar IDENTITY_VAL_LOCAL \(Db2 SQL\)](#)

Creación de tablas para mantener la integridad de los datos

Para asegurarse de que solo se añadan datos válidos a las tablas, puede utilizar restricciones, desencadenantes e índices exclusivos. Por ejemplo, puede que necesite asegurarse de que todos los elementos de la tabla de inventario tengan números de elemento válidos y para evitar que se añadan elementos sin números de elemento válidos.

Acerca de esta tarea

Conceptos introductorios

[Creación de relaciones con restricciones de referencia \(Introducción a Db2 para z/OS\)](#)

Conceptos relacionados

[Creación de relaciones con restricciones de referencia \(Introducción a Db2 para z/OS\)](#)

Tareas relacionadas

[Modificación de una tabla para mantener la integridad referencial \(Db2 Administration Guide\)](#)

[Creación de índices para mejorar el rendimiento de la integridad referencial de las claves foráneas \(Db2 Performance\)](#)

[Creación de tablas para mantener la integridad de los datos](#)

Para asegurarse de que solo se añadan datos válidos a las tablas, puede utilizar restricciones, desencadenantes e índices exclusivos. Por ejemplo, puede que necesite asegurarse de que todos los elementos de la tabla de inventario tengan números de elemento válidos y para evitar que se añadan elementos sin números de elemento válidos.

[Uso de la integridad de referencia para la coherencia de datos \(Managing Security\)](#)

Modos de mantener la integridad de los datos

Al añadir o modificar datos en una tabla de Db2, debe asegurarse de que los datos son válidos. Puede utilizar dos técnicas para asegurarse de que los datos válidos sean restricciones y desencadenantes.

Las restricciones son reglas que limitan los valores que puede insertar, eliminar o actualizar en una tabla. Hay dos tipos de restricciones:

- Las restricciones de verificación determinan los valores que puede contener una columna. Las restricciones de verificación se tratan en [“Restricciones de comprobación” en la página 137](#).
- Las restricciones referenciales preservan las relaciones entre tablas. Las restricciones referenciales se tratan en [“Restricciones referenciales” en la página 139](#). En [“Restricciones referenciales informativas” en la página 140](#) se analiza un tipo específico de restricciones referenciales, la restricción referencial informativa.

Para mantener la integridad de los datos, Db2 aplica restricciones de verificación y restricciones referenciales a los datos de una tabla. Cuando se infringen o podrían infringirse este tipo de restricciones, Db2 coloca el espacio de tabla o la partición que contiene la tabla en estado CHECK-pending (pendiente de comprobación).

Los desencadenantes son una serie de acciones que se invocan cuando se actualiza una tabla. Los desencadenantes se tratan en [“Creación de un desencadenante” en la página 160](#).

Referencia relacionada

[Estado pendiente de CHECK \(Db2 Utilities\)](#)

Restricciones de comprobación

Una *restricción de comprobación* es una regla que especifica los valores permitidos en una o más columnas de cada fila de una tabla base. Por ejemplo, puede definir una restricción de comprobación para garantizar que todos los valores de una columna que contienen años son números positivos.

Las restricciones de verificación designan los valores que pueden contener columnas específicas de una tabla base, proporcionándole un método para controlar la integridad de los datos introducidos en las tablas. Puede crear tablas con restricciones de verificación utilizando la instrucción CREATE TABLE, o puede añadir las restricciones con la instrucción ALTER TABLE. Sin embargo, si la integridad de la verificación se ve comprometida o no se puede garantizar para una tabla, el espacio de la tabla o la partición que contiene la tabla se coloca en un estado de verificación pendiente. La integridad de la verificación es la condición que existe cuando cada fila de una tabla se ajusta a las restricciones de verificación definidas en esa tabla.

Por ejemplo, es posible que desee asegurarse de que ningún salario puede ser inferior a 15 000 dólares. Para ello, puede crear la siguiente restricción de verificación:

```
CREATE TABLE EMPSAL
  (ID      INTEGER      NOT NULL,
   SALARY  INTEGER      CHECK (SALARY >= 15000));
```

El uso de restricciones de verificación facilita la tarea de programación, ya que no es necesario aplicar dichas restricciones en los programas de aplicación o con una rutina de validación. Definir restricciones de verificación en una o más columnas de una tabla cuando se crea o modifica dicha tabla.

Consideraciones sobre la restricción de cheques

La sintaxis de una restricción de verificación se comprueba cuando se define la restricción, pero no se comprueba el significado de la restricción. Los siguientes ejemplos muestran errores que no se detectan. La columna " C1 " se define como INTEGER NOT NULL.

Restricciones de cheques permitidas pero erróneas:

- Una restricción de verificación autocontradictoria:

```
CHECK (C1 > 5 AND C1 < 2)
```

- Dos restricciones de verificación que se contradicen entre sí:

```
CHECK (C1 > 5)  
CHECK (C1 < 2)
```

- Dos restricciones de verificación, una de las cuales es redundante:

```
CHECK (C1 > 0)  
CHECK (C1 >= 1)
```

- Una restricción de verificación que contradice la definición de la columna:

```
CHECK (C1 IS NULL)
```

- Una restricción de verificación que repite la definición de la columna:

```
CHECK (C1 IS NOT NULL)
```

Una restricción de verificación no se comprueba para ver si es coherente con otros tipos de restricciones. Por ejemplo, una columna de una tabla dependiente puede tener una restricción referencial con una regla de eliminación de SET NULL. También puede definir una restricción de verificación que prohíba los valores nulos en la columna. Como resultado, un intento de eliminar una fila principal falla, porque establecer la fila dependiente en nula viola la restricción de verificación.

Del mismo modo, no se comprueba la coherencia de una restricción de verificación con una rutina de validación, que se aplica a una tabla antes de una restricción de verificación. Si la rutina requiere que una columna sea mayor o igual a 10 y una restricción de verificación requiere que la misma columna sea menor que 10, las inserciones de tabla no son posibles. Los planes y paquetes no necesitan ser reajustados después de que las restricciones de verificación se definen o eliminan de una tabla.

Cuando se aplican restricciones de verificación

Después de definir las restricciones de verificación en una tabla, cualquier cambio debe satisfacer esas restricciones si se realiza mediante:

- La utilidad LOAD con la opción ENFORCE CONSTRAINT
- Una operación de inserción SQL
- Una operación de actualización SQL

Una fila satisface una restricción de verificación si su condición se evalúa como verdadera o como desconocida. Una condición puede evaluarse como desconocida para una fila si una de las columnas nombradas contiene el valor nulo para esa fila.

Cualquier restricción definida en las columnas de una tabla base se aplica a las vistas definidas en esa tabla base.

Cuando se utiliza ALTER TABLE para añadir una restricción de verificación a tablas ya pobladas, la aplicación de la restricción de verificación viene determinada por el valor del registro especial CURRENT RULES de la siguiente manera:

- Si el valor es STD, la restricción de verificación se aplica inmediatamente cuando se define. Si una fila no se ajusta, la restricción de verificación no se añade a la tabla y se produce un error.
- Si el valor es Db2, la restricción de verificación se añade a la descripción de la tabla, pero su aplicación se aplaza. Debido a que puede haber filas en la tabla que violen la restricción de verificación, la tabla se coloca en estado CHECK-pending.

Restricciones referenciales

Una restricción referencial es una regla que especifica que los únicos valores válidos para una columna concreta son aquellos valores que existen en otra columna de la tabla especificada. Por ejemplo, una restricción de referencia puede garantizar que todos los ID de cliente de una tabla de transacciones existen en la columna de ID de una tabla de clientes.

Una tabla puede servir como "lista maestra" de todas las apariciones de una entidad. En la aplicación de muestra, la tabla de empleados sirve para eso; los números que aparecen en esa tabla son los únicos números de empleado válidos. Del mismo modo, la tabla de departamentos proporciona una lista maestra de todos los números de departamento válidos; la tabla de actividades del proyecto proporciona una lista maestra de las actividades realizadas para los proyectos; y así sucesivamente.

La siguiente figura muestra las relaciones que existen entre las tablas en la aplicación de muestra. Las flechas apuntan desde las tablas principales a las tablas dependientes.

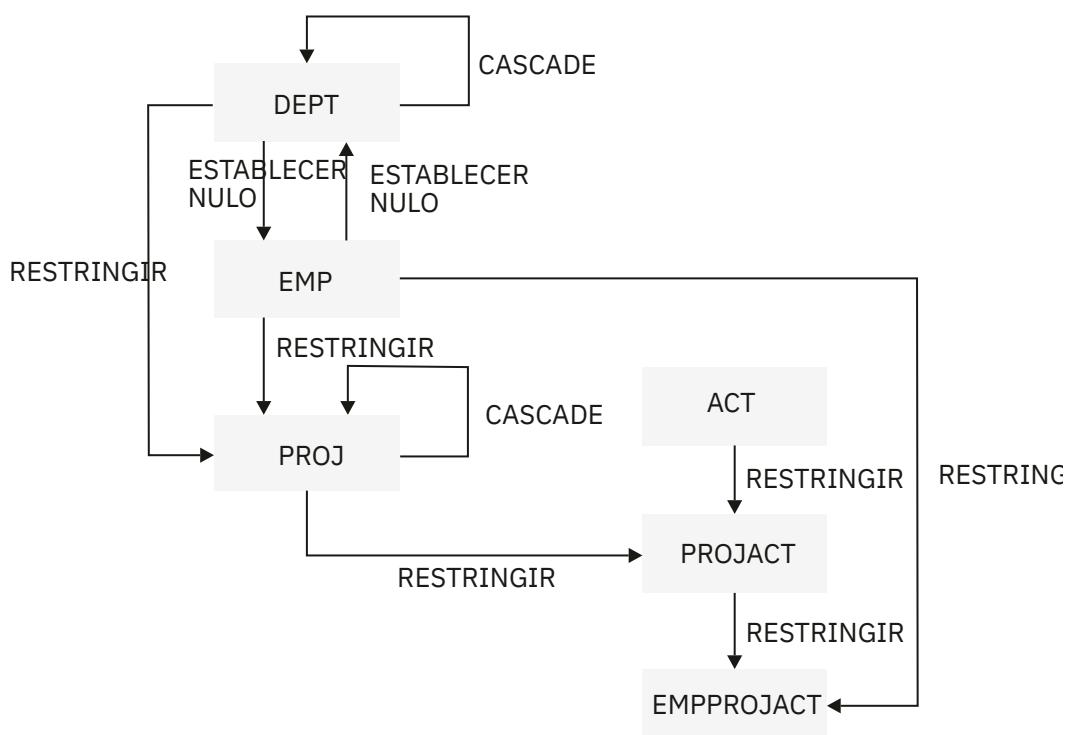


Figura 4. Relaciones entre tablas en la aplicación de ejemplo

Cuando una tabla se refiere a una entidad para la que existe una lista maestra, debe identificar una ocurrencia de la entidad que realmente aparece en la lista maestra; de lo contrario, la referencia no es válida o la lista maestra está incompleta. Las restricciones referenciales refuerzan la relación entre una tabla y una lista maestra.

Restricciones en los ciclos de las tablas dependientes

Un ciclo es un conjunto de dos o más tablas. Las tablas están ordenadas de manera que cada una depende de la anterior, y la primera depende de la última. Cada tabla del ciclo es descendiente de sí misma. Db2 restringe ciertas operaciones en ciclos.

En la aplicación de muestra, las tablas de empleados y departamentos son un ciclo; cada una depende de la otra.

Db2 no permite crear un ciclo en el que una operación de eliminación en una tabla implique esa misma tabla. La aplicación de ese principio crea reglas sobre la adición de una clave foránea a una tabla:

- En un ciclo de dos tablas, ninguna de las dos reglas de eliminación puede ser CASCADE.
- En un ciclo de más de dos tablas, dos o más reglas de eliminación no deben ser CASCADE. Por ejemplo, en un ciclo con tres tablas, dos de las reglas de eliminación deben ser distintas de CASCADE. Este concepto se ilustra en la siguiente figura. El ciclo de la izquierda es válido porque dos o más de las reglas de eliminación no son CASCADE. El ciclo de la derecha no es válido porque contiene dos eliminaciones en cascada.

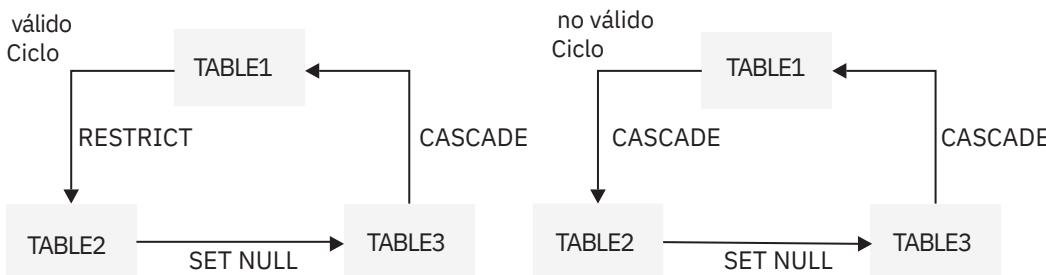


Figura 5. Ciclos de eliminación válidos e inválidos

De forma alternativa, una operación de eliminación en una tabla autorreferenciada debe implicar la misma tabla, y la regla de eliminación debe ser CASCADA o SIN ACCIÓN.

Recomendación: Evite crear un ciclo en el que todas las reglas de eliminación sean RESTRICT y ninguna de las claves externas permita valores nulos. Si lo hace, no podrá eliminar ninguna fila de ninguna de las tablas.

Restricciones referenciales en tablas con seguridad multinivel con granularidad a nivel de fila

No se pueden utilizar restricciones referenciales en una columna de etiqueta de seguridad, que se utiliza para la seguridad multinivel con granularidad a nivel de fila. Sin embargo, puede utilizar restricciones referenciales en otras columnas de la fila.

Db2 no aplica seguridad multinivel con granularidad a nivel de fila cuando ya está aplicando restricciones referenciales. Las restricciones referenciales se aplican cuando se producen las siguientes situaciones:

- Se aplica una operación de inserción a una tabla dependiente.
- Se aplica una operación de actualización a una clave foránea de una tabla dependiente o a la clave principal de una tabla principal.
- Se aplica una operación de eliminación a una tabla principal. Además de que se apliquen todas las restricciones referenciales, el sistema de gestión de versiones (Db2) aplica todas las reglas de eliminación para todas las filas dependientes que se ven afectadas por la operación de eliminación. Si no se cumplen todas las restricciones referenciales y reglas de eliminación, la operación de eliminación no tendrá éxito.
- Se ejecuta el programa de utilidad de carga con la opción ENFORCE CONSTRAINTS en una tabla dependiente.
- Se ejecuta el programa de utilidad CHECK DATA.

Conceptos relacionados

Seguridad multinivel (Gestión de la seguridad)

Restricciones referenciales informativas

Una restricción referencial informativa es una restricción referencial que Db2 no aplica durante las operaciones normales. Utilice estas restricciones solo cuando la integridad referencial pueda aplicarse por otros medios, como al recuperar datos de otras fuentes. Estas restricciones podrían mejorar el rendimiento al permitir que la consulta cumpla los requisitos para la reescritura automática de consultas.

Db2 ignora las restricciones referenciales de información durante las operaciones de inserción, actualización y eliminación. Algunas utilidades ignoran estas restricciones; otras las reconocen. Por ejemplo, CHECK DATA y LOAD ignoran estas restricciones. QUIESCE TABLESPACESET reconoce estas restricciones al poner en reposo todos los espacios de tabla relacionados con el espacio de tabla especificado.

Debería utilizar este tipo de restricción de referencia únicamente cuando un proceso de aplicaciones comprueba los datos en una relación de integridad de referencia. Por ejemplo, al insertar una fila en una tabla dependiente, la aplicación debe verificar que exista una clave externa como clave principal o única en la tabla principal. Para definir una restricción referencial informativa, utilice la opción NOT ENFORCED de la definición de restricción referencial en una sentencia CREATE TABLE o ALTER TABLE.

Las restricciones referenciales informativas suelen ser útiles, especialmente en un entorno de almacén de datos, por varias razones:

- Para evitar los gastos generales de la aplicación por parte de Db2.

Por lo general, los datos de un almacén de datos se han extraído y depurado de otras fuentes. La integridad referencial puede que ya esté garantizada. En esta situación, la aplicación por parte de Db2 es innecesaria.

- Para permitir que más consultas puedan reescribirse automáticamente.

La reescritura automática de consultas es un proceso que examina una consulta enviada que hace referencia a tablas de origen y, si procede, reescribe la consulta para que se ejecute contra una tabla de consulta materializada que se ha derivado de esas tablas de origen. Este proceso utiliza restricciones referenciales informativas para determinar si la consulta puede utilizar una tabla de consulta materializada. La reescritura automática de consultas da como resultado una reducción significativa del tiempo de ejecución de las consultas, especialmente en el caso de las consultas de apoyo a la toma de decisiones que operan sobre enormes cantidades de datos.

Tareas relacionadas

[Utilización de tablas de consulta materializadas para mejorar el rendimiento de SQL \(Db2 Performance\)](#)

Referencia relacionada

[CREATE TABLE declaración \(Db2 SQL\)](#)

Definición de una clave padre y un índice exclusivo

Una *clave padre* es una clave primaria o una clave exclusiva de la tabla padre de una restricción de referencia. Los valores de una clave padre determinan los valores válidos de la clave foránea en la restricción. Debe crear un índice exclusivo en una clave padre.

Acerca de esta tarea

La clave principal de una tabla, si existe, identifica de forma única cada ocurrencia de una entidad en la tabla. La cláusula PRIMARY KEY de las sentencias CREATE TABLE o ALTER TABLE identifica la columna o columnas de la clave principal. Cada columna identificada debe definirse como NOT NULL.

Otra forma de permitir solo valores únicos en una columna es especificar la cláusula UNIQUE al crear o modificar una tabla.

Una tabla que va a ser matriz de tablas dependientes debe tener una clave primaria o única; las claves foráneas de las tablas dependientes se refieren a la clave primaria o única. De lo contrario, una clave principal es opcional. Considere definir una clave principal si cada fila de su tabla pertenece a una ocurrencia única de alguna entidad. Si define una clave principal, debe crearse un índice (*el índice principal*) en el mismo conjunto de columnas, en el mismo orden que esas columnas. Si está definiendo restricciones referenciales para que las aplique Db2 , tome medidas para mantener la integridad de los datos leídos antes de crear o modificar cualquiera de las tablas involucradas.

Una tabla no puede tener más de una clave primaria. Una clave principal tiene las mismas restricciones que las claves de índice:

- La clave no puede incluir más de 64 columnas.

- No puede especificar dos veces el nombre de una columna.
- La suma de los atributos de longitud de columna no puede ser superior a 2000.

Usted define una lista de columnas como la clave principal de una tabla con la cláusula PRIMARY KEY en la instrucción CREATE TABLE.

Procedimiento

Utilice la cláusula PRIMARY KEY en una instrucción ALTER TABLE. En este caso, ya debe existir un índice único.

Tenga en cuenta los siguientes elementos cuando planifique las claves principales:

- El modelo teórico de una base de datos relacional sugiere que cada tabla debe tener una clave principal para identificar de forma única las entidades que describe. Sin embargo, debe sopesar ese modelo con el posible coste de los gastos generales de mantenimiento del índice. Db2 no requiere que defina una clave principal para tablas sin dependientes.
- Elija una clave principal cuyos valores no cambien con el tiempo. Elegir una clave principal con valores persistentes refuerza la buena práctica de tener identificadores únicos que permanezcan iguales durante la vida de la ocurrencia de la entidad.
- Una columna de clave principal no debe tener valores predeterminados a menos que la clave principal sea una sola columna TIMESTAMP.
- Elija el número mínimo de columnas para garantizar la singularidad de la clave principal.
- Una vista que se puede actualizar y que se define en una tabla con una clave principal debe incluir todas las columnas de la clave. Aunque esto solo es necesario si la vista se utiliza para inserciones, la identificación única de filas puede ser útil si la vista se utiliza para actualizaciones, eliminaciones o selecciones.
- Elimine una clave principal más adelante si cambia su base de datos o aplicación utilizando SQL.

Conceptos relacionados

[Modos de mantener la integridad de los datos](#)

Al añadir o modificar datos en una tabla de Db2, debe asegurarse de que los datos son válidos. Puede utilizar dos técnicas para asegurarse de que los datos válidos sean restricciones y desencadenantes.

Referencia relacionada

[ALTER TABLE declaración \(Db2 SQL\)](#)

[CREATE TABLE declaración \(Db2 SQL\)](#)

Columnas clave padre

Una *clave padre* es una clave primaria o una clave exclusiva de la tabla padre de una restricción de referencia. Esta clave consta de una columna o conjunto de columnas. Los valores de una clave padre determinan los valores válidos de la clave foránea en la restricción.

Si cada fila de una tabla representa relaciones para una entidad única, la tabla debe tener una columna o un conjunto de columnas que proporcione un identificador único para las filas de la tabla. Esta columna (o conjunto de columnas) se denomina *clave principal* de la tabla. Para asegurarse de que la clave principal no contenga valores duplicados, debe crear un índice único en la columna o columnas que constituyen la clave principal. La definición de la clave principal se denomina integridad de la entidad, ya que requiere que cada entidad tenga una clave única.

En algunos casos, puede ser útil utilizar una marca de tiempo como parte de la clave, por ejemplo, cuando una tabla no tiene una clave única "natural" o si la secuencia de llegada es la clave.

Las claves principales para algunas de las tablas de muestra son:

Tabla

Columna de clave

Tabla de empleados

EMPNO

Tabla DEPARTMENT

DEPTNO

Tabla PROJECT

PROJNO

Tabla 36 en la página 143 muestra parte de la tabla del proyecto que tiene la columna de clave principal, PROJNO.

Tabla 36. Parte de la tabla de proyectos con la columna de clave principal, PROJNO		
PROJNO	PROJNAME	DEPTNO
MA2100	WELD LINE AUTOMATION	D01
MA2110	W L PROGRAMMING	D11

Tabla 37 en la página 143 muestra parte de la tabla de actividades del proyecto, que tiene una clave principal que contiene más de una columna. La clave principal es *una clave compuesta*, que consta de las columnas PRONNO, ACTNO y ACSTDATE.

Tabla 37. Parte de la tabla de actividades del proyecto con una clave principal compuesta				
PROJNO	ACTNO	ACSTAFF	ACSTDATE	ACENDATE
AD3100	10	.50	1982-01-01	1982-07-01
AD3110	10	1.00	1982-01-01	1983-01-01
AD3111	60	.50	1982-03-15	1982-04-15

Definición de una clave foránea

Utilice las claves foráneas para imponer relaciones de referencia entre las tablas. Una clave foránea es una columna o conjunto de columnas que hace referencia a la clave principal en la tabla principal.

Antes de empezar

Se cumplen los siguientes requisitos previos:

- El conjunto de privilegios debe incluir el privilegio ALTER o REFERENCES en las columnas de la clave principal.
- Existe un índice único en las columnas de clave principal de la tabla principal.

Procedimiento

Para definir una clave externa, utilice uno de los siguientes métodos:

- Emitir una declaración CREATE TABLE y especificar una cláusula FOREIGN KEY.
 - Elija un nombre de restricción para la relación que se define mediante una clave externa.
Si no elige un nombre, Db2 genera uno a partir del nombre de la primera columna de la clave externa, de la misma manera que genera el nombre de un espacio de tabla creado implícitamente.
Por ejemplo, los nombres de las relaciones en las que la tabla de actividad empleado-proyecto es dependiente se registrarían, por defecto, (en la columna RELNAME de SYSIBM.SYSFOREIGNKEYS) como EMPNO y PROJNO.

El nombre se utiliza en mensajes de error, consultas al catálogo y sentencias DROP FOREIGN KEY. Por lo tanto, es posible que desee elegir una si está experimentando con el diseño de su base de datos y tiene más de una clave foránea que comienza con la misma columna (de lo contrario, Db2 genera el nombre).

- Especifique los nombres de las columnas que identifican las columnas de la clave principal.

Una clave foránea puede referirse a una clave única o a una clave principal de la tabla principal. Si la clave externa hace referencia a una clave única no principal, debe especificar los nombres

de las columnas de la clave de forma explícita. Si los nombres de las columnas de la clave no se especifican explícitamente, el valor predeterminado es hacer referencia a los nombres de las columnas de la clave principal de la tabla principal.

- Emitir una instrucción ALTER TABLE y especificar la cláusula FOREIGN KEY.

Puede añadir una clave externa a una tabla existente; de hecho, a veces es la única forma de proceder. Para que una tabla sea autorreferencial, debe añadir una clave externa después de crearla. Cuando se añade una clave foránea a una tabla poblada, el espacio de la tabla se pone en estado CHECK-pending.

Ejemplo

El siguiente ejemplo muestra una instrucción CREATE TABLE que especifica los nombres de restricción REPAPA y REPAE para las claves externas en la tabla de actividad empleado-a-proyecto.

```
CREATE TABLE DSN8C10.EMPPROJACT
  (EMPNO      CHAR(6)          NOT NULL,
   PROJNO     CHAR(6)          NOT NULL,
   ACTNO      SMALLINT        NOT NULL,
   CONSTRAINT REPAPA FOREIGN KEY (PROJNO, ACTNO)
     REFERENCES DSN8C10.PROJACT ON DELETE RESTRICT,
   CONSTRAINT REPAE FOREIGN KEY (EMPNO)
     REFERENCES DSN8C10.EMP ON DELETE RESTRICT)
IN DATABASE DSN8D12A;
```

Qué hacer a continuación

[Si se eliminan a menudo filas de la tabla principal, es mejor crear un índice en la clave foránea.](#)

Tareas relacionadas

[Adición de claves padre y claves foráneas \(Db2 Administration Guide\)](#)

Referencia relacionada

[CREATE TABLE declaración \(Db2 SQL\)](#)

[ALTER TABLE declaración \(Db2 SQL\)](#)

[Tabla de catálogo SYSFOREIGNKEYS \(Db2 SQL\)](#)

Mantener la integridad referencial al utilizar el cifrado de datos

Si utiliza datos cifrados en una restricción referencial, la clave principal de la tabla principal y la clave externa de la tabla dependiente deben tener el mismo valor cifrado.

Acerca de esta tarea

El valor cifrado debe extraerse de la tabla principal (la clave primaria) y utilizarse para la tabla dependiente (la clave externa). Puede hacerlo de una de las dos formas siguientes:

- Utilice la cláusula FINAL TABLE en una sentencia SELECT de UPDATE, SELECT de INSERT o SELECT de MERGE.
- Utilice la función ENCRYPT_TDES para cifrar la clave externa utilizando la misma contraseña que la clave principal. El valor cifrado de la clave externa será el mismo que el valor cifrado de la clave principal.

La sentencia SET ENCRYPTION PASSWORD establece la contraseña que se utilizará para la función ENCRYPT_TDES.

Referencia relacionada

[Función escalar ENCRYPT_TDES o ENCRYPT \(Db2 SQL\)](#)

[CONTRASEÑA DE ENCRIPCIÓN registro especial \(Db2 SQL\)](#)

Creación de tablas de trabajo para las tablas de muestra EMP y DEPT

Antes de probar sentencias SQL que insertan, actualizan y eliminan filas en el DSN8C10.EMP y DSN8C10.DEPT tablas de muestra, debe crear duplicados de estas tablas. Cree duplicados para que

las tablas de muestra originales permanezcan intactas. Estas tablas duplicadas se denominan *tablas de trabajo*.

Acerca de esta tarea

Este tema muestra cómo crear las tablas de trabajo de departamento y empleado y cómo llenar una tabla de trabajo con el contenido de otra tabla:

Cada uno de estos temas asume que ha iniciado sesión utilizando su propio ID de autorización. El ID de autorización califica el nombre de cada objeto que usted cree. Por ejemplo, si su ID de autorización es SMITH y crea la tabla YDEPT, el nombre de la tabla será SMITH.YDEPT. Si desea acceder a la tabla DSN8C10.DEPT, debe referirse a ella por su nombre completo. Si desea acceder a su propia tabla YDEPT, solo tiene que referirse a ella como YDEPT.

Utilice las siguientes sentencias para crear una nueva tabla de departamentos llamada YDEPT, basada en la tabla existente, DSN8C10.DEPT, y un índice para YDEPT:

```
CREATE TABLE YDEPT  
  LIKE DSN8C10.DEPT;  
  
CREATE UNIQUE INDEX YDEPTX  
  ON YDEPT (DEPTNO);
```

Si desea que DEPTNO sea una clave principal, como en la tabla de ejemplo, defina explícitamente la clave. Utilice una instrucción ALTER TABLE, como en el siguiente ejemplo:

```
ALTER TABLE YDEPT  
  PRIMARY KEY(DEPTNO);
```

Puede utilizar una instrucción INSERT para copiar las filas de la tabla de resultados de una fullselect de una tabla a otra. La siguiente declaración copia todas las filas de DSN8C10.DEPT a su propia mesa de trabajo YDEPT:

```
INSERT INTO YDEPT  
  SELECT *  
    FROM DSN8C10.DEPT;
```

Para obtener información sobre el uso de la instrucción INSERT, consulte “[Inserción de filas utilizando la sentencia INSERT](#)” en la página 352.

Puede utilizar las siguientes sentencias para crear una nueva tabla de empleados llamada YEMP:

```
CREATE TABLE YEMP  
(EMPNO   CHAR(6)      PRIMARY KEY NOT NULL,  
  FIRSTNME VARCHAR(12)  NOT NULL,  
  MIDINIT  CHAR(1)      NOT NULL,  
  LASTNAME VARCHAR(15)  NOT NULL,  
  WORKDEPT CHAR(3)      REFERENCES YDEPT  
                        ON DELETE SET NULL,  
  PHONENO  CHAR(4)      UNIQUE NOT NULL,  
  HIREDATE DATE         ,  
  JOB      CHAR(8)      ,  
  EDLEVEL  SMALLINT    ,  
  SEX      CHAR(1)      ,  
  BIRTHDATE DATE         ,  
  SALARY   DECIMAL(9, 2) ,  
  BONUS    DECIMAL(9, 2) ,  
  COMM     DECIMAL(9, 2) );
```

Esta declaración también crea una restricción referencial entre la clave externa en YEMP (WORKDEPT) y la clave primaria en YDEPT (DEPTNO). También restringe todos los números de teléfono a números únicos.

Si desea cambiar la definición de una tabla después de crearla, utilice la instrucción ALTER TABLE con una cláusula RENAME. Si desea cambiar el nombre de una tabla después de crearla, utilice la instrucción RENAME.

Puede cambiar la definición de una tabla utilizando la instrucción ALTER TABLE solo de ciertas maneras. Por ejemplo, puede añadir y eliminar restricciones en las columnas de una tabla. También puede cambiar el tipo de datos de una columna dentro de los tipos de datos de caracteres, dentro de los tipos de datos numéricos y dentro de los tipos de datos gráficos. Puede añadir una columna a una tabla. Sin embargo, no puede utilizar la instrucción ALTER TABLE para eliminar una columna de una tabla.

Tareas relacionadas

[Modificación de tablas de Db2 \(Db2 Administration Guide\)](#)

Referencia relacionada

[ALTER TABLE declaración \(Db2 SQL \)](#)

[RENAME declaración \(Db2 SQL \)](#)

Creación de tablas temporales creadas

Utilice tablas temporales creadas cuando necesite almacenar datos solo durante el proceso de una aplicación, pero desee compartir la definición de la tabla.

Acerca de esta tarea

Db2 no realiza operaciones de registro y bloqueo para tablas temporales creadas. Por lo tanto, las sentencias SQL que utilizan estas tablas pueden ejecutar consultas de manera eficiente.

Cada proceso de aplicaciones tiene su propia instancia de la tabla temporal creada.

Una instancia de una tabla temporal creada existe en el servidor actual hasta que se produce una de las siguientes acciones:

- El proceso de aplicaciones termina.
- La conexión del servidor remoto a través del cual se creó la instancia finaliza.
- La unidad de trabajo en la que se creó la instancia se completa.

Cuando ejecutas una sentencia ROLLBACK, Db2 elimina la instancia de la tabla temporal creada.

Cuando ejecutas una instrucción COMMIT, el lenguaje de consulta de bases de datos (Db2) elimina la instancia de la tabla temporal creada a menos que se defina un cursor para acceder a la tabla temporal creada con la cláusula WITH HOLD y esté abierto.

La definición de una tabla temporal creada se realiza mediante la instrucción SQL CREATE GLOBAL TEMPORARY TABLE.

Procedimiento

Para crear una tabla temporal creada:

1. Defina la tabla emitiendo la instrucción CREATE GLOBAL TEMPORARY TABLE.

Por ejemplo, la siguiente instrucción crea la definición de una tabla llamada TEMPPROD:

```
CREATE GLOBAL TEMPORARY TABLE TEMPPROD  
  (SERIAL      CHAR(8)      NOT NULL,  
   DESCRIPTION  VARCHAR(60)  NOT NULL,  
   MFGCOST     DECIMAL(8,2),  
   MFGDEPT    CHAR(3),  
   MARKUP      SMALLINT,  
   SALESDEPT   CHAR(3),  
   CURDATE     DATE        NOT NULL);
```

También puede crear esta misma definición copiando la definición de una tabla base (llamada PROD) utilizando la cláusula LIKE:

```
CREATE GLOBAL TEMPORARY TABLE TEMPPROD LIKE PROD;
```

Restricción: No puede utilizar la sentencia MERGE con tablas temporales creadas.

Las sentencias SQL de los ejemplos crean definiciones idénticas para la tabla TEMPPROD, pero estas tablas difieren ligeramente de la tabla de muestra PROD. La tabla de muestras PROD contiene dos

columnas, DESCRIPTION y CURDATE, que están definidas como NOT NULL WITH DEFAULT. Debido a que las tablas temporales creadas no admiten valores predeterminados no nulos, las columnas DESCRIPTION y CURDATE de la tabla TEMPPROD se definen como NOT NULL y no tienen valores predeterminados.

Después de ejecutar una de las dos sentencias CREATE, la definición de TEMPPROD existe, pero no existen instancias de la tabla.

2. Crear una instancia de la tabla temporal creada utilizándola en una aplicación.

Db2 crea una instancia de la tabla cuando se especifica en una de las siguientes sentencias SQL:

- OPEN
- SELECT
- INSERT
- DELETE

Por ejemplo, supongamos que ha definido TEMPPROD como se describe en el paso anterior y, a continuación, ejecuta una aplicación que contiene las siguientes sentencias:

```
EXEC SQL DECLARE C1 CURSOR FOR SELECT * FROM TEMPPROD;
EXEC SQL INSERT INTO TEMPPROD SELECT * FROM PROD;
EXEC SQL OPEN C1;
:
EXEC SQL COMMIT;
:
EXEC SQL CLOSE C1;
```

Cuando ejecutas la instrucción INSERT, Db2 crea una instancia de TEMPPROD y rellena esa instancia con filas de la tabla PROD. Cuando se ejecuta la sentencia COMMIT, el procedimiento almacenado (Db2) elimina todas las filas de TEMPPROD. Sin embargo, supongamos que cambia la declaración de cursor C1 por la siguiente declaración:

```
EXEC SQL DECLARE C1 CURSOR WITH HOLD
FOR SELECT * FROM TEMPPROD;
```

En este caso, Db2 no elimina el contenido de TEMPPROD hasta que finaliza la aplicación porque C1, un cursor que se define con la cláusula WITH HOLD, está abierto cuando se ejecuta la instrucción COMMIT. En cualquier caso, Db2 elimina la instancia de TEMPPROD cuando finaliza la aplicación.

3. Cuando la tabla ya no sea necesaria, emita una instrucción DROP.

Por ejemplo, para eliminar la definición de TEMPPROD, debe ejecutar la siguiente instrucción:

```
DROP TABLE TEMPPROD;
```

Referencia relacionada

[CREATE GLOBAL TEMPORARY TABLE declaración \(Db2 SQL\)](#)

[DROP declaración \(Db2 SQL\)](#)

Tablas temporales

Utilice tablas temporales cuando necesite almacenar datos únicamente para la duración de un proceso de aplicación. En función de si desea compartir la definición de tabla, podrá crear una tabla temporal creada o una tabla temporal declarada.

Los dos tipos de tablas temporales son:

- Creó tablas temporales, que define mediante una instrucción CREATE GLOBAL TEMPORARY TABLE
- Tablas temporales declaradas, que se definen mediante una instrucción DECLARE GLOBAL TEMPORARY TABLE

Las sentencias SQL que utilizan tablas temporales pueden ejecutarse más rápidamente por las siguientes razones:

- Db2 no proporciona registro alguno de las mesas temporales creadas. Para las tablas temporales declaradas, Db2 proporciona un registro limitado que puede limitarse aún más mediante la opción NOT LOGGED de la instrucción DECLARE GLOBAL TEMPORARY TABLE.
- Db2 no proporciona bloqueo para las tablas temporales creadas. Db2 , ofrece un bloqueo limitado para las mesas temporales declaradas.

Las tablas temporales son especialmente útiles cuando se necesita ordenar o consultar tablas de resultados intermedios que contienen un gran número de filas, pero se desea almacenar solo un pequeño subconjunto de esas filas de forma permanente.

Las tablas temporales también pueden devolver conjuntos de resultados de procedimientos almacenados. Los siguientes temas proporcionan más detalles sobre las tablas temporales creadas y las tablas temporales declaradas:

- [“Creación de tablas temporales creadas” en la página 146](#)
- [“Creación de tablas temporales declaradas” en la página 148](#)

Para obtener más información, consulte [“Escribir un procedimiento externo para devolver conjuntos de resultados a un cliente distribuido” en la página 293.](#)

Creación de tablas temporales declaradas

Utilice tablas temporales declaradas cuando necesite almacenar datos solo durante el proceso de una aplicación y no necesite compartir la definición de la tabla. La definición de esta tabla solo existe mientras se ejecuta el proceso de solicitud. Db2 realiza operaciones limitadas de registro y bloqueo para tablas temporales declaradas.

Antes de empezar

Antes de poder definir tablas temporales declaradas, debe tener una base de datos WORKFILE que tenga al menos un espacio de tabla con un tamaño de página de 32 KB.

Acerca de esta tarea

Se crea una instancia de una tabla temporal declarada mediante la instrucción SQL DECLARE GLOBAL TEMPORARY TABLE. Solo el proceso de aplicación en el que se declara la tabla conoce esa instancia, por lo que puede declarar tablas temporales con el mismo nombre en diferentes aplicaciones. El calificador para una tabla temporal declarada es SESSION.

Para crear una tabla temporal declarada, especifique la instrucción DECLARE GLOBAL TEMPORARY TABLE. En esa declaración, especifique las columnas que debe contener la tabla realizando una de las siguientes acciones:

Procedimiento

Para crear una tabla temporal declarada:

1. Emitir una declaración DECLARE GLOBAL TEMPORARY TABLE.

En esa declaración, puede especificar las columnas que debe contener la tabla realizando una de las siguientes acciones:

- Especifique todas las columnas de la tabla. Por ejemplo, la siguiente declaración define una tabla temporal declarada llamada TEMPPROD especificando explícitamente las columnas.

```
DECLARE GLOBAL TEMPORARY TABLE TEMPPROD
  (SERIAL      CHAR(8)      NOT NULL WITH DEFAULT '99999999',
   DESCRIPTION  VARCHAR(60)    NOT NULL,
   PRODCOUNT   INTEGER GENERATED ALWAYS AS IDENTITY,
   MFGCOST     DECIMAL(8,2),
   MFGDEPT    CHAR(3),
   MARKUP      SMALLINT,
```

```
SALESDEPT    CHAR(3),  
CURDATE      DATE      NOT NULL);
```

A diferencia de las columnas de tablas temporales creadas, las columnas de tablas temporales declaradas pueden incluir la cláusula WITH DEFAULT.

- Utilice una cláusula LIKE para copiar la definición de una tabla base, una tabla temporal creada o una vista. Por ejemplo, la siguiente sentencia define una tabla temporal declarada llamada TEMPPROD copiando la definición de una tabla base. La tabla base tiene una columna de identidad que la tabla temporal declarada también utiliza como columna de identidad.

```
DECLARE GLOBAL TEMPORARY TABLE TEMPPROD LIKE BASEPROD  
INCLUDING IDENTITY COLUMN ATTRIBUTES;
```

- Si la tabla base, la tabla temporal creada o la vista de la que selecciona columnas tienen columnas de identidad, puede especificar que las columnas correspondientes en la tabla temporal declarada también sean columnas de identidad. Para incluir estas columnas de identidad, especifique la cláusula INCLUDING IDENTITY COLUMN ATTRIBUTES cuando defina la tabla temporal declarada.

Si la tabla fuente tiene una columna de marca de tiempo de cambio de fila, puede especificar que esos atributos de columna se hereden en la tabla temporal declarada especificando INCLUDING ROW CHANGE TIMESTAMP COLUMN ATTRIBUTES.

- Utilice un fullselect para elegir columnas específicas de una tabla base, una tabla temporal creada o una vista. Por ejemplo, la siguiente declaración define una tabla temporal declarada llamada TEMPPROD seleccionando columnas de una vista. La vista tiene una columna de identidad que la tabla temporal declarada también utiliza como columna de identidad. La tabla temporal declarada hereda sus valores de columna predeterminados de los valores de columna predeterminados de una tabla base en la que se basa la vista.

```
DECLARE GLOBAL TEMPORARY TABLE TEMPPROD  
AS (SELECT * FROM PRODVIEW)  
DEFINITION ONLY  
INCLUDING IDENTITY COLUMN ATTRIBUTES  
INCLUDING COLUMN DEFAULTS;
```

Si desea que las columnas de tabla temporales declaradas hereden los valores predeterminados de las columnas de la tabla o vista que se nombra en fullselect, especifique la cláusula INCLUDING COLUMN DEFAULTS. Si desea que las columnas de tabla temporales declaradas tengan valores predeterminados que se correspondan con sus tipos de datos, especifique la cláusula USING TYPE DEFAULTS.

Db2 crea una instancia vacía de una tabla temporal declarada.

2. Complete una de las siguientes acciones:

- Rellene la tabla temporal declarada mediante sentencias INSERT.
- Modificar la tabla utilizando sentencias UPDATE o DELETE buscadas o posicionadas.
- Consulta la tabla utilizando sentencias SELECT.
- Crear índices en la tabla temporal declarada. La creación de un índice que especifique un grupo de almacenamiento o búfer diferente del grupo de almacenamiento o búfer de índice predeterminado de la base de datos del archivo de trabajo requiere privilegios de autorización USE adicionales para el grupo de almacenamiento o búfer.

3. Después de ejecutar una instrucción DECLARE GLOBAL TEMPORARY TABLE, la definición de la tabla temporal declarada existe mientras se ejecuta el proceso de aplicación. Si necesita eliminar la definición antes de que finalice el proceso de aplicación, emita una instrucción DROP TABLE. Por ejemplo, para eliminar la definición de TEMPPROD, ejecute la siguiente instrucción:

```
DROP TABLE SESSION.TEMPPROD;
```

Ejemplo

La cláusula ON COMMIT que especifique en la instrucción DECLARE GLOBAL TEMPORARY TABLE determina si Db2 mantiene o elimina todas las filas de la tabla cuando ejecuta una instrucción COMMIT en una aplicación con una tabla temporal declarada. ON COMMIT DELETE ROWS, que es el valor predeterminado, hace que se eliminan todas las filas de la tabla en un punto de confirmación, a menos que haya un cursor retenido abierto en la tabla en el punto de confirmación. ON COMMIT PRESERVE ROWS hace que las filas permanezcan más allá del punto de confirmación.

Por ejemplo, supongamos que ejecuta la siguiente instrucción en un programa de aplicación:

```
EXEC SQL DECLARE GLOBAL TEMPORARY TABLE TEMPPROD  
  AS (SELECT * FROM BASEPROD)  
  DEFINITION ONLY  
  INCLUDING IDENTITY COLUMN ATTRIBUTES  
  INCLUDING COLUMN DEFAULTS  
  ON COMMIT PRESERVE ROWS;  
EXEC SQL INSERT INTO SESSION.TEMPPROD SELECT * FROM BASEPROD;  
:  
EXEC SQL COMMIT;  
:
```

Cuando Db2 ejecuta la instrucción DECLARE GLOBAL TEMPORARY TABLE anterior, Db2 crea una instancia vacía de TEMPPROD. La sentencia INSERT rellena esa instancia con filas de la tabla BASEPROD. El calificador SESSION debe especificarse en cualquier declaración que haga referencia a TEMPPROD. Cuando Db2 ejecuta la instrucción COMMIT, Db2 mantiene todas las filas en TEMPPROD porque TEMPPROD se define con ON COMMIT PRESERVE ROWS. Cuando el programa finaliza, Db2 elimina TEMPPROD.

Referencia relacionada

[DECLARE GLOBAL TEMPORARY TABLE declaración \(Db2 SQL\)](#)

Suministro de clave única para una tabla

Si una tabla no tiene valores de columna únicos, puede proporcionar un identificador único utilizando columnas ROWID o columnas de identidad para almacenar valores únicos para cada fila de una tabla.

Procedimiento

Para proporcionar un identificador único para una tabla que aún no tiene una columna con valores únicos, utilice los siguientes enfoques.

- Añada una columna con el tipo de datos ROWID o una columna de identidad. Las columnas ROWID y las columnas de identidad contienen un valor único para cada fila de la tabla.
- Puede definir la columna como GENERATED ALWAYS, lo que significa que no puede insertar valores en la columna, o GENERATED BY DEFAULT, lo que significa que Db2 genera un valor si no especifica uno.
- Si define la columna ROWID o de identidad como GENERATED BY DEFAULT, también debe definir un índice único que incluya solo esa columna para garantizar la singularidad.

Conceptos relacionados

[Tipo de datos ROWID \(Introducción a Db2 para z/OS\)](#)

[Reglas para insertar datos en una columna de identidad](#)

Una columna de identidad contiene un valor numérico único para cada fila de la tabla. Si puede insertar datos en una columna de identidad y cómo se insertan esos datos depende de cómo se define la columna.

Fijación de tablas con definiciones incompletas

Si una tabla tiene una definición incompleta, no puede cargar la tabla, insertar datos, recuperar datos, actualizar datos ni suprimir datos. Sin embargo, puede descartar la tabla, crear el índice primario y descartar o crear otros índices.

Antes de empezar

Para comprobar si una tabla tiene una definición incompleta, mira la columna STATUS en SYSIBM.SYSTABLES. El valor I indica que la definición está incompleta.

Acerca de esta tarea

Una definición de tabla está incompleta en cualquiera de las siguientes circunstancias:

- **Si la tabla se define con una clave principal o única** y se cumplen todas las condiciones siguientes:
 - El espacio de la mesa para la mesa se creó explícitamente.
 - La declaración no se está ejecutando con el procesador de esquemas.
 - La tabla no tiene un índice principal o único para la clave principal o única definida.
- **Si la tabla tiene una columna ROWID que se define como generada por defecto** y se cumplen todas las condiciones siguientes:
 - El espacio de la mesa para la mesa se creó explícitamente.
 - El registro especial ESTABLECER NORMAS ACTUALES no está establecido en STD.
 - No se ha definido ningún índice único en la columna ROWID.
- **Si la tabla tiene una columna LOB** y se cumplen todas las condiciones siguientes:
 - El espacio de la mesa para la mesa se creó explícitamente.
 - El registro especial ESTABLECER NORMAS ACTUALES no está establecido en STD.
 - No se han definido todos los objetos LOB auxiliares para la columna LOB.

Procedimiento

Para completar la definición de una tabla, utilice una de las siguientes acciones:

- Cree un índice primario o modifique la tabla para eliminar la clave primaria.
- Cree un índice único en la clave única o modifique la tabla para eliminar la clave única.
- Definir un índice único en la columna ROWID.
- Cree los objetos LOB necesarios.

Ejemplo

Para crear el índice principal de la tabla de actividades del proyecto, emita la siguiente instrucción SQL:

```
CREATE UNIQUE INDEX XPROJAC1  
  ON DSN8C10.PROJACT (PROJNO, ACTNO, ACSTDATE);
```

RENOMBRAR TABLA en un escenario de mantenimiento de tablas

La instrucción RENAME TABLE es útil cuando necesitas desconectar temporalmente una tabla para realizar un mantenimiento que implique cambios estructurales en la tabla. Las aplicaciones pueden seguir ejecutándose en otra copia de la tabla hasta que se complete el mantenimiento.

Una forma de lograrlo es referirse al nombre de la tabla como un nombre no cualificado en todas las aplicaciones. El nombre de tabla sin cualificar está cualificado implícitamente por el contenido del registro especial CURRENT SCHEMA. Debe establecer CURRENT SCHEMA en el esquema de la tabla real para que las aplicaciones accedan a la tabla real. Antes de desconectar la tabla real, cambie el registro especial CURRENT SCHEMA al nombre del esquema de la copia alternativa de la tabla. Cuando todas las aplicaciones se ejecutan con la copia alternativa de la tabla, la tabla real puede modificarse. Un ejemplo de tal modificación es añadir una columna a la tabla.

Más tarde, cuando se complete el mantenimiento de la tabla, puede establecer el registro especial CURRENT SCHEMA con el nombre del esquema de la tabla real para que todas las aplicaciones vuelvan a utilizar la tabla real.

Referencia relacionada

[RENAME declaración \(Db2 SQL\)](#)

[ESQUEMA ACTUAL registro especial \(Db2 SQL\)](#)

Descarte de tablas

Cuando descarta una tabla, suprime los datos y la definición de tabla. También suprime todos los sinónimos, vistas, índices, restricciones referenciales y restricciones de comprobación asociadas con esa tabla.

Acerca de esta tarea

La siguiente instrucción SQL elimina la tabla YEMP:

```
DROP TABLE YEMP;
```

Utilice la instrucción DROP TABLE con cuidado: Eliminar una tabla **no equivale a** eliminar todas sus filas. Cuando se elimina una tabla, se pierden más que sus datos y su definición. Perdes todos los sinónimos, vistas, índices y restricciones referenciales y de verificación que están asociados con esa tabla. También perderá todas las autorizaciones que se le hayan concedido en la mesa.

Referencia relacionada

[DROP declaración \(Db2 SQL\)](#)

Definición de una vista

Una vista es una especificación con nombre de una tabla de resultados. Utilice las vistas para controlar qué usuarios tienen acceso a determinados datos o para simplificar la escritura de sentencias SQL.

Acerca de esta tarea

Utilice la instrucción CREATE VIEW para definir una vista y darle un nombre, tal como lo hace para una tabla. La vista que se crea con la siguiente instrucción muestra el nombre de cada director de departamento con los datos del departamento en el DSN8C10.DEPT tabla.

```
CREATE VIEW VDEPT AS  
  SELECT DEPTNO, MGRNO, LASTNAME, ADMRDEPT  
    FROM DSN8C10.DEPT, DSN8C10.EMP  
   WHERE DSN8C10.EMP.EMPNO = DSN8C10.DEPT.MGRNO;
```

Cuando un programa accede a los datos definidos por una vista, Db2 utiliza la definición de la vista para devolver un conjunto de filas a las que el programa puede acceder con sentencias SQL.

Para ver los departamentos que son administrados por el departamento D01 y los gerentes de esos departamentos, ejecute la siguiente declaración, que devuelve información de la vista VDEPTM:

```
SELECT DEPTNO, LASTNAME  
  FROM VDEPTM  
 WHERE ADMRDEPT = 'D01';
```

Cuando se crea una vista, se puede hacer referencia a los registros especiales SESSION_USER y CURRENT_SQLID en la instrucción CREATE VIEW. Al hacer referencia a la vista, Db2 utiliza el valor del registro especial SESSION_USER o CURRENT_SQLID que pertenece al usuario de la instrucción SQL (SELECT, UPDATE, INSERT o DELETE) en lugar del creador de la vista. En otras palabras, una referencia a un registro especial en una definición de vista se refiere a su valor de tiempo de ejecución.

Puede especificar un periodo concreto para una vista, sujeto a ciertas restricciones. Además, para una vista que haga referencia a una tabla temporal de período de aplicación o a una tabla bitemporal, puede especificar una cláusula de período para una operación de actualización o eliminación en la vista.

Una columna de una vista puede basarse en una columna de una tabla base que sea una columna de identidad. La columna en la vista también es una columna de identidad, **excepto** en cualquiera de las siguientes circunstancias:

- La columna aparece más de una vez en la vista.
- La vista se basa en una combinación de dos o más tablas.
- La vista se basa en la unión de dos o más tablas.
- Cualquier columna de la vista se deriva de una expresión que hace referencia a una columna de identidad.

Puede utilizar las vistas para limitar el acceso a ciertos tipos de datos, como la información salarial. De forma alternativa, puede utilizar la cláusula IMPLICITLY HIDDEN de una instrucción CREATE TABLE para definir una columna de una tabla que se ocultará de algunas operaciones.

También puede utilizar las vistas para las siguientes acciones:

- Hacer que un subconjunto de los datos de una tabla esté disponible para una aplicación. Por ejemplo, una vista basada en la tabla de empleados podría contener filas solo para un departamento en particular.
- Combinar columnas de dos o más tablas y poner los datos combinados a disposición de una aplicación. Mediante el uso de una instrucción SELECT que haga coincidir los valores de una tabla con los de otra, puede crear una vista que presente datos de ambas tablas. Sin embargo, **solo puede seleccionar** datos de este tipo de vista. **No puede actualizar, eliminar o insertar datos utilizando una vista que une dos o más tablas.**
- Combinar filas de dos o más tablas y poner los datos combinados a disposición de una aplicación. Al utilizar dos o más subselecciones que están conectadas por un operador de conjuntos como UNION, puede crear una vista que presente datos de varias tablas. Sin embargo, **solo puede seleccionar** datos de este tipo de vista. **No puede actualizar, eliminar o insertar datos utilizando una vista que contenga operaciones UNION.**
- Presentar datos computarizados y poner los datos resultantes a disposición de una aplicación. Puede calcular dichos datos utilizando cualquier función u operación que pueda utilizar en una instrucción SELECT.

Tareas relacionadas

[Cambio de datos mediante vistas que hacen referencia a tablas temporales \(Db2 Administration Guide\)](#)

Referencia relacionada

[CREATE VIEW declaración \(Db2 SQL\)](#)

Información relacionada

[Implementación de vistas de Db2 \(Db2 Administration Guide\)](#)

Vistas

Una vista no contiene datos; es una definición almacenada de un conjunto de filas y columnas. Una vista puede presentar todos o algunos de los datos en una o más tablas.

Aunque no puede modificar una vista existente, puede eliminarla y crear una nueva si sus tablas base cambian de una manera que afecte a la vista. Eliminar y crear vistas no afecta a las tablas base ni a sus datos.

Restricciones al cambiar datos a través de una vista

Algunas vistas son de solo lectura y, por lo tanto, no se pueden utilizar para actualizar los datos de la tabla. Para aquellas opiniones que se pueden actualizar, se aplican varias restricciones.

Tenga en cuenta las siguientes restricciones al cambiar datos a través de una vista:

- Debe tener la autorización adecuada para insertar, actualizar o eliminar filas utilizando la vista.
- Cuando se utiliza una vista para insertar una fila en una tabla, la definición de la vista debe especificar todas las columnas de la tabla base que no tienen un valor predeterminado. La fila que se está insertando debe contener un valor para cada una de esas columnas.
- Las vistas que puede utilizar para actualizar datos están sujetas a las mismas restricciones referenciales y de verificación que las tablas que utilizó para definir las vistas.

Puede utilizar la opción WITH CHECK de la sentencia CREATE VIEW para especificar la restricción de que cada fila que se inserte o actualice a través de la vista debe ajustarse a la definición de la vista. Puede seleccionar cada fila que se inserte o actualice a través de una vista que se cree con la opción WITH CHECK.

- Para una operación de actualización en una vista que hace referencia a una tabla temporal de período de aplicación o a una tabla bitemporal, la tabla de resultados de la fullselect externa de la definición de vista, explícita o implícitamente, debe incluir las columnas de inicio y fin del período BUSINESS_TIME.
- Para una operación de actualización o eliminación en una vista que haga referencia a una tabla temporal de período de aplicación o a una tabla bitemporal, la vista no debe definirse con un disparador INSTEAD OF.

Para vistas complejas, puede hacer posibles las operaciones de inserción, actualización y eliminación definiendo activadores INSTEAD OF.

Tareas relacionadas

[Inserción, actualización y supresión de datos en vistas utilizando desencadenantes INSTEAD OF](#)
 Los desencadenantes INSTEAD OF son desencadenantes que se ejecutan en lugar de la sentencia INSERT, UPDATE o DELETE que activa el desencadenante. Solamente se pueden definir estos desencadenantes para vistas. Utilice los desencadenantes INSTEAD OF para insertar, actualizar y suprimir datos en vistas complejas.

[Cambio de datos mediante vistas que hacen referencia a tablas temporales \(Db2 Administration Guide\)](#)

Referencia relacionada

[CREATE VIEW declaración \(Db2 SQL\)](#)

Dejar una opinión

Cuando sueltas una vista, también sueltas todas las vistas que están definidas en esa vista. La tabla base no se ve afectada.

Ejemplo

La siguiente instrucción SQL elimina la vista VDEPTM:

```
DROP VIEW VDEPTM;
```

Creación de una expresión de tabla común

Crear una expresión de tabla común le ahorra la sobrecarga de crear y eliminar una vista normal que solo necesita utilizar una vez. Además, durante la preparación del extracto, Db2 no necesita acceder al catálogo para la vista, lo que le ahorra gastos generales adicionales.

Acerca de esta tarea

Utilice la cláusula WITH para crear una expresión de tabla común.

Procedimiento

Para crear una expresión de tabla común, utilice uno de los siguientes enfoques:

- Especifique una cláusula WITH al principio de una instrucción SELECT.

Por ejemplo, la siguiente sentencia busca el departamento con la mayor remuneración total. La consulta implica dos niveles de agregación. En primer lugar, debe determinar el pago total de cada departamento utilizando la función SUM y ordenar los resultados utilizando la cláusula GROUP BY. A continuación, debe encontrar el departamento con el salario total más alto en función del salario total de cada departamento.

```
WITH DTOTAL (workdept, totalpay) AS
  (SELECT deptno, sum(salary+bonus)
   FROM DSN8810.EMP
   GROUP BY workdept)
SELECT workdept
  FROM DTOTAL
 WHERE totalpay = (SELECT max(totalpay)
                      FROM DTOTAL);
```

La tabla de resultados para la expresión de tabla común, DTOTAL, contiene el número de departamento y el salario total de cada departamento en la tabla de empleados. El fullselect del ejemplo anterior utiliza la tabla de resultados para DTOTAL para encontrar el departamento con la paga total más alta. La tabla de resultados de todo el extracto se parece a los siguientes resultados:

```
WORKDEPT
=====
D11
```

- Utilice expresiones de tabla comunes especificando WITH antes de un fullselect en una sentencia CREATE VIEW.

Esta técnica es útil si necesita utilizar los resultados de una expresión de tabla común en más de una consulta.

Por ejemplo, la siguiente declaración busca los departamentos que tienen una remuneración total superior a la media y guarda los resultados como la vista RICH_DEPT:

```
CREATE VIEW RICH_DEPT (workdept) AS
  WITH DTOTAL (workdept, totalpay) AS
    (SELECT workdept, sum(salary+bonus)
     FROM DSN8C10.EMP
     GROUP BY workdept)
  SELECT workdept
    FROM DTOTAL
   WHERE totalpay > (SELECT AVG(totalpay)
                      FROM DTOTAL);
```

El fullselect del ejemplo anterior utiliza la tabla de resultados para DTOTAL para encontrar los departamentos que tienen una remuneración total superior a la media. La tabla de resultados se guarda como la vista RICH_DEPT y tiene un aspecto similar a los siguientes resultados:

```
WORKDEPT
=====
A00
D11
D21
```

- Utilice expresiones de tabla comunes especificando WITH antes de un fullselect en una instrucción INSERT.

Por ejemplo, la siguiente declaración utiliza la tabla de resultados para VITALDEPT para encontrar el número de gerentes de cada departamento que tiene un número de ingenieros superiores superior a la media. A continuación, se inserta el número de cada gerente en la tabla vital_mgr.

```
INSERT INTO vital_mgr (mgrno)
  WITH VITALDEPT (workdept, se_count) AS
    (SELECT workdept, count(*)
     FROM DSN8C10.EMP
     WHERE job = 'senior engineer'
     GROUP BY workdept)
  SELECT d.manager
    FROM DSN8C10.DEPT d, VITALDEPT s
   WHERE d.workdept = s.workdept
     AND s.se_count > (SELECT AVG(se_count)
                        FROM VITALDEPT);
```

Referencia relacionada

[expresión-tabla-común \(Db2 SQL\)](#)

expresiones de tabla comunes

Una expresión de tabla común es como una vista temporal que se define y se utiliza durante la duración de una instrucción SQL.

Puede definir una expresión de tabla común siempre que pueda tener una sentencia fullselect. Por ejemplo, puede incluir una expresión de tabla común en una instrucción SELECT, INSERT, SELECT INTO o CREATE VIEW.

Cada expresión de tabla común debe tener un nombre único y definirse solo una vez. Sin embargo, puede hacer referencia a una expresión de tabla común muchas veces en la misma instrucción SQL. A diferencia de las vistas normales o de las expresiones de tabla anidadas, que obtienen sus tablas de resultados para cada referencia, todas las referencias a expresiones de tabla comunes en una instrucción determinada comparten la misma tabla de resultados.

Puede utilizar una expresión de tabla común en las siguientes situaciones:

- Cuando desee evitar crear una vista (cuando no se requiera el uso general de la vista y no se utilicen actualizaciones o eliminaciones posicionadas)
- Cuando la tabla de resultados se basa en variables del host
- Cuando la misma tabla resultante necesite compartirse en una selección completa
- Cuando los resultados deben derivarse mediante recursividad

Referencia relacionada

[expresión-tabla-común \(Db2 SQL\)](#)

Ejemplos de expresiones de tablas comunes recursivas

El SQL recursivo es muy útil en las aplicaciones BOM (Bill Of Materials).

Considere una tabla de piezas con subpiezas asociadas y la cantidad de subpiezas requeridas por cada pieza. Para obtener más información sobre SQL recursivo, consulte [“Creación de SQL recursivo mediante expresiones de tabla comunes”](#) en la página 410.

Para los ejemplos de este tema, cree la siguiente tabla:

```
CREATE TABLE PARTLIST
  (PART VARCHAR(8),
   SUBPART VARCHAR(8),
   QUANTITY INTEGER);
```

Suponga que la tabla PARTLIST se rellena con los valores que se encuentran en la siguiente tabla:

Tabla 38. Tabla de LISTA DE PARTES

PART	SUBPARTE	CANTIDAD
00	01	5
00	05	3
01	02	2
01	03	3
01	04	4
01	06	3
02	05	7
02	06	6

Tabla 38. Tabla de LISTA DE PARTES (continuación)

PART	SUBPARTE	CANTIDAD
03	07	6
04	08	10
04	09	11
05	10	10
05	11	10
06	12	10
06	13	10
07	14	8
07	12	8

Ejemplo 1: Explosión en un solo nivel:

La explosión de un solo nivel responde a la pregunta: «¿Qué partes se necesitan para construir la parte identificada con '01'?» La lista incluirá las subpiezas directas, subpiezas de subpiezas, etc. Sin embargo, si una pieza se utiliza varias veces, las subpiezas correspondientes sólo aparecerán en la lista una vez.

```
WITH RPL (PART, SUBPART, QUANTITY) AS
  (SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
   FROM PARTLIST ROOT
   WHERE ROOT.PART = '01'
  UNION ALL
   SELECT CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
   FROM RPL PARENT, PARTLIST CHILD
   WHERE PARENT.SUBPART = CHILD.PART)
SELECT DISTINCT PART, SUBPART, QUANTITY
  FROM RPL
 ORDER BY PART, SUBPART, QUANTITY;
```

La consulta anterior incluye una expresión de tabla común, identificada por el nombre RPL, que expresa la parte recursiva de esta consulta. Ilustra los elementos básicos de una expresión de tabla común recursiva.

El primer operando (fullselect) de la UNIÓN, denominado fullselect de inicialización, obtiene las subpartes directas de la parte «01». La cláusula FROM de este fullselect hace referencia a la tabla de origen y nunca se referirá a sí misma (RPL en este caso). El resultado de esta primera selección completa se introduce en la expresión de tabla común RPL. Como en este ejemplo, UNION debe ser siempre UNION ALL.

El segundo operando (fullselect) de la UNIÓN utiliza RPL para calcular subpartes de subpartes utilizando la cláusula FROM para referirse a la expresión de tabla común RPL y la tabla de origen PARTLIST con una unión de una parte de la tabla de origen (hijo) a una subparte del resultado actual contenido en RPL (padre). El resultado vuelve entonces de nuevo a RPL. El segundo operando de UNION se utiliza repetidamente hasta que no existan más subpartes.

SELECT DISTINCT de la selección completa principal de esta consulta, garantiza que no aparezca en la lista la misma pieza/subpieza más de una vez.

El resultado de la consulta se muestra en la siguiente tabla:

Tabla 39. Tabla de resultados para el ejemplo 1

PART	SUBPARTE	CANTIDAD
01	02	2
01	03	3

Tabla 39. Tabla de resultados para el ejemplo 1 (continuación)

PART	SUBPARTE	CANTIDAD
01	04	4
01	06	3
02	05	7
02	06	6
03	07	6
04	08	10
04	09	11
05	10	10
05	11	10
06	12	10
06	13	10
07	12	8
07	14	8

Observe en el resultado que la parte «01» contiene la subparte «02», que contiene la subparte «06», y así sucesivamente. Además, observe que la parte «06» se alcanza dos veces, una directamente a través de la parte «01» y otra a través de la parte «02». En el resultado, sin embargo, las subpartes de la parte «06» se enumeran solo una vez (esto es el resultado de usar un SELECT DISTINCT).

Recuerde que con expresiones de tabla comunes recursivas es posible introducir un bucle infinito. En este ejemplo, se crearía un bucle infinito si la condición de búsqueda del segundo operando que une las tablas padre e hija se codificara de la siguiente manera:

```
WHERE PARENT.SUBPART = CHILD.SUBPART
```

Este bucle infinito se crea al no codificar lo que se pretende. Debe determinar cuidadosamente qué codificar para que haya un final definido del ciclo de recursión.

El resultado producido por este ejemplo podría producirse en un programa de aplicación sin utilizar una expresión de tabla común recursiva. Sin embargo, dicha aplicación requeriría codificar una consulta diferente para cada nivel de recursividad. Además, la aplicación tendría que volver a introducir todos los resultados en la base de datos para ordenar el resultado final. Todo ello hace que la lógica de la aplicación se complique y que el funcionamiento no sea el esperado. La lógica de la aplicación se vuelve más difícil e inefficiente para otras consultas de listas de materiales, como las consultas de explosiones resumidas y con sangría.

Ejemplo 2: Explosión resumida:

Una explosión resumida responde a la pregunta: «¿Cuál es la cantidad total de cada pieza necesaria para construir la pieza '01'?» La principal diferencia con respecto a una explosión de un solo nivel es la necesidad de agregar las cantidades. Una explosión de un solo nivel indica la cantidad de subpartes necesarias para la pieza siempre que se requiera. No indica cuántos de cada subparte se necesitan para construir la parte '01'.

```
WITH RPL (PART, SUBPART, QUANTITY) AS
(
    SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
    FROM PARTLIST ROOT
    WHERE ROOT.PART = '01'
    UNION ALL
    SELECT PARENT.PART, CHILD.SUBPART,
```

```

        PARENT.QUANTITY*CHILD.QUANTITY
        FROM RPL PARENT, PARTLIST CHILD
        WHERE PARENT.SUBPART = CHILD.PART
    )
SELECT PART, SUBPART, SUM(QUANTITY) AS "Total QTY Used"
    FROM RPL
    GROUP BY PART, SUBPART
    ORDER BY PART, SUBPART;

```

En la consulta anterior, la lista de selección del segundo operando de la UNIÓN en la expresión de tabla común recursiva, identificada por el nombre RPL, muestra la agregación de la cantidad. Para determinar cuántos de cada subparte se utilizan, la cantidad del padre se multiplica por la cantidad por padre de un hijo. Si una pieza se utiliza varias veces en lugares diferentes, requerirá otra agregación final. Esto se hace agrupando las partes y subpartes en la expresión de tabla común RPL y utilizando la función de columna SUM en la lista de selección de la selección completa principal.

El resultado de la consulta se muestra en la siguiente tabla:

Tabla 40. Tabla de resultados para el ejemplo 2

PART	SUBPARTE	Cantidad total utilizada
01	02	2
01	03	3
01	04	4
01	05	14
01	06	15
01	07	18
01	08	40
01	09	44
01	10	140
01	11	140
01	12	294
01	13	150
01	14	144

Tenga en cuenta la cantidad total para la subparte '06'. El valor de 15 se deriva de una cantidad de 3 directamente para la parte '01' y una cantidad de 6 para la parte '02' que la parte '01' necesita dos veces.

Ejemplo 3: Profundidad de control:

Puede controlar la profundidad de una consulta recursiva para responder a la pregunta: «¿Cuáles son los dos primeros niveles de piezas que se necesitan para construir la pieza '01'?» Para mayor claridad en este ejemplo, el nivel de cada parte se incluye en la tabla de resultados.

```

WITH RPL (LEVEL, PART, SUBPART, QUANTITY) AS
(
    SELECT 1, ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
        FROM PARTLIST ROOT
        WHERE ROOT.PART = '01'
    UNION ALL
        SELECT PARENT.LEVEL+1, CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
        FROM RPL PARENT, PARTLIST CHILD
        WHERE PARENT.SUBPART = CHILD.PART
            AND PARENT.LEVEL < 2
)
SELECT PART, LEVEL, SUBPART, QUANTITY
    FROM RPL;

```

Esta consulta es similar a la consulta del ejemplo 1. Se introduce la columna NIVEL para contar el nivel de cada subparte con respecto a la parte original. En la inicialización fullselect, el valor de la columna LEVEL se inicializa a 1. En la subsiguiente selección completa, el nivel de la tabla principal se incrementa en 1. Para controlar el número de niveles en el resultado, la segunda fullselect incluye la condición de que el nivel del padre debe ser inferior a 2. Esto garantiza que la segunda selección completa sólo procesará hijos en el segundo nivel.

El resultado de la consulta se muestra en la siguiente tabla:

Tabla 41. Tabla de resultados para el ejemplo 3

PART	NIVEL	SUBPARTE	CANTIDAD
01	1	02	2
01	1	03	3
01	1	04	4
01	1	06	3
02	2	05	7
02	2	06	6
03	2	07	6
04	2	08	10
04	2	09	11
06	2	12	10
06	2	13	10

Creación de un desencadenante

Un disparador es un conjunto de instrucciones SQL que se ejecutan cuando se produce un determinado evento en una tabla o vista. Utilice activadores para controlar los cambios en bases de datos Db2. Los desencadenantes son más eficaces que las restricciones porque pueden supervisar un amplio rango de cambios y pueden realizar las acciones siguientes: En este tema se describe el soporte para desencadenantes avanzados.

Acerca de esta tarea

Uso de activadores para datos activos:

Por ejemplo, una restricción puede impedir una actualización en la columna de salario de la tabla de empleados si el nuevo valor supera una determinada cantidad. Un disparador puede controlar la cantidad en la que cambia el salario, así como el valor del salario. Si el cambio supera una determinada cantidad, el activador podría sustituir un valor válido y llamar a una función definida por el usuario para enviar un aviso a un administrador sobre la actualización no válida.

Los desencadenadores también trasladan la lógica de la aplicación a un Db2, lo que puede resultar en un desarrollo más rápido de la aplicación y un mantenimiento más sencillo. Por ejemplo, puede escribir aplicaciones para controlar los cambios salariales en la tabla de empleados, pero cada programa de aplicación que cambie la columna de salario debe incluir lógica para comprobar esos cambios. Un método mejor es definir un desencadenante que controle los cambios en la columna de salario. Db2 , a continuación, comprueba si hay alguna aplicación que modifique los salarios.

Ejemplo de creación y uso de un desencadenante:

De forma automática, los desencadenantes ejecutan un conjunto de sentencias de SQL cada vez que se produce un suceso especificado. Estas sentencias SQL pueden realizar tareas como la validación y edición

de cambios en tablas, la lectura y modificación de tablas, o la invocación de funciones o procedimientos almacenados que realizan operaciones tanto dentro como fuera de Db2.

Usted crea activadores utilizando la instrucción CREATE TRIGGER. La siguiente figura muestra un ejemplo de una instrucción CREATE TRIGGER.

```
1 CREATE TRIGGER REORDER
2   AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
3     REFERENCING NEW AS N_ROW
4
5       FOR EACH ROW
6         WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
7           BEGIN ATOMIC
8             CALL ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
9                           N_ROW.ON_HAND,
10                          N_ROW.PARTNO);
11          END
```

Las partes de este activador son:

- 1** Nombre del desencadenante (REORDENAR)
- 2** Hora de activación del disparador (DESPUÉS)
- 3** Evento desencadenante (ACTUALIZACIÓN)
- 4** Nombre de la tabla de sujetos (PARTES)
- 5** Nuevo nombre de correlación de variable de transición (N_ROW)
- 6** Granularidad (PARA CADA FILA)
- 7** Condición desencadenante (CUANDO...)
- 8** Cuerpo del disparador (BEGIN ATOMIC...FIN;)

Cuando ejecutas esta instrucción CREATE TRIGGER, Db2 crea un paquete de activación llamado REORDER y asocia el paquete de activación con la tabla PARTS. Db2 registra la marca de tiempo cuando crea el desencadenante. Si define otros desencadenantes en la tabla PARTS, Db2 utiliza esta marca de tiempo para determinar qué desencadenante activar primero cuando se produce el evento desencadenante. El disparador ya está listo para usar.

Después de que Db2 actualice las columnas ON_HAND o MAX_STOCKED en cualquier fila de la tabla PARTS, se activa el desencadenador REORDER. El desencadenante llama a un procedimiento almacenado llamado ISSUE_SHIP_REQUEST si, después de actualizar una fila, la cantidad de piezas disponibles es inferior al 10 % de la cantidad máxima almacenada. En la condición desencadenante, el calificador N_ROW representa un valor en una fila modificada después del evento desencadenante.

Cuando ya no deseé utilizar el desencadenador REORDER, puede eliminarlo ejecutando la instrucción:

```
DROP TRIGGER REORDER;
```

Al ejecutar esta sentencia se suprime el desencadenador REORDER y su paquete desencadenador asociado llamado REORDER.

Si se le caen las PIEZAS de la mesa, Db2 también se le cae el paquete de activación REORDER y su activador.

Partes de un gatillo:

Un disparador contiene las siguientes partes:

- Nombre de desencadenante
- tabla sujeto
- tiempo de activación de desencadenante
- suceso desencadenante
- granularidad
- nombres de correlación para variables de transición y tablas de transición
- acción desencadenada que consiste en una condición de búsqueda opcional y un cuerpo desencadenante

Nombre del activador:

Especifique un nombre para su activador. Puede utilizar un calificador o dejar que Db2 determine el calificador. Cuando Db2 crea un paquete desencadenante para el desencadenador, utiliza el mismo calificador que el ID de colección del paquete desencadenante.

Asunto de la tabla o vista:

Cuando realiza una operación de inserción, actualización o eliminación en esta tabla o vista, se activa el desencadenante. Debe nombrar una tabla local o una vista en la instrucción CREATE TRIGGER. No se puede definir un desencadenante en una tabla de catálogo.

Hora de activación del disparador:

Las opciones para el tiempo de activación del disparador son ANTES, DESPUÉS y EN LUGAR DE. Se pueden definir activadores ANTES y DESPUÉS para una tabla. EN VEZ DE se pueden definir activadores para una vista.

ANTES significa que el activador se activa antes de que Db2 realice cualquier cambio en la tabla de sujetos, y que la acción activada no activa ningún otro activador. AFTER significa que el activador se activa después de que Db2 realice cambios en la tabla de sujetos y pueda activar otros activadores. INSTEAD OF significa que el activador se activa cuando se intenta cambiar la vista del sujeto. Los disparadores con un tiempo de activación de ANTES se conocen como disparadores de antes. Los disparadores con un tiempo de activación DESPUÉS se conocen como disparadores posteriores. Los disparadores con un tiempo de activación de INSTEAD OF se conocen como disparadores en lugar de.

Evento desencadenante:

Cada desencadenante está asociado a un evento. Un disparador se activa cuando se produce el evento desencadenante en la tabla o vista del sujeto. El evento desencadenante es una de las siguientes operaciones SQL:

- insertar
- actualización
- suprimir

Un evento desencadenante también puede ser una operación de actualización o eliminación que se produce como resultado de una restricción referencial con ON DELETE SET NULL u ON DELETE CASCADE.

Un disparador puede activarse mediante una instrucción MERGE para operaciones de eliminación, inserción y actualización.

Los disparadores no se activan como resultado de las actualizaciones realizadas en las tablas por las utilidades de Db2 , con la excepción de la utilidad LOAD cuando se especifica con las opciones RESUME YES y SHRLEVEL CHANGE.

Cuando el evento desencadenante de un disparador es una operación de actualización, el disparador se denomina disparador de actualización. Del mismo modo, los desencadenantes de las operaciones de inserción se denominan desencadenantes de inserción, y los desencadenantes de las operaciones de eliminación se denominan desencadenantes de eliminación.

La instrucción SQL que realiza la operación SQL desencadenante se denomina instrucción SQL desencadenante. Cada evento desencadenante está asociado a una tabla o vista de sujeto y a una operación SQL.

El siguiente desencadenante se define con un evento desencadenante de inserción:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END
```

Si la operación SQL desencadenante es una operación de actualización, el evento puede asociarse a columnas específicas de la tabla en cuestión. En este caso, el activador se activa solo si la operación de actualización actualiza alguna de las columnas especificadas.

El siguiente desencadenante, PAYROLL1, que invoca la función definida por el usuario llamada PAYROLL_LOG, se activa solo si se realiza una operación de actualización en la columna SALARIO o BONIFICACIÓN de la tabla NÓMINA:

```
CREATE TRIGGER PAYROLL1
  AFTER UPDATE OF SALARY, BONUS ON PAYROLL
  FOR EACH STATEMENT
  BEGIN ATOMIC
    VALUES(PAYROLL_LOG(USER, 'UPDATE', CURRENT TIME, CURRENT DATE));
  END
```

Granularidad:

La instrucción SQL desencadenante podría modificar varias filas de la tabla. La granularidad del desencadenante determina si el desencadenante se activa solo una vez para la instrucción SQL desencadenante o una vez por cada fila que la instrucción SQL modifica. Los valores de granularidad son:

- FOR EACH ROW

El activador se activa una vez por cada fila que Db2 modifica en la tabla o vista de asunto. Si la instrucción SQL desencadenante no modifica ninguna fila, el desencadenante no se activa. Sin embargo, si la instrucción SQL desencadenante actualiza un valor en una fila al mismo valor, el desencadenante se activa. Por ejemplo, si se define un desencadenador UPDATE en la tabla COMPANY_STATS, la siguiente instrucción SQL activará el desencadenador.

```
UPDATE COMPANY_STATS SET NBEMP = NBEMP;
```

- FOR EACH STATEMENT

El activador se activa una vez cuando se ejecuta la instrucción SQL de activación. El disparador se activa incluso si la instrucción SQL desencadenante no modifica ninguna fila.

Los desencadenantes con una granularidad de PARA CADA FILA se conocen como desencadenantes de fila. Los activadores con una granularidad de PARA CADA ESTADO se conocen como activadores de estado. Los activadores de declaración solo pueden ser posteriores a los activadores.

La siguiente declaración es un ejemplo de un desencadenador de fila:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMP
  FOR EACH ROW
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END
```

El desencadenante NEW_HIRE se activa una vez por cada fila insertada en la tabla de empleados.

Variables de transición:

Cuando codifica un desencadenador de fila, es posible que necesite consultar los valores de las columnas en cada fila actualizada de la tabla o vista de asunto. Para ello, especifique un nombre de correlación

(que se utilizará al hacer referencia a las variables de transición) en la cláusula REFERENCING de su instrucción CREATE TRIGGER. Los dos tipos de variables de transición son:

- Las variables de transición antiguas capturan los valores de las columnas antes de que la instrucción SQL desencadenante los actualice. Puede utilizar la cláusula REFERENCING OLD para definir un nombre de correlación para hacer referencia a variables de transición antiguas para desencadenadores de actualización y eliminación.
- Las nuevas variables de transición capturan los valores de las columnas después de que la instrucción SQL desencadenante las actualice. Puede utilizar la cláusula REFERENCING NEW para definir un nombre de correlación para hacer referencia a nuevas variables de transición para desencadenadores de actualización e inserción.

Las variables de transición pueden referenciarse en cualquier parte de una instrucción SQL donde se pueda especificar una expresión o variable en los desencadenadores. Visite [Referencias a parámetros y variables SQL en SQL PL \(Db2 SQL\)](#) para obtener más información.

En el siguiente ejemplo se utilizan variables de transición e invocaciones de la función IDENTITY_VAL_LOCAL para acceder a valores que se asignan a columnas de identidad.

Supongamos que ha creado las tablas T y S, con las siguientes definiciones:

```
CREATE TABLE T
  (ID SMALLINT GENERATED BY DEFAULT AS IDENTITY (START WITH 100),
   C2 SMALLINT,
   C3 SMALLINT,
   C4 SMALLINT);

CREATE TABLE S
  (ID SMALLINT GENERATED ALWAYS AS IDENTITY,
   C1 SMALLINT);
```

Defina un desencadenante antes de insertar en T que utilice la función integrada IDENTITY_VAL_LOCAL para recuperar el valor actual de la columna de identidad ID, y utilice variables de transición para actualizar las demás columnas de T con el valor de la columna de identidad.

```
CREATE TRIGGER TR1
  NO CASCADE BEFORE INSERT
  ON T REFERENCING NEW AS N
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    SET N.C3 =N.ID;
    SET N.C4 =IDENTITY_VAL_LOCAL();
    SET N.ID =N.C2 *10;
    SET N.C2 =IDENTITY_VAL_LOCAL();
  END
```

Ahora suponga que ejecuta la siguiente instrucción INSERT:

```
INSERT INTO S (C1) VALUES (5);
```

Esta sentencia inserta una fila en S con un valor de 5 para la columna "C1" y un valor de 1 para la columna de identidad "ID". A continuación, suponga que ejecuta la siguiente instrucción SQL, que activa el desencadenador TR1:

```
INSERT INTO T (C2)
  VALUES (IDENTITY_VAL_LOCAL());
```

Esta declaración insertada, y la posterior activación del activador TR1, tienen los siguientes resultados:

- La sentencia INSERT obtiene el valor más reciente que se asignó a una columna de identidad (1) e inserta ese valor en la columna "C2" de la tabla T. 1 es el valor que Db2 insertó en la columna de identidad ID de la tabla S.
- Cuando se ejecuta la instrucción INSERT, Db2 inserta el valor 100 en la columna de identidad ID column de C2.

- La primera declaración en el cuerpo del trigger TR1 inserta el valor de la variable de transición N.ID (100) en la columna C3. N.ID es el valor que contiene la columna de identidad ID *después de que se ejecute la instrucción INSERT*.
- La segunda declaración en el cuerpo de la activación de la e TR1 a inserta el valor nulo en la columna de la e C4 a. Por definición, el resultado de la función IDENTITY_VAL_LOCAL en la acción desencadenada de un disparador antes de la inserción es el valor nulo.
- La tercera declaración en el cuerpo del disparador TR1 inserta 10 veces el valor de la variable de transición N.C2 (10*1) en la columna de identidad ID de la tabla T. N.C2 es el valor que contiene la columna C2 *después de que se ejecute INSERT*.
- C2 La cuarta declaración en el cuerpo de la activación de la e TR1 a inserta el valor nulo en la columna de la e a. Por definición, el resultado de la función IDENTITY_VAL_LOCAL en la acción desencadenada de un disparador antes de la inserción es el valor nulo.

Tablas de transición:

Si desea hacer referencia al conjunto completo de filas que modifica una sentencia SQL desencadenante, en lugar de filas individuales, utilice una tabla de transición. Al igual que las variables de transición, un nombre de correlación (para referirse a las columnas de la tabla de transición) puede aparecer en la cláusula REFERENCING de una instrucción CREATE TRIGGER. Los nombres de esas columnas son los mismos que el nombre de la columna en la tabla o vista para la que se define el desencadenador. Las tablas de transición son válidas tanto para los desencadenantes de fila como para los desencadenantes de sentencia. Los dos tipos de tablas de transición son:

- Las tablas de transición antiguas, especificadas con la cláusula OLD TABLE *transition-table-name*, capturan los valores de las columnas antes de que la instrucción SQL desencadenante los actualice. Puede definir tablas de transición antiguas para desencadenantes de actualización y eliminación.
- Las nuevas tablas de transición, especificadas con la cláusula NEW TABLE *transition-table-name*, capturan los valores de las columnas después de que la instrucción SQL desencadenante los actualice. Puede definir nuevas variables de transición para los desencadenantes de actualización e inserción.

El ámbito de los nombres de las tablas de transición antiguas y nuevas es el cuerpo desencadenante. Si se especifican nombres de correlación para las variables de transición antiguas y nuevas en el desencadenador, una referencia a una variable de transición debe estar calificada con el nombre de correlación asociado. El nombre de una variable de transición también puede ser el mismo que el nombre de una variable SQL o variable global, o el nombre de una columna en una tabla o vista a la que se hace referencia en el desencadenador. Los nombres idénticos se deben calificar explícitamente. La calificación de un nombre puede aclarar si el nombre se refiere a una columna, variable global, variable SQL, parámetro SQL o variable de transición. Para evitar ambigüedades, califique una variable de transición con el nombre de correlación especificado en la cláusula REFERENCING en la instrucción CREATE TRIGGER o ALTER TRIGGER que definió el disparador.

El siguiente ejemplo accede a una nueva tabla de transición para capturar el conjunto de filas que se insertan en la tabla INVOICE:

```
CREATE TRIGGER LRG_ORDER
AFTER INSERT ON INVOICE
REFERENCING NEW TABLE AS N_TABLE
FOR EACH STATEMENT
BEGIN ATOMIC
  SELECT LARGE_ORDER_ALERT(CUST_NO,
    TOTAL_PRICE, DELIVERY_DATE)
  FROM N_TABLE WHERE TOTAL_PRICE > 10000;
END
```

La sentencia SELECT en LRG_ORDER hace que la función definida por el usuario LARGE_ORDER_ALERT se ejecute para cada fila de la tabla de transición N_TABLE que satisfaga la cláusula WHERE (TOTAL_PRICE > 10000).

Acción desencadenada:

Cuando se activa un desencadenante, se produce una acción desencadenada. Cada activador tiene una acción desencadenada, que consiste en una condición de activación opcional y un cuerpo de activación.

Condición desencadenante:

Si desea que la acción activada se produzca solo cuando se cumplan ciertas condiciones, codifique una condición de activación. Una condición de activación es similar a un predicado en un SELECT, excepto que la condición de activación comienza con WHEN, en lugar de WHERE. Si no incluye una condición desencadenante en su acción desencadenada, el cuerpo del desencadenante se ejecuta cada vez que se activa el desencadenante.

Para un desencadenante de fila, Db2 evalúa la condición de desencadenante una vez por cada fila modificada de la tabla en cuestión. Para un disparador de sentencia, Db2 evalúa la condición de disparo una vez por cada ejecución de la sentencia SQL disparadora.

Si la condición desencadenante de un before trigger tiene un fullselect, el fullselect no puede hacer referencia a la tabla de sujetos.

El siguiente ejemplo muestra una condición de activación que hace que el cuerpo de activación se ejecute solo cuando el número de artículos pedidos es mayor que el número de artículos disponibles:

```
CREATE TRIGGER CK_AVAIL
  BEFORE INSERT ON ORDERS
  REFERENCING NEW AS NEW_ORDER
  FOR EACH ROW
  WHEN (NEW_ORDER.QUANTITY >
    (SELECT ON_HAND FROM PARTS
     WHERE NEW_ORDER.PARTNO=PARTS.PARTNO))
  BEGIN ATOMIC
    VALUES(ORDER_ERROR(NEW_ORDER.PARTNO,
      NEW_ORDER.QUANTITY));
  END
```

Cuerpo del gatillo:

En el cuerpo del disparador, se codifican las sentencias SQL que se desean ejecutar siempre que la condición del disparador sea verdadera. El cuerpo del disparador puede incluir una única *sentencia SQL-control-statement*, incluida una sentencia compuesta o una *sentencia SQL disparada* que se ejecutará para la *acción disparada*. Las declaraciones que puede utilizar en un cuerpo de activador dependen del tiempo de activación del activador. Visite [Instrucción CREATE TRIGGER \(disparador avanzado\) \(Db2 SQL\)](#) y [Lenguaje de procedimiento de SQL \(SQL PL\) \(Db2 SQL\)](#) para obtener más información sobre la definición de desencadenadores SQL. Utilice instrucciones de control para desarrollar activadores que contengan lógica.

Debido a que puede incluir declaraciones INSERT, DELETE, UPDATE y MERGE en el cuerpo del disparador, la ejecución del cuerpo del disparador puede provocar la activación de otros disparadores. Consulte “[Desencadenante en cascada](#)” en la página 173 para obtener más información.

ejemplos

Ejemplo 1

Defina un desencadenante para incrementar el recuento de empleados cuando se contrate a un nuevo empleado. El siguiente ejemplo también explica cómo determinar por qué se permite una instrucción SQL en el desencadenador.

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    1
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END
```

La instrucción UPDATE (1) es una instrucción SQL que está permitida porque aparece en el diagrama de sintaxis para la *instrucción SQL desencadenada*.

Ejemplo 2

Defina un desencadenante para devolver una condición de error y deshacer cualquier cambio realizado por el desencadenante, así como las acciones que resulten de las restricciones referenciales en la tabla de sujetos. Utilice la instrucción SIGNAL para indicar la información de error que se debe

devolver. Cuando Db2 ejecuta la instrucción SIGNAL, devuelve un SQLCA a la aplicación con SQLCODE -438. El SQLCA también incluye los siguientes valores, que usted suministra en la instrucción SIGNAL:

- Un valor de 5 caracteres que utiliza Db2 como SQLSTATE
- Un mensaje de error que Db2 coloca en el campo SQLERRMC

En el siguiente ejemplo, la sentencia SIGNAL hace que Db2 devuelva un SQLCA con SQLSTATE 75001 y finalice la operación de actualización salarial si el aumento de salario de un empleado es superior al 20 %:

```
CREATE TRIGGER SAL_ADJ
  BEFORE UPDATE OF SALARY ON EMP
  REFERENCING OLD AS OLD_EMP
  NEW AS NEW_EMP
  FOR EACH ROW
  WHEN (NEW_EMP.SALARY > (OLD_EMP.SALARY * 1.20))
  BEGIN ATOMIC
    SIGNAL SQLSTATE '75001'
      ('Invalid Salary Increase - Exceeds 20%');
  END
```

Ejemplo 3

Definir un desencadenante para asignar la fecha actual a la columna HIRE_DATE cuando se inserta una fila en la tabla EMP. Dado que los disparadores before operan en filas de una tabla antes de que esas filas se modifiquen, no se pueden realizar operaciones en el cuerpo de un disparador before que modifiquen directamente la tabla en cuestión. Sin embargo, puede utilizar la sentencia *transition-variable-assignment-statement* para modificar los valores de una fila antes de que esos valores entren en la tabla. Por ejemplo, este desencadenante utiliza una nueva variable de transición (NEW_VAR.HIRE_DATE) para asignar la fecha de hoy como fecha de contratación del nuevo empleado:

```
CREATE TRIGGER HIREDATE
  NO CASCADE BEFORE INSERT ON EMP
  REFERENCING NEW AS NEW_VAR
  FOR EACH ROW
  BEGIN ATOMIC
    SET NEW_VAR.HIRE_DATE = CURRENT_DATE;
  END
```

Ejemplo 4

En el siguiente ejemplo, la tabla CLASS_SCHED contiene una fila para el horario de cada clase en una escuela. Cuando se añade una fila de horario de clase a la tabla, se activa el desencadenador VALIDATE_SCHED. En el desencadenador, se utilizan instrucciones de control SQL para comprobar y responder a los siguientes errores en las horas de inicio y finalización de la clase:

Tipo de error

La hora de finalización es nula

Respuesta

Establecer la hora de finalización una hora después de la hora de inicio

La hora de finalización es posterior a las 9:00 p.m.

Mostrar un mensaje de error

El día de inicio es un fin de semana

Mostrar un mensaje de error

```
CREATE TRIGGER VALIDATE_SCHED
  BEFORE INSERT ON CLASS_SCHED
  REFERENCING NEW AS N
  FOR EACH ROW
  VS: BEGIN

    IF (N.ENDING IS NULL) THEN [1]
      SET N.ENDING = N.STARTING + 1 HOUR; [3]
    END IF;
    IF (N.ENDING > '21:00') THEN [1]
      SIGNAL SQLSTATE '80000' SET MESSAGE_TEXT = [2]
        'CLASS ENDING TIME IS AFTER 9 PM';
    ELSEIF (N.DAY=1 OR N.DAY=7) THEN
      SIGNAL SQLSTATE '80001' SET MESSAGE_TEXT = [2]
```

```
'CLASS CANNOT BE SCHEDULED ON A WEEKEND';
END IF;
END VS
```

El desencadenador SQL tiene las siguientes instrucciones:

- Las sentencias IF (1) y las sentencias SIGNAL (2) son instrucciones de control SQL.
- La declaración de asignación SET (3) es una instrucción de control SQL que asigna valores a variables.

Tareas relacionadas

Ocultación del código fuente de procedimientos SQL, funciones SQL y desencadenantes (Db2 Administration Guide)

Referencia relacionada

[Instrucción CREATE TRIGGER \(disparador avanzado\) \(Db2 SQL\)](#)

[Instrucción CREATE TRIGGER \(disparador básico\) \(Db2 SQL\)](#)

[LOAD \(Db2 Utilities\)](#)

Invocar un procedimiento almacenado o una función definida por el usuario desde un desencadenador

Un cuerpo de activación solo puede incluir sentencias SQL. Para realizar acciones o utilizar lógica que no está disponible en las sentencias SQL, cree funciones definidas por el usuario o procedimientos almacenados. A continuación, invóquelos desde el interior del cuerpo del activador.

Acerca de esta tarea

Conceptos introductorios

[Desencadenantes \(Introducción a DB2 para z/OS\)](#)

Restricción: No puede incluir instrucciones INSERT, UPDATE, DELETE o MERGE en procedimientos almacenados o funciones definidas por el usuario que sean invocadas por un BEFORE TRIGGER. Estas acciones no están permitidas, porque los desencadenadores BEFORE no deben modificar ninguna tabla.

Procedimiento

Para invocar un procedimiento almacenado o una función definida por el usuario desde un desencadenador:

1. Asegúrese de que el procedimiento almacenado o la función definida por el usuario se definan antes de definir el desencadenante.
 - Defina procedimientos utilizando la instrucción CREATE PROCEDURE.
 - Defina los desencadenantes utilizando la instrucción CREATE FUNCTION.
2. Invoque la función definida por el usuario o el procedimiento almacenado realizando una de las siguientes acciones:
 - Para invocar una función definida por el usuario, incluya la función definida por el usuario en una de las siguientes declaraciones en el desencadenador:

Sentencia SELECT

Utilice una instrucción SELECT para ejecutar la función de forma condicional. El número de veces que se ejecuta la función definida por el usuario depende del número de filas de la tabla de resultados de la instrucción SELECT. Por ejemplo, en el siguiente desencadenante, la instrucción SELECT invoca la función definida por el usuario LARGE_ORDER_ALERT. Esta función se ejecuta una vez por cada fila de la tabla de transición N_TABLE con un precio de orden superior a 10 000:

```
CREATE TRIGGER LRG_ORDR
AFTER INSERT ON INVOICE
REFERENCING NEW TABLE AS N_TABLE
FOR EACH STATEMENT MODE DB2SQL
BEGIN ATOMIC
```

```
SELECT LARGE_ORDER_ALERT(CUST_NO, TOTAL_PRICE, DELIVERY_DATE)
      FROM N_TABLE WHERE TOTAL_PRICE > 10000;
END
```

Sentencia VALUES

Utilice la instrucción VALUES para ejecutar una función de forma incondicional. La función se ejecuta una vez por cada ejecución de un disparador de sentencia o una vez por cada fila en un disparador de fila. En el siguiente ejemplo, la función definida por el usuario PAYROLL_LOG se ejecuta cada vez que se activa el desencadenador PAYROLL1. Este activador se activa cuando se produce una operación de actualización.

```
CREATE TRIGGER PAYROLL1
  AFTER UPDATE ON PAYROLL
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES(PAYROLL_LOG(USER, 'UPDATE',
                        CURRENT TIME, CURRENT DATE));
  END
```

]

- Para invocar un procedimiento almacenado, incluya una instrucción CALL en el desencadenador. Los parámetros de esta llamada de procedimiento almacenado deben ser constantes, variables de transición, localizadores de tabla o expresiones.
Si el parámetro es una variable de transición o un localizador de tabla, y la sentencia CALL está en un disparador BEFORE o AFTER, Db2 devuelve una advertencia.
- 3. Para pasar tablas de transición del desencadenador a la función definida por el usuario o al procedimiento almacenado, utilice localizadores de tablas.

Cuando se llama a una función definida por el usuario o a un procedimiento almacenado desde un desencadenador, es posible que se quiera dar a la función o al procedimiento acceso a todo el conjunto de filas modificadas. En este caso, utilice localizadores de tablas para pasar un puntero a la tabla de transición antigua o nueva.

La mayor parte del código para usar un localizador de tabla está en la función o procedimiento almacenado que recibe el localizador.

Para pasar la tabla de transición desde un desencadenador, especifique el parámetro TABLE *transition-table-name* cuando invoque la función o el procedimiento almacenado. Este parámetro hace que Db2 pase un localizador de tabla para la tabla de transición a la función definida por el usuario o al procedimiento almacenado. Por ejemplo, el siguiente desencadenador pasa un localizador de tabla para una tabla de transición NEWEMPS al procedimiento almacenado CHECKEMP:

```
CREATE TRIGGER EMPRAISE
  AFTER UPDATE ON EMP
  REFERENCING NEW TABLE AS NEWEMPS
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    CALL CHECKEMP(TABLE NEWEMPS);
  END
```

Conceptos relacionados

[Pasos para crear y utilizar una función definida por el usuario](#)

Una función definida por el usuario es similar a un subprograma o función del lenguaje principal. No obstante, una función definida por el usuario a menudo es la mejor opción para una aplicación de SQL ya que se puede invocar en una sentencia de SQL.

[Tareas relacionadas](#)

Acceso a tablas de transición en una función o un procedimiento almacenado definido por el usuario

Si desea hacer referencia al conjunto completo de filas que modifica una sentencia SQL desencadenante, en lugar de filas individuales, utilice una tabla de transición. Puede hacer referencia a una tabla de transición en funciones y procedimientos definidos por el usuario que se invocan desde un desencadenante.

[Creación de procedimientos almacenados](#)

Un archivo de extensión. *procedimiento almacenado*, es un código ejecutable que puede ser llamado por otros programas. El proceso de creación depende del tipo de procedimiento.

Creación de una función definida por el usuario

Puede ampliar la funcionalidad SQL de Db2 añadiendo sus propias definiciones de proveedor o de terceros.

Referencia relacionada

[CALL declaración \(Db2 SQL\)](#)

[Instrucción CREATE FUNCTION \(resumen\) \(Db2 SQL\)](#)

[Instrucción CREATE PROCEDURE \(resumen\) \(Db2 SQL\)](#)

[sentencia-select \(Db2 SQL\)](#)

[VALUES declaración \(Db2 SQL\)](#)

Inserción, actualización y supresión de datos en vistas utilizando desencadenantes INSTEAD OF

Los desencadenantes INSTEAD OF son desencadenantes que se ejecutan en lugar de la sentencia INSERT, UPDATE o DELETE que activa el desencadenante. Solamente se pueden definir estos desencadenantes para vistas. Utilice los desencadenantes INSTEAD OF para insertar, actualizar y suprimir datos en vistas complejas.

Acerca de esta tarea

Las vistas complejas son aquellas vistas que se definen en expresiones o en varias tablas. En algunos casos, esas opiniones son de solo lectura. En estos casos, los activadores INSTEAD OF hacen posibles las operaciones de inserción, actualización y eliminación. Si la vista compleja no es de solo lectura, puede solicitar una operación de inserción, actualización o eliminación. Sin embargo, Db2 decide automáticamente cómo realizar esa operación en las tablas base a las que se hace referencia en la vista. Con los desencadenadores INSTEAD OF, puede definir exactamente cómo debe ejecutarse una operación de inserción, actualización o eliminación en la vista (Db2). Ya no dejas la decisión a Db2.

Procedimiento

Para insertar, actualizar o eliminar datos en una vista utilizando activadores INSTEAD OF:

- Defina uno o más activadores INSTEAD OF en la vista utilizando una instrucción CREATE TRIGGER.

Puede crear un desencadenante para cada una de las siguientes operaciones: INSERT, UPDATE y DELETE. Estos desencadenantes definen la acción que debe realizar Db2 para cada una de estas operaciones.

- Enviar una declaración de INSERTAR, ACTUALIZAR o ELIMINAR en la vista.

Db2 ejecuta el desencadenante apropiado INSTEAD OF.

Ejemplo

Supongamos que crea la siguiente vista en las tablas de muestra DSN8C10.EMP y DSN8C10.DEPT:

```
CREATE VIEW EMPV (EMPNO, FIRSTNAME, MIDINIT, LASTNAME, PHONENO, HIREDATE, DEPTNAME)
AS SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME, PHONENO, HIREDATE, DEPTNAME
      FROM DSN8C10.EMP, DSN8C10.DEPT WHERE DSN8C10.EMP.WORKDEPT
                                         = DSN8C10.DEPT.DEPTNO
```

Supongamos que también define los siguientes tres desencadenantes INSTEAD OF:

```
CREATE TRIGGER EMPV_INSERT INSTEAD OF INSERT ON EMPV
REFERENCING NEW AS NEWEMP
FOR EACH ROW MODE DB2SQL
  INSERT INTO DSN8C10.EMP (EMPNO, FIRSTNAME, MIDINIT, LASTNAME, WORKDEPT,
                           PHONENO, HIREDATE)
    VALUES(NEWEMP.EMPNO, NEWEMP.FIRSTNAME, NEWEMP.MIDINIT, NEWEMP.LASTNAME,
           COALESCE((SELECT D.DEPTNO FROM DSN8C10.DEPT AS D
```

```

        WHERE D.DEPTNAME = NEWEMP.DEPTNAME),
        RAISE_ERROR('70001', 'Unknown department name')),
        NEWEMP.PHONENO, NEWEMP.HIREDATE)

CREATE TRIGGER EMPV_UPDATE INSTEAD OF UPDATE ON EMPV
REFERENCING NEW AS NEWEMP OLD AS OLDEMP
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    UPDATE DSN8C10.EMP AS E
    SET (E.FIRSTNME, E.MIDINIT, E.LASTNAME, E.WORKDEPT, E.PHONENO,
        E.HIREDATE)
    = (NEWEMP.FIRSTNME, NEWEMP.MIDINIT, NEWEMP.LASTNAME,
        COALESCE((SELECT D.DEPTNO FROM DSN8C10.DEPT AS D
        WHERE D.DEPTNAME = OLDEMP.DEPTNAME),
        RAISE_ERROR ('70001', 'Unknown department name'))
        NEWEMP.PHONENO, NEWEMP.HIREDATE)
    WHERE NEWEMP.EMPNO = E.EMPNO;
    UPDATE DSN8C10.DEPT D SET D.DEPTNAME=NEWEMP.DEPTNAME
    WHERE D.DEPTNAME=OLDEMP.DEPTNAME;
END

CREATE TRIGGER EMPV_DELETE INSTEAD OF DELETE ON EMPV
REFERENCING OLD AS OLDEMP
FOR EACH ROW MODE DB2SQL
    DELETE FROM DSN8C10.EMP AS E WHERE E.EMPNO = OLDEMP.EMPNO

```

Debido a que la vista está en una consulta con una combinación interna, la vista es de solo lectura. Sin embargo, los activadores INSTEAD OF hacen posibles las operaciones de inserción, actualización y eliminación.

La siguiente tabla describe lo que ocurre con las distintas operaciones de inserción, actualización y eliminación en la vista EMPV.

Tabla 42. Resultados de los desencadenantes INSTEAD OF

Sentencia SQL	Resultado
INSERT INTO EMPV VALUES (...)	El activador EMPV_INSERT está activado. Este activador inserta la fila en la tabla base DSN8C10.EMP si el nombre del departamento coincide con un valor de la columna WORKDEPT de la tabla DSN8C10.DEPT. De lo contrario, se emite un error. Si se hubiera utilizado una consulta en lugar de una cláusula VALUES en la instrucción INSERT, el cuerpo del desencadenador se procesaría para cada fila de la consulta.
UPDATE EMPV SET DEPTNAME='PLANNING & STRATEGY' WHERE DEPTNAME='PLANNING'	Se activa el desencadenante EMPV_UPDATE. Este desencadenador actualiza la columna NOMDEPARTAMENTO en el archivo de datos de la tienda (DSN8C10).DEPT para las filas que cumplen los requisitos.
DELETE FROM EMPV WHERE HIREDATE<'1910-01-01'	Se activa el desencadenante EMPV_DELETE. Este desencadenante elimina las filas que cumplen los requisitos de la tabla.EMP de DSN8C10.

Referencia relacionada

[Instrucción CREATE TRIGGER \(disparador básico\) \(Db2 SQL\)](#)

Errores encontrados en un desencadenante

Una instrucción SQL en el cuerpo de un disparador puede fallar durante la ejecución del disparador, provocando un error.

Suponiendo que no se definen manipuladores en el desencadenador, si alguna instrucción SQL en el cuerpo del desencadenador falla durante la ejecución del desencadenador, Db2 revierte todos los cambios realizados por la instrucción SQL desencadenante y las instrucciones SQL desencadenadas. Sin

embargo, si el cuerpo del desencadenador ejecuta acciones que están fuera del control de Db2, o no están bajo la misma coordinación de compromiso que el subsistema de Db2 en el que se ejecuta el desencadenador, Db2 no puede deshacer esas acciones. Ejemplos de acciones externas que no están bajo el control de Db2 son:

- Realizar actualizaciones que no están bajo el control de RRS
- Envío de un mensaje de correo electrónico

Si el desencadenador ejecuta acciones externas que están bajo la misma coordinación de compromiso que el subsistema de Db2 bajo el cual se ejecuta el desencadenador, y se produce un error durante la ejecución del desencadenador, Db2 coloca el proceso de aplicación que emitió la declaración desencadenante en un estado de reversión obligatoria. La aplicación debe entonces ejecutar una operación de reversión para revertir esas acciones externas. Ejemplos de acciones externas que están bajo la misma coordinación de compromiso que la operación SQL desencadenante son:

- Ejecución de una operación de actualización distribuida
- Desde una función definida por el usuario o un procedimiento almacenado, ejecutar una acción externa que afecte a un gestor de recursos externo que esté bajo el control de RRS.

Paquetes desencadenantes

Un paquete de activación es un tipo especial de paquete que se crea solo cuando ejecuta una instrucción CREATE TRIGGER. Un paquete desencadenante se ejecuta sólo cuando se activa el desencadenante asociado.

Al igual que con cualquier otro paquete, Db2 marca un paquete desencadenante como no válido cuando se elimina una tabla, índice o vista de la que depende el paquete desencadenante. Db2 ejecuta una reasignación automática la próxima vez que se active el disparador. Sin embargo, si la reasociación automática falla, Db2 no marca el paquete de activación como inoperativo.

A diferencia de otros paquetes, un paquete de activación se libera si se elimina la tabla en la que está definido el activador, por lo que solo se puede volver a crear el paquete de activación volviendo a crear la tabla y el activador.

Db2 admite activadores básicos y avanzados. Usted utiliza un subcomando REBIND diferente para cada tipo.

- Para los paquetes de activación básicos, utilice el subcomando REBIND TRIGGER PACKAGE. También puede utilizar REBIND TRIGGER PACKAGE para cambiar un subconjunto limitado de las opciones de enlace predeterminadas que Db2 utilizó al crear el paquete. Puede identificar desencadenantes básicos consultando la tabla de catálogo SYSIBM.SYSTRIGGERS. Los valores en blanco de la columna SQLPL identifican desencadenantes básicos.
- Para paquetes de activación avanzados, utilice el subcomando PAQUETE DE REENLACE. Puede utilizar la instrucción ALTER TRIGGER para cambiar los valores de opción con los que Db2 vinculó originalmente el paquete de activación. Puede identificar desencadenantes avanzados consultando la tabla de catálogo SYSIBM.SYSTRIGGERS. Los valores 'Y' en la columna SQLPL identifican desencadenantes avanzados.

Si emite un comando REBIND PACKAGE contra un paquete para un activador avanzado, las únicas opciones de enlace que puede cambiar son EXPLAIN , pero EXPLAIN(ONLY) no se acepta, FLAG, PLANMGMT y CONCENTRATESTMT. Si intenta cambiar otras opciones de enlace, el mandato no se ejecutará correctamente y devolverá el mensaje DSNT215I.

Referencia relacionada

[REBIND TRIGGER PACKAGE \(Db2 \)](#)

[REBIND PACKAGE subcomando \(DSN\) \(Comandos de Db2 \)](#)

Desencadenante en cascada

Cuando un desencadenante realiza una operación SQL, puede modificar la tabla de asunto u otras tablas con desencadenantes, por lo tanto, Db2 también activa los desencadenantes. Esta situación se conoce como cascada de activador.

Un desencadenante que se activa como resultado de otro desencadenante puede activarse al mismo nivel que el desencadenante original o a un nivel diferente. Dos activadores, A y B, se activan en diferentes niveles si el activador B se activa después de que el activador A se active y se completa antes de que el activador A se complete. Si el activador B se activa después de que el activador A se active y se completa después de que el activador A se complete, entonces los activadores están al mismo nivel.

Por ejemplo, en estos casos, el activador A y el activador B se activan al mismo nivel:

- La tabla X tiene dos desencadenantes definidos en ella, A y B. A es un desencadenante anterior y B es un desencadenante posterior. Una actualización de la tabla X hace que se activen tanto el activador A como el activador B.
- El desencadenador A actualiza la tabla X, que tiene una restricción referencial con la tabla Y, que tiene definido el desencadenador B. La restricción referencial hace que se actualice la tabla Y, lo que activa el desencadenador B.

En estos casos, el activador A y el activador B se activan a diferentes niveles:

- El activador A se define en la tabla X y el activador B se define en la tabla Y. El desencadenante B es un desencadenante de actualización. Una actualización de la tabla X activa el desencadenador A, que contiene una instrucción UPDATE en la tabla Y en su cuerpo desencadenador. Esta instrucción UPDATE activa el desencadenador B.
- El desencadenador A llama a un procedimiento almacenado. El procedimiento almacenado contiene una instrucción INSERT para la tabla X, que tiene definido el desencadenador de inserción B. Cuando se ejecuta la sentencia INSERT en la tabla X, se activa el desencadenador B.

Cuando los activadores se activan en diferentes niveles, se denomina *activación en cascada*. La activación en cascada solo puede producirse para los activadores posteriores, ya que Db2 no admite la activación en cascada de los activadores anteriores.

Para evitar la posibilidad de una cascada de disparadores interminable, Db2 solo admite 16 niveles de cascada de disparadores, procedimientos almacenados y funciones definidas por el usuario. Si se activa un desencadenante, una función definida por el usuario o un procedimiento almacenado en el nivel de 17th, Db2 devuelve SQLCODE -724 y deshace todos los cambios SQL en los 16 niveles de cascada. Sin embargo, al igual que con cualquier otro error SQL que se produzca durante la ejecución del desencadenador, si se produce alguna acción que esté fuera del control de Db2, dicha acción no se deshace.

Puede escribir un programa de monitorización que emita solicitudes IFI READS para recopilar información de seguimiento de Db2 sobre los niveles de activación en cascada, funciones definidas por el usuario y procedimientos almacenados en sus programas.

Tareas relacionadas

[Invocación de IFI desde un programa de supervisor \(Db2 Performance\)](#)

Orden de activación de varios desencadenantes

Puede crear varios desencadenantes para la misma tabla de asunto, suceso y hora de activación. El orden en el que se activan estos desencadenantes es el orden en el que se crearon los desencadenantes.

Db2 registra la marca de tiempo cuando se ejecuta cada instrucción CREATE TRIGGER. Cuando se produce un evento en una tabla que activa más de un desencadenante, Db2 utiliza las marcas de tiempo almacenadas para determinar qué desencadenante activar primero.

Db2 siempre se activan todos los desencadenantes anteriores que están definidos en una tabla antes de los desencadenantes posteriores que están definidos en esa tabla, pero dentro del conjunto de

desencadenantes anteriores, el orden de activación es por marca de tiempo, y dentro del conjunto de desencadenantes posteriores, el orden de activación es por marca de tiempo.

En este ejemplo, los desencadenantes NEWHIRE1 y NEWHIRE2 tienen el mismo evento desencadenante (INSERT), la misma tabla de sujetos (EMP) y el mismo tiempo de activación (AFTER). Supongamos que la instrucción CREATE TRIGGER para NEWHIRE1 se ejecuta antes de la instrucción CREATE TRIGGER para NEWHIRE2:

```
CREATE TRIGGER NEWHIRE1
  AFTER INSERT ON EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
  END

CREATE TRIGGER NEWHIRE2
  AFTER INSERT ON EMP
  REFERENCING NEW AS N_EMP
  FOR EACH ROW MODE DB2SQL
  BEGIN ATOMIC
    UPDATE DEPTS SET NBEMP = NBEMP + 1
      WHERE DEPT_ID = N_EMP.DEPT_ID;
  END
```

Cuando se produce una operación de inserción en la tabla EMP, Db2 activa primero NEWHIRE1 porque NEWHIRE1 se creó primero. Ahora supongamos que alguien crea y vuelve a crear NEWHIRE1. NEWHIRE1 ahora tiene una marca de tiempo posterior a NEWHIRE2, por lo que la próxima vez que se produzca una operación de inserción en EMP, NEWHIRE2 se activará antes que NEWHIRE1.

Si se definen dos activadores de fila para la misma acción, el activador que se creó antes se activa primero para todas las filas afectadas. A continuación, se activa el segundo desencadenante para todas las filas afectadas. En el ejemplo anterior, supongamos que una sentencia INSERT con un fullselect inserta 10 filas en la tabla EMP. NEWHIRE1 se activa para las 10 filas, luego se activa NEWHIRE2 para las 10 filas.

Referencia relacionada

[Instrucción CREATE TRIGGER \(disparador avanzado\) \(Db2 SQL\)](#)

[Instrucción CREATE TRIGGER \(disparador básico\) \(Db2 SQL\)](#)

Interacciones entre desencadenadores y restricciones referenciales

Al crear desencadenantes, necesita comprender las interacciones entre los desencadenantes y las restricciones en las tablas. También es necesario para comprender el efecto que el orden del proceso de esas restricciones y desencadenantes puede tener en los resultados.

En general, los siguientes pasos ocurren al activar la instrucción SQL S1 realiza una operación de inserción, actualización o eliminación en la tabla T1:

1. Db2 determina las filas de T1 o que se van a modificar. Llama a ese conjunto de filas M1. El contenido de M1 depende de la operación SQL:

- Para una operación de eliminación, todas las filas que satisfagan la condición de búsqueda de la instrucción para una operación de eliminación buscada, o la fila actual para una operación de eliminación posicionada
- Para una operación de inserción, la fila identificada por la instrucción VALUES, o las filas identificadas por la tabla de resultados de una cláusula SELECT dentro de la instrucción INSERT
- Para una operación de actualización, todas las filas que satisfacen la condición de búsqueda de la declaración para una operación de actualización buscada, o la fila actual para una operación de actualización posicionada

2. Db2 procesa todos los activadores previos que se definen en T1, por orden de creación.

Cada antes de activar ejecuta la acción activada una vez para cada fila en M1. Si M1 está vacío, la acción activada no se ejecuta.

Si se produce un error al ejecutar la acción desencadenada, Db2 deshace todos los cambios realizados por S1.

3. Db2 realiza los cambios especificados en la declaración S1 a la tabla T1, a menos que se defina un desencadenante INSTEAD OF para esa acción. Si se define un desencadenante apropiado INSTEAD OF, Db2 ejecuta el desencadenante en lugar de la declaración y omite los pasos restantes de esta lista.

Si se produce un error, Db2 deshace todos los cambios realizados por S1.

4. Si M1 no está vacío, Db2 aplica todas las restricciones y comprobaciones siguientes que se definen en la tabla T1:

- Restricciones referenciales
- Restricciones de comprobación
- Cheques que se deben a actualizaciones de la tabla a través de vistas definidas CON OPCIÓN DE CHEQUE

La aplicación de restricciones referenciales con reglas de DELETE CASCADE o DELETE SET NULL se activa antes de los desencadenantes de eliminación o antes de los desencadenantes de actualización en las tablas dependientes.

Si se infringe alguna restricción, Db2 revierte todos los cambios realizados por acciones de restricción o por la declaración S1.

5. Db2 procesa todos los after triggers que se definen en T1 y todos los after triggers en tablas que se modifican como resultado de acciones de restricciones referenciales, en orden de creación.

Cada activador de fila posterior ejecuta la acción activada una vez por cada fila en M1. Si M1 está vacío, la acción activada no se ejecuta.

Cada activador de la sentencia after ejecuta la acción desencadenada una vez por cada ejecución de S1, incluso si M1 está vacío.

Si alguna de las acciones activadas contiene operaciones SQL de inserción, actualización o eliminación, Db2 repite los pasos del 1 al 5 para cada operación.

Si se produce un error al ejecutar la acción desencadenada, o si una acción desencadenada se encuentra en el nivel e 17th o de la cascada de desencadenantes, Db2 deshace todos los cambios realizados en el paso 5 y en todos los pasos anteriores.

Por ejemplo, la tabla DEPT es una tabla principal de EMP, con estas condiciones:

- La columna DEPTNO de DEPT es la clave principal.
- La columna WORKDEPT de EMP es la clave externa.
- La restricción es ON DELETE SET NULL.

Supongamos que el siguiente desencadenante se define en EMP:

```
CREATE TRIGGER EMPRAISE
  AFTER UPDATE ON EMP
  REFERENCING NEW TABLE AS NEWEMPS
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES(CHECKEMP(TABLE NEWEMPS));
  END
```

Supongamos también que una instrucción SQL elimina la fila con el número de departamento E21 de DEPT. Debido a la restricción, Db2 encuentra las filas en EMP con un valor WORKDEPT de E21 y establece WORKDEPT en esas filas en nulo. Esto equivale a una operación de actualización en EMP, que tiene el desencadenante de actualización EMPRAISE. Por lo tanto, como EMPRAISE es un desencadenador posterior, EMPRAISE se activa después de que la acción de restricción establezca los valores de WORKDEPT en nulo.

Interacciones entre desencadenantes y tablas que tienen seguridad multinivel con granularidad a nivel de fila

Un desencadenante BEFORE afecta al valor de la variable de transición que está asociada a una columna de etiqueta de seguridad.

Si una tabla de sujetos tiene una columna de etiqueta de seguridad, la columna de la tabla de transición o variable de transición que se corresponde con la columna de etiqueta de seguridad de la tabla de sujetos no hereda el atributo de etiqueta de seguridad. Esto significa que el control de seguridad multinivel con granularidad a nivel de fila no se aplica a la tabla de transición ni a la variable de transición. Si agrega una columna de etiqueta de seguridad a una tabla de asuntos mediante la instrucción ALTER TABLE, las reglas son las mismas que cuando se agrega cualquier columna a una tabla de asuntos porque la columna de la tabla de transición o la variable de transición que corresponde a la columna de etiqueta de seguridad no hereda el atributo de etiqueta de seguridad.

Si el ID que está utilizando no tiene privilegios de escritura y ejecuta una operación de inserción o actualización, el valor de la etiqueta de seguridad de su ID se asigna a la columna de etiquetas de seguridad de las filas que está insertando o actualizando.

Cuando se activa un desencadenante BEFORE, el valor de la variable de transición que corresponde a la columna de etiqueta de seguridad es la etiqueta de seguridad del ID si se cumple alguna de las siguientes condiciones:

- El usuario no tiene privilegios de escritura
- El valor de la columna de la etiqueta de seguridad no está especificado

Si el usuario no tiene privilegios de escritura y el desencadenante cambia la variable de transición que corresponde a la columna de etiqueta de seguridad, el valor de la columna de etiqueta de seguridad vuelve a cambiar al valor de etiqueta de seguridad del usuario antes de que la fila se escriba en la página.

Conceptos relacionados

[Seguridad multinivel \(Gestión de la seguridad\)](#)

Desencadenantes que devuelven resultados incoherentes

Cuando se crean disparadores y se escriben sentencias SQL que activan esos disparadores, es necesario asegurarse de que la ejecución de esas sentencias siempre produce los mismos resultados.

Dos razones comunes por las que puede obtener resultados inconsistentes son:

- Las sentencias UPDATE o DELETE posicionadas que utilizan subconsultas no correlacionadas hacen que los disparadores operen en una tabla de resultados más grande de lo que pretendías.
- Db2 no siempre procesa las filas en el mismo orden, por lo que los disparadores que propagan las filas de una tabla pueden generar tablas de resultados diferentes en momentos diferentes.

Los siguientes ejemplos demuestran estas situaciones.

Ejemplo: Efecto de una subconsulta no correlacionada en una acción desencadenada : Supongamos que las tablas T1 y T2 tienen este aspecto:

Table T1	Table T2
A1	B1
==	==
1	1
2	2

El siguiente desencadenante se define en T1:

```
CREATE TRIGGER TR1
AFTER UPDATE OF T1
FOR EACH ROW
MODE DB2SQL
BEGIN ATOMIC
    DELETE FROM T2 WHERE B1 = 2;
END
```

Ahora suponga que una aplicación ejecuta las siguientes sentencias para realizar una operación de actualización posicionada:

```
EXEC SQL BEGIN DECLARE SECTION;
  long hv1;
EXEC SQL END DECLARE SECTION;
:
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT A1 FROM T1
  WHERE A1 IN (SELECT B1 FROM T2)
  FOR UPDATE OF A1;
:
EXEC SQL OPEN C1;
:
while(SQLCODE>=0 && SQLCODE!=100)
{
  EXEC SQL FETCH C1 INTO :hv1;
  UPDATE T1 SET A1=5 WHERE CURRENT OF C1;
}
```

Cuando Db2 ejecuta la sentencia `FETCH` que posiciona el cursor `C1` por primera vez, Db2 evalúa la subselección, `SELECT B1 FROM T2`, para producir una tabla de resultados que contiene las dos filas de la columna `T2`:

```
1
2
```

Cuando Db2 ejecuta la instrucción `UPDATE` posicionada por primera vez, se activa el desencadenador `TR1`. Cuando se ejecuta el cuerpo de la e `TR1` a de activación, la fila con el valor 2 se elimina de `T2`. Sin embargo, como `SELECT B1 FROM T2` se evalúa solo una vez, cuando la sentencia `FETCH` se ejecuta de nuevo, Db2 encuentra la segunda fila de `T1`, aunque la segunda fila de `T2` se haya eliminado. La sentencia `FETCH` posiciona el cursor en la segunda fila de `T1`, y se actualiza la segunda fila de `T1`. La operación de actualización hace que el activador se active de nuevo, lo que provoca que Db2 intente eliminar la segunda fila de `T2`, aunque esa fila ya se había eliminado.

Para evitar el procesamiento de la segunda fila después de que debería haberse eliminado, utilice una subconsulta correlacionada en la declaración del cursor:

```
DCL C1 CURSOR FOR
  SELECT A1 FROM T1 X
  WHERE EXISTS (SELECT B1 FROM T2 WHERE X.A1 = B1)
  FOR UPDATE OF A1;
```

En este caso, la subconsulta, `SELECT B1 FROM T2 WHERE X.A1 = B1`, se evalúa para cada sentencia `FETCH`. La primera vez que se ejecuta la sentencia `FETCH`, posiciona el cursor en la primera fila de `T1`. La operación `UPDATE` posicionada activa el disparador, que elimina la segunda fila de `T2`. Por lo tanto, cuando la sentencia `FETCH` se ejecuta de nuevo, no se selecciona ninguna fila, por lo que no se produce ninguna operación de actualización ni se activa ninguna acción.

Ejemplo: Efecto del orden de procesamiento de filas en una acción desencadenada : El siguiente ejemplo muestra cómo el orden de procesamiento de filas puede cambiar el resultado de un desencadenador posterior a la fila.

Supongamos que las tablas `T1`, `T2` y `T3` tienen este aspecto:

Table T1	Table T2	Table T3
A1 == 1 2	B1 == (empty)	C1 == (empty)

El siguiente desencadenante se define en `T1`:

```
CREATE TRIGGER TR1
  AFTER UPDATE ON T1
  REFERENCING NEW AS N
  FOR EACH ROW
  MODE DB2SQL
  BEGIN ATOMIC
    INSERT INTO T2 VALUES(N.C1);
```

```
INSERT INTO T3 (SELECT B1 FROM T2);  
END
```

Ahora suponga que un programa ejecuta la siguiente instrucción UPDATE:

```
UPDATE T1 SET A1 = A1 + 1;
```

El contenido de las tablas T2 y T3 después de que se ejecute la instrucción UPDATE depende del orden en que Db2 actualiza las filas de T1.

Si Db2 actualiza primero la primera fila de T1, después de que la instrucción UPDATE y el desencadenador se ejecuten por primera vez, los valores en las tres tablas son:

Table T1	Table T2	Table T3
A1 == 2 2	B1 == 2	C1 == 2

Después de actualizar la segunda fila de T1, los valores de las tres tablas son:

Table T1	Table T2	Table T3
A1 == 2 3	B1 == 2 3	C1 == 2 3

Sin embargo, si Db2 actualiza primero la segunda fila de T1, después de que la instrucción UPDATE y el desencadenador se ejecuten por primera vez, los valores en las tres tablas son:

Table T1	Table T2	Table T3
A1 == 1 3	B1 == 3	C1 == 3

Después de actualizar la primera fila de T1, los valores de las tres tablas son:

Table T1	Table T2	Table T3
A1 == 2 3	B1 == 3 2	C1 == 3 2

Conversión de desencadenantes existentes para dar soporte a prestaciones avanzadas

Puede convertir los desencadenantes básicos existentes para aprovechar las prestaciones avanzadas, incluido el soporte para más sentencias SQL, incluido el SQL PL en el cuerpo del desencadenante, soporte para más tipos de variables y otras ventajas.

Antes de empezar

Los desencadenantes avanzados reciben soporte en el nivel de compatibilidad de aplicación V12R1M500 o superior.

Puede identificar desencadenantes básicos consultando la tabla de catálogo SYSIBM.SYSTRIGGERS. Los valores en blanco de la columna SQLPL identifican desencadenantes básicos.

Acerca de esta tarea

Desencadenantes avanzados ofrece las siguientes ventajas sobre los desencadenantes básicos:

- En la definición de desencadenante, un desencadenante avanzado puede:

- Incluir más tipos de sentencias de SQL, incluidas sentencias de control de SQL PL, sentencias de SQL dinámico y comentarios de SQL.
- Definir y hacer referencia a más tipos de variables, incluidas las variables de SQL y las variables globales.
- Especificar explícitamente las opciones de enlace.
- Definir varias versiones del desencadenante.
- Todas las variables de transición pueden ser valores nulos.
- Las sentencias ALTER TRIGGER pueden cambiar las opciones y cambiar o regenerar el cuerpo del desencadenante.
- La cláusula OR REPLACE se puede utilizar en sentencias CREATE TRIGGER (avanzadas). Permite el uso de una sola sentencia CREATE para definir un nuevo desencadenante o una versión de desencadenante, o actualizar una versión de desencadenante o un desencadenante existente, si ya existe.

Para obtener más información sobre las diferencias entre los desencadenantes básicos y avanzados, consulte [Desencadenantes \(Introducción a DB2 para z/OS\)](#).

Procedimiento

Para cambiar un desencadenante básico existente por un desencadenante avanzado, realice los pasos siguientes:

1. Modifique la sentencia CREATE TRIGGER original en una sentencia CREATE TRIGGER (avanzada) eliminando cualquiera de los elementos siguientes:
 - La cláusula MODE DB2SQL. Db2 intenta crear un desencadenante básico si se incluye esa cláusula.
 - Sentencias *fullselect* o VALUES autónomas. Puede utilizar sentencias SELECT INTO o VALUES INTO en su lugar.
2. Utilice uno de los siguientes enfoques para convertir a la nueva definición de desencadenante avanzado:
 - Emite la sentencia CREATE TRIGGER (avanzada) modificada con la cláusula OR REPLACE.
 - Emite una sentencia DROP para el desencadenante original y, a continuación, emita la nueva sentencia CREATE TRIGGER.

El desencadenante existente se elimina de forma efectiva y se define un desencadenante avanzado. Si se definen varios desencadenantes en la tabla asociada, el orden de activación de desencadenante cambia.

Qué hacer a continuación

Si se definen varios desencadenantes en la tabla asociada, es posible que tenga que restaurar el orden de activación original de los desencadenantes. Para ello, debe descartar y volver a crear los desencadenantes que se crearon después de que se creara originalmente el desencadenante convertido, en el mismo orden en que se crearon originalmente. Para obtener más información sobre el orden de activación de múltiples activadores, consulte “[Orden de activación de varios desencadenantes](#)” en la página 173.

Referencia relacionada

[DROP declaración \(Db2 SQL\)](#)

[Instrucción CREATE TRIGGER \(disparador avanzado\) \(Db2 SQL\)](#)

[Tabla de catálogo SYSTRIGERS \(Db2 SQL\)](#)

Objetos de secuencia

Una secuencia es un objeto definido por el usuario que genera una secuencia de valores numéricos de acuerdo con la especificación con la que se creó la secuencia. Las secuencias, a diferencia de las columnas de identidad, no están asociadas a las tablas. Las aplicaciones hacen referencia a un objeto de secuencia para obtener su valor actual o siguiente.

La secuencia de valores numéricos se genera en un orden monótono ascendente o descendente. La relación entre secuencias y tablas está controlada por la aplicación, no por Db2.

Su aplicación puede hacer referencia a un objeto de secuencia y coordinar el valor como claves en varias filas y tablas. Sin embargo, una columna de tabla que obtiene sus valores de un objeto de secuencia no tiene necesariamente valores únicos en esa columna. Incluso si el objeto de secuencia se ha definido con la cláusula NO CYCLE, alguna otra aplicación podría insertar valores en esa columna de la tabla distintos de los valores que usted obtiene al hacer referencia a ese objeto de secuencia.

Db2 siempre genera números de secuencia en orden de solicitud. Sin embargo, en un grupo de intercambio de datos en el que los valores de secuencia son almacenados en caché por varios miembros de Db2 simultáneamente, las asignaciones de valores de secuencia podrían no estar en orden numérico. Además, es posible que haya lagunas en los valores de los números de secuencia por las siguientes razones:

- Si Db2 finaliza de forma anormal antes de asignar todos los valores almacenados en caché
- Si su aplicación revierte una transacción que incrementa la secuencia
- Si el enunciado que contiene NEXT VALUE falla después de incrementar la secuencia

Puede crear un objeto de secuencia con la sentencia CREATE SEQUENCE, modificarlo con la sentencia ALTER SEQUENCE y eliminarlo con la sentencia DROP SEQUENCE. Usted concede acceso a una secuencia con la sentencia GRANT (privilegio) ON SEQUENCE, y revoca el acceso a la secuencia con la sentencia REVOKE (privilegio) ON SEQUENCE.

Los valores que genera Db2 para una secuencia dependen de cómo se cree la secuencia. La opción START WITH (Comenzar con) determina el primer valor que genera Db2 . Los valores avanzan por el valor INCREMENT BY en orden ascendente o descendente.

Las opciones MINVALUE y MAXVALUE determinan los valores mínimo y máximo que genera Db2 . La opción CICLO o SIN CICLO determina si Db2 envuelve los valores generados cuando alcanza el valor máximo para una secuencia ascendente o el valor mínimo en una secuencia descendente.

Claves en varias tablas : Puede utilizar el mismo número de secuencia como valor clave en dos tablas separadas generando primero el valor de secuencia con una expresión NEXT VALUE para insertar la primera fila en la primera tabla. A continuación, puede hacer referencia a este mismo valor de secuencia con una expresión PREVIOUS VALUE para insertar las otras filas en la segunda tabla.

Ejemplo : Supongamos que una tabla ORDERS y una tabla ORDER_ITEMS se definen de la siguiente manera:

```
CREATE TABLE ORDERS
  (ORDERNO      INTEGER NOT NULL,
   ORDER_DATE   DATE DEFAULT,
   CUSTNO       SMALLINT
   PRIMARY KEY (ORDERNO));

CREATE TABLE ORDER_ITEMS
  (ORDERNO      INTEGER NOT NULL,
   PARTNO      INTEGER NOT NULL,
   QUANTITY    SMALLINT NOT NULL,
   PRIMARY KEY (ORDERNO,PARTNO),
   CONSTRAINT REF_ORDERNO FOREIGN KEY (ORDERNO)
     REFERENCES ORDERS (ORDERNO) ON DELETE CASCADE);
```

Cree una secuencia llamada ORDER_SEQ para usarla como valores clave para las tablas ORDERS y ORDER_ITEMS:

```
CREATE SEQUENCE ORDER_SEQ AS INTEGER
  START WITH 1
  INCREMENT BY 1
  NO MAXVALUE
  NO CYCLE
  CACHE 20;
```

A continuación, puede utilizar el mismo número de secuencia como valor de clave principal para la tabla ORDERS y como parte del valor de clave principal para la tabla ORDER_ITEMS:

```
INSERT INTO ORDERS (ORDERNO, CUSTNO)
    VALUES (NEXT VALUE FOR ORDER_SEQ, 12345);

INSERT INTO ORDER_ITEMS (ORDERNO, PARTNO, QUANTITY)
    VALUES (PREVIOUS VALUE FOR ORDER_SEQ, 987654, 2);
```

La expresión NEXT VALUE en la primera sentencia INSERT genera un valor de número de secuencia para el objeto de secuencia ORDER_SEQ. La expresión PREVIOUS VALUE en la segunda sentencia INSERT recupera ese mismo valor porque fue el número de secuencia generado más recientemente para ese objeto de secuencia dentro del proceso de aplicación actual.

Db2 objetos extensiones relacionales

Con las extensiones de objetos de Db2, puede incorporar conceptos y metodologías orientados a objetos en su base de datos relacional ampliando Db2 con conjuntos más ricos de tipos de datos y funciones.

Con esas extensiones, puede almacenar instancias de tipos de datos orientados a objetos en columnas de tablas y operar con ellas utilizando funciones en sentencias SQL. Además, puede controlar los tipos de operaciones que los usuarios pueden realizar en esos tipos de datos.

Db2 proporciona las siguientes extensiones de objetos:

- Objetos grandes (LOB)

Los tipos de datos VARCHAR, VARGRAPHIC y VARBINARY tienen un límite de almacenamiento de 32 KB. Aunque esto podría ser suficiente para datos de texto de tamaño pequeño a mediano, las aplicaciones a menudo necesitan almacenar documentos de texto grandes. También pueden necesitar almacenar una amplia variedad de tipos de datos adicionales, como audio, video, dibujos, texto y gráficos mezclados, e imágenes. Db2 proporciona tres tipos de datos para almacenar estos objetos de datos como cadenas de hasta 2 GB - 1 de tamaño. Los tres tipos de datos son objetos binarios de gran tamaño (BLOB), objetos de gran tamaño de caracteres (CLOB) y objetos de gran tamaño de caracteres de doble byte (DBCLOB).

Para un análisis detallado de los LOB, consulte [“Almacenamiento de datos LOB en tablas de Db2”](#) en la página 131 y [Objetos grandes \(LOB\) \(Db2 SQL\)](#).

- Tipos diferenciados

Un tipo distinto es un tipo de datos definido por el usuario que comparte su representación interna con un tipo de datos integrado, pero que se considera un tipo separado e incompatible por motivos semánticos. Por ejemplo, es posible que desee definir un tipo de imagen o un tipo de audio, los cuales tienen semánticas bastante diferentes, pero que utilizan el tipo de datos incorporado BLOB para su representación interna.

Para un análisis detallado de los distintos tipos, consulte [“Tipos diferenciados”](#) en la página 182.

- Funciones definidas por el usuario

Las funciones integradas que se suministran con un Db2 son un conjunto de funciones útiles, pero es posible que no satisfagan todos sus requisitos. Para esos casos, puede utilizar funciones definidas por el usuario. Por ejemplo, una función integrada podría realizar un cálculo que usted necesita, pero la función no acepta los distintos tipos que usted quiere pasarle. A continuación, puede definir una función basada en una función incorporada, denominada función definida por el usuario de origen, que acepte sus tipos distintos. Es posible que tenga que realizar otro cálculo en sus sentencias SQL para el que no exista ninguna función integrada. En esa situación, puede definir y escribir una función SQL o una función externa.

Para una explicación detallada de las funciones definidas por el usuario, consulte [“Pasos para crear y utilizar una función definida por el usuario”](#) en la página 196.

Creación de un tipo diferenciado

Los tipos distintos son útiles cuando se desea que Db2 maneje ciertos datos de manera diferente a otros datos del mismo tipo de datos. Por ejemplo, aunque todas las monedas se pueden declarar como tipo DECIMAL, no desea que se comparan euros con el yen japonés.

Procedimiento

Emitir la declaración CREATE DISTINCT TYPE.

Por ejemplo, puede crear tipos distintos para euros y yenes emitiendo las siguientes sentencias SQL:

```
CREATE DISTINCT TYPE EURO AS DECIMAL(9,2);
CREATE DISTINCT TYPE JAPANESE_YEN AS DECIMAL(9,2);
```

Referencia relacionada

[Instrucción CREATE TYPE \(tipo distinto\) \(Db2 SQL\)](#)

Tipos diferenciados

Un tipo distinto es un tipo de datos definido por el usuario que comparte su representación interna con un tipo de datos incorporado (su tipo de origen), pero se considera un tipo de datos independiente e incompatible para la mayoría de las operaciones.

Cada tipo distinto tiene la misma representación interna que un tipo de datos integrado.

Supongamos que desea definir algunos datos de audio y vídeo en una tabla de tipo " Db2 ". Puede definir columnas para ambos tipos de datos como BLOB, pero es posible que desee utilizar un tipo de datos que describa los datos de forma más específica. Para ello, defina tipos distintos. A continuación, puede utilizar esos tipos cuando defina columnas en una tabla o manipule los datos de esas columnas. Por ejemplo, puede definir tipos distintos para los datos de audio y vídeo de esta manera:

```
CREATE DISTINCT TYPE AUDIO AS BLOB (1M);
CREATE DISTINCT TYPE VIDEO AS BLOB (1M);
```

Entonces, su declaración CREATE TABLE podría tener este aspecto:

```
CREATE TABLE VIDEO_CATALOG;
(VIDEO_NUMBER CHAR(6) NOT NULL,
VIDEO_SOUND AUDIO,
VIDEO_PICS VIDEO,
ROW_ID ROWID NOT NULL GENERATED ALWAYS);
```

Para obtener más información sobre los datos de LOB, consulte ["Almacenamiento de datos LOB en tablas de Db2 "](#) en la página 131 y [Objetos grandes \(LOB\) \(Db2 SQL\)](#).

Después de definir tipos y columnas distintos de esos tipos, puede utilizar esos tipos de datos de la misma manera que utiliza los tipos integrados. Puede utilizar los tipos de datos en asignaciones, comparaciones, invocaciones de funciones y llamadas a procedimientos almacenados. Sin embargo, cuando asignas un valor de columna a otro o comparas dos valores de columna, esos valores deben ser del mismo tipo distinto. Por ejemplo, debe asignar un valor de columna de tipo VIDEO a una columna de tipo VIDEO, y puede comparar un valor de columna de tipo AUDIO solo con una columna de tipo AUDIO. Cuando asignas un valor de variable de host a una columna con un tipo distinto, puedes utilizar cualquier tipo de datos de host que sea compatible con el tipo de datos de origen del tipo distinto. Por ejemplo, para recibir un valor de AUDIO o VÍDEO, puede definir una variable de host de esta manera:

```
SQL TYPE IS BLOB (1M) HVAV;
```

Cuando se utiliza un tipo distinto como argumento de una función, debe existir una versión de esa función que acepte ese tipo distinto. Por ejemplo, si la función SIZE toma un tipo BLOB como entrada, no puede utilizar automáticamente un valor de tipo AUDIO como entrada. Sin embargo, puede crear una función definida por el usuario que tome el tipo AUDIO como entrada. Por ejemplo:

```
CREATE FUNCTION SIZE(AUDIO)
RETURNS INTEGER
SOURCE SIZE(BLOB(1M));
```

Uso de tipos distintos en programas de aplicación : La razón principal para usar tipos distintos es que Db2 aplica *una tipificación fuerte* para tipos distintos. La tipificación fuerte garantiza que solo se puedan utilizar funciones, procedimientos, comparaciones y asignaciones que estén definidas para un tipo de datos.

Por ejemplo, si ha definido una función definida por el usuario para convertir U.S. dólares a euros, no querrá que nadie utilice esta misma función definida por el usuario para convertir yenes japoneses a euros porque la función U.S. dólares a euros devuelve una cantidad incorrecta. Supongamos que define tres tipos distintos:

```
CREATE DISTINCT TYPE US_DOLLAR AS DECIMAL(9,2);
CREATE DISTINCT TYPE EURO AS DECIMAL(9,2);
CREATE DISTINCT TYPE JAPANESE_YEN AS DECIMAL(9,2);
```

Si se define una función de conversión que toma un parámetro de entrada de tipo US_DOLLAR como entrada, Db2 devuelve un error si se intenta ejecutar la función con un parámetro de entrada de tipo JAPANESE_YEN.

Ejemplo de tipos distintos, funciones definidas por el usuario y LOB

Puede crear y utilizar un tipo distinto basado en un tipo de datos LOB.

El ejemplo de este tema demuestra los siguientes conceptos:

- Creación de un tipo distinto basado en un tipo de datos LOB
- Definir una función definida por el usuario con un tipo distinto como argumento
- Crear una tabla con una columna de tipo distinta basada en un tipo LOB
- Definición de un espacio de tabla LOB, una tabla auxiliar y un índice auxiliar
- Insertar datos de una variable de host en una columna de tipo distinto basada en una columna LOB
- Ejecución de una consulta que contiene una invocación de función definida por el usuario
- Lanzar un localizador LOB al tipo de datos de entrada de una función definida por el usuario

Supongamos que guarda los documentos de correo electrónico que se envían a su empresa en una tabla de almacenamiento intermedio (Db2). El tipo de datos de correo electrónico (Db2) de un documento de correo electrónico es un CLOB, pero lo define como un tipo distinto para poder controlar los tipos de operaciones que se realizan en el correo electrónico. El tipo distinto se define así:

```
CREATE DISTINCT TYPE E_MAIL AS CLOB(5M);
```

También ha definido y escrito funciones definidas por el usuario para buscar y devolver la siguiente información sobre un documento de correo electrónico:

- Asunto:
- Emisor
- Fecha de envío
- Contenido de mensaje
- Indicador de si el documento contiene una cadena especificada por el usuario

Las definiciones de funciones definidas por el usuario tienen este aspecto:

```
CREATE FUNCTION SUBJECT(E_MAIL)
RETURNS VARCHAR(200)
EXTERNAL NAME 'SUBJECT'
LANGUAGE C
PARAMETER STYLE SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION;
```

```
CREATE FUNCTION SENDER(E_MAIL)
RETURNS VARCHAR(200)
EXTERNAL NAME 'SENDER'
LANGUAGE C
PARAMETER STYLE SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION;
```

```

CREATE FUNCTION SENDING_DATE(E_MAIL)
RETURNS DATE
EXTERNAL NAME 'SENDDATE'
LANGUAGE C
PARAMETER STYLE SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION;

CREATE FUNCTION CONTENTS(E_MAIL)
RETURNS CLOB(1M)
EXTERNAL NAME 'CONTENTS'
LANGUAGE C
PARAMETER STYLE SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION;

CREATE FUNCTION CONTAINS(E_MAIL, VARCHAR (200))
RETURNS INTEGER
EXTERNAL NAME 'CONTAINS'
LANGUAGE C
PARAMETER STYLE SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION;

```

La tabla que contiene los documentos de correo electrónico se define así:

```

CREATE TABLE DOCUMENTS
(LAST_UPDATE_TIME TIMESTAMP,
DOC_ROWID ROWID NOT NULL GENERATED ALWAYS,
A_DOCUMENT E_MAIL);

```

Debido a que la tabla contiene una columna con un tipo de datos de origen CLOB, la tabla requiere un espacio de tabla LOB asociado, una tabla auxiliar y un índice en la tabla auxiliar. Utilice sentencias como esta para definir el espacio de tabla LOB, la tabla auxiliar y el índice:

```

CREATE LOB TABLESPACE DOCTSLOB
LOG YES
GBPCACHE SYSTEM;

CREATE AUX TABLE DOCAUX_TABLE
IN DOCTSLOB
STORES DOCUMENTS COLUMN A_DOCUMENT;

CREATE INDEX A_IX_DOC ON DOCAUX_TABLE;

```

Para llenar la tabla de documentos, se escribe código que ejecuta una instrucción INSERT para poner la primera parte de un documento en la tabla, y luego ejecuta múltiples instrucciones UPDATE para concatenar las partes restantes del documento. Por ejemplo:

```

EXEC SQL BEGIN DECLARE SECTION;
char hv_current_time[26];
SQL TYPE IS CLOB (1M) hv_doc;
EXEC SQL END DECLARE SECTION;
/* Determine the current time and put this value */
/* into host variable hv_current_time. */
/* Read up to 1 MB of document data from a file */
/* into host variable hv_doc. */
:
/* Insert the time value and the first 1 MB of */
/* document data into the table. */
EXEC SQL INSERT INTO DOCUMENTS
VALUES(:hv_current_time, DEFAULT, E_MAIL(:hv_doc));

/* Although there is more document data in the */
/* file, read up to 1 MB more of data, and then */
/* use an UPDATE statement like this one to */
/* concatenate the data in the host variable */
/* to the existing data in the table. */
EXEC SQL UPDATE DOCUMENTS

```

```
SET A_DOCUMENT = A_DOCUMENT || E_MAIL(:hv_doc)
WHERE LAST_UPDATE_TIME = :hv_current_time;
```

Ahora que los datos están en la tabla, puede ejecutar consultas para obtener más información sobre los documentos. Por ejemplo, puede ejecutar esta consulta para determinar qué documentos contienen la palabra «performance»:

```
SELECT SENDER(A_DOCUMENT), SENDING_DATE(A_DOCUMENT),
SUBJECT(A_DOCUMENT)
FROM DOCUMENTS
WHERE CONTAINS(A_DOCUMENT, 'performance') = 1;
```

Dado que los documentos de correo electrónico pueden ser muy grandes, es posible que desee utilizar localizadores LOB para manipular los datos del documento en lugar de obtener todo un documento en una variable de host. Puede utilizar un localizador LOB en cualquier tipo distinto que esté definido en uno de los tipos LOB. El siguiente ejemplo muestra cómo puede convertir un localizador LOB en un tipo distinto y, a continuación, utilizar el resultado en una función definida por el usuario que toma un tipo distinto como argumento:

```
EXEC SQL BEGIN DECLARE SECTION
  long hv_len;
  char hv_subject[200];
  SQL TYPE IS CLOB_LOCATOR hv_email_locator;
EXEC SQL END DECLARE SECTION
:
/* Select a document into a CLOB locator.      */
EXEC SQL SELECT A_DOCUMENT, SUBJECT(A_DOCUMENT)
  INTO :hv_email_locator, :hv_subject
  FROM DOCUMENTS
 WHERE LAST_UPDATE_TIME = :hv_current_time;
:
/* Extract the subject from the document. The   */
/* SUBJECT function takes an argument of type    */
/* E_MAIL, so cast the CLOB locator as E_MAIL. */
EXEC SQL SET :hv_subject =
  SUBJECT(CAST(:hv_email_locator AS E_MAIL));
:
```

Matrices en sentencias de SQL

Una matriz es un conjunto ordenado de elementos de un único tipo de datos incorporado. Una matriz puede tener un tipo de matriz definido por el usuario asociado, o puede ser el resultado de una operación SQL que devuelve un valor de matriz sin un tipo de matriz definido por el usuario asociado.

Las matrices pueden ser *matrices ordinarias* y *matrices asociativas*.

Las matrices ordinarias tienen un límite superior definido por el usuario. Se puede acceder a los elementos de la matriz y modificarlos mediante su valor de índice. Los elementos de la matriz se mencionan en las sentencias SQL mediante el uso de una indexación basada en uno; por ejemplo, MYARRAY[1], MYARRAY[2], etc.

Las matrices asociativas no tienen límite superior. Las matrices asociativas contienen un conjunto ordenado de cero o más elementos, donde cada elemento de la matriz está ordenado por un valor de índice asociado y puede ser referenciado por este. El tipo de datos de los valores de índice puede ser un entero o una cadena de caracteres, pero todos los valores de índice de la matriz tienen el mismo tipo de datos.

Las matrices solo pueden utilizarse en los siguientes contextos:

- Parámetros para funciones SQL
- RETURN tipos de datos de funciones SQL
- Parámetros de procedimientos SQL
- Variables SQL que se declaran en funciones SQL
- Variables SQL que se declaran en procedimientos SQL

Puede crear una matriz creando un tipo de matriz y, a continuación, definiendo una variable de matriz de ese tipo. Por ejemplo:

```
-- CREATE ORDINARY ARRAY TYPE INTARRAY  
CREATE TYPE INTARRAY AS INTEGER ARRAY[100];  
-- IN AN SQL PROCEDURE, DEFINE ARRAY INTA OF THE INTARRAY TYPE  
DECLARE INTA INTARRAY;  
-- CREATE ASSOCIATIVE ARRAY TYPE CHARARRAY  
CREATE TYPE CHARARRAY AS CHAR(10) ARRAY[VARCHAR(10)];  
-- IN AN SQL PROCEDURE, DEFINE ARRAY CHARA OF THE CHARARRAY TYPE  
DECLARE CHARA CHARARRAY;
```

No se puede recuperar el contenido de una columna directamente en una matriz. Debe utilizar la función ARRAY_AGG para crear una matriz que sea el resultado intermedio de una instrucción SELECT y, a continuación, recuperar el contenido de esa matriz en una variable o parámetro de matriz SQL. Por ejemplo:

```
-- INTB IS AN OUT PARAMETER OF ORDINARY ARRAY TYPE INTARRAY.  
-- COL2 IS AN INTEGER COLUMN.  
-- ARRAY_AGG RETRIEVES THE VALUES FROM COL2, AND PUTS THEM INTO AN ARRAY.  
SELECT ARRAY_AGG(COL2) INTO INTB FROM TABLE1;
```

Puede recuperar datos de una matriz utilizando la especificación UNNEST para asignar elementos de la matriz a una tabla de resultados intermedios. Por ejemplo:

```
-- IDS AND NAMES ARE ARRAYS OF TYPE INTARRAY.  
INSERT INTO PERSONS(ID, NAME)  
(SELECT T.I, T.N FROM UNNEST(IDS, NAMES) AS T(I, N));
```

Para llenar matrices, se utilizan *constructores de matrices*.

Por ejemplo, esta declaración llena una matriz ordinaria:

```
SET CHARA = ARRAY['1','2','3','4','5','6'];
```

Por ejemplo, estas declaraciones llenan una matriz asociativa, que debe llenarse un elemento cada vez:

```
SET CANADACAPITALS['Alberta'] = 'Edmonton';  
SET CANADACAPITALS['Manitoba'] = 'Winnipeg';  
SET CANADACAPITALS['Ontario'] = 'Toronto';  
SET CANADACAPITALS['Nova Scotia'] = 'Halifax';
```

Hay varias funciones integradas disponibles para manipular matrices. Estas funciones son:

ARRAY_DELETE

Suprime elementos de una matriz.

ARRAY_FIRST

Devuelve el valor de índice mínimo de una matriz.

ARRAY_LAST

Devuelve el valor de índice máximo de una matriz.

ARRAY_NEXT

Devuelve el siguiente valor de índice de matriz más grande, relativo a un valor de índice de matriz especificado.

ARRAY_PRIOR

Devuelve el siguiente valor de índice de matriz más pequeño, relativo a un valor de índice de matriz especificado.

CARDINALITY

Devuelve el número de elementos de una matriz.

MAX_CARDINALITY

Devuelve el número máximo de elementos que puede contener una matriz.

TRIM_ARRAY

Elimina elementos del final de una matriz ordinaria.

Conceptos relacionados

[Comparaciones de tipos definidos por el usuario \(Db2 SQL\)](#)

[Asignaciones de tipos definidos por el usuario \(Db2 SQL\)](#)

[Tipos y valores de matriz \(Db2 SQL\)](#)

Referencia relacionada

[Constructor de matrices \(Db2 SQL\)](#)

[Función agregada ARRAY_AGG \(SQL Db2 \)](#)

[Función escalar ARRAY_DELETE \(SQL Db2 \)](#)

[Función escalar ARRAY_FIRST \(SQL Db2 \)](#)

[Función escalar ARRAY_NEXT \(Db2 SQL\)](#)

[Función escalar ARRAY_PRIOR \(SQL Db2 \)](#)

[CARDINALITY función escalar \(Db2 SQL\)](#)

[Función escalar MAX_CARDINALITY \(Db2 SQL\)](#)

[Función escalar TRIM_ARRAY \(SQL Db2 \)](#)

Ejemplo de utilización de matrices en un procedimiento de SQL

Un ejemplo muestra muchas de las formas en que puede utilizar matrices en un procedimiento SQL nativo.

El ejemplo muestra cómo:

- Crear un tipo de matriz asociativa.
- Cree un tipo de matriz normal.
- Cree un procedimiento almacenado con matrices como parámetros.
- Definir matrices como variables SQL.
- Utilice la función incorporada ARRAY_AGG en una declaración de cursor para asignar las filas de una tabla de resultados de una sola columna a los elementos de una matriz. Utilice el cursor para recuperar la matriz en un parámetro de salida SQL.
- Utilice un constructor de matriz para inicializar una matriz.
- Asignar una constante o una expresión a un elemento de matriz.
- Utilice la especificación UNNEST para generar la tabla de resultados intermedios a partir de una matriz para una subselección dentro de una instrucción INSERT.
- Utilice la función integrada ARRAY_AGG para asignar las filas de una tabla de resultados de una sola columna a los elementos de una matriz y, a continuación, asigne esa matriz a un parámetro OUT SQL de matriz.
- Utilice la función integrada CARDINALITY para determinar cuántas veces se debe ejecutar un bucle WHILE.
- Utilice un marcador de parámetro para una variable de matriz y un índice de matriz en la cláusula WHERE de una instrucción SELECT.
- Utilice la función integrada ARRAY_AGG en la lista SELECT de una instrucción SELECT INTO y asigne la matriz resultante a un parámetro SQL OUT de matriz.
- Actualizar los valores de las columnas con elementos de matriz.

En este ejemplo, el signo de almohadilla (#) se utiliza como carácter de terminación SQL.

```
--  
-- CREATE ASSOCIATIVE ARRAY TYPES  
--  
CREATE TYPE CHARARRAY AS CHAR(10) ARRAY[VARCHAR(3)]#  
CREATE TYPE BIGINTARRAY AS BIGINT ARRAY[INTEGER]#  
--  
-- CREATE ORDINARY ARRAY TYPES  
--
```

```

CREATE TYPE INTARRAY AS INTEGER ARRAY[100]#
CREATE TYPE STRINGARRAY AS VARCHAR(10) ARRAY[100]#
-- 
-- CREATE TABLES THAT ARE USED IN SQL PROCEDURE PROCESSPERSONS
-- 
CREATE TABLE PERSONS (ID INTEGER, NAME VARCHAR(10))#
CREATE TABLE ARRAYTEST (CHARCOL CHAR(10), INTCOL INT)#
-- SQL PROCEDURE PROCESSPERSONS HAS THREE ARRAY PARAMETERS:
-- OUTSETARRAY IS AN OUT PARAMETER OF ORDINARY ARRAY TYPE STRINGARRAY.
-- OUTSELECTWITHCURSOR IS AN OUT PARAMETER OF ORDINARY ARRAY TYPE STRINGARRAY.
-- OUTSELECTWITHARRAYAGG IS AN OUT PARAMETER OF ORDINARY ARRAY TYPE INTARRAY.
-- 
CREATE PROCEDURE PROCESSPERSONS(OUT OUTSETARRAY STRINGARRAY,
                                INOUT INTO INT,
                                OUT OUTSELECTWITHCURSOR STRINGARRAY,
                                OUT OUTMAXCARDINALITY BIGINT,
                                OUT OUTSELECTWITHARRAYAGG INTARRAY)
ARRAYDEMO: BEGIN
-- DECLARE SQL VARIABLES OF ORDINARY ARRAY TYPES
DECLARE IDS_ORDARRAYVAR INTARRAY;
DECLARE INT_ORDARRAYVAR INTARRAY;
DECLARE NAMES_ORDARRAYVAR STRINGARRAY;
-- DECLARE SQL VARIABLES OF ASSOCIATIVE ARRAY TYPES
DECLARE CHAR_ASSOCARRAYVAR CHARARRAY;
DECLARE BIGINT_ASSOCARRAYVAR BIGINTARRAY;
-- DECLARE SCALAR SQL VARIABLES
DECLARE DECFLOAT_VAR DECFLOAT;
DECLARE BIGINT_VAR BIGINT;
DECLARE SMALLINT_VAR SMALLINT;
DECLARE INT_VAR INT DEFAULT 1;
DECLARE STMT_VAR CHAR(100);
-- DECLARE A CURSOR
DECLARE C2 CURSOR FOR S1;
-- 
-- THE RESULT TABLE OF CURSOR C1 IS AN ARRAY THAT IS POPULATED BY
-- RETRIEVING THE VALUES OF THE NAME COLUMN FROM TABLE PERSONS,
-- ORDERING THE VALUES BY ID, AND USING THE ARRAY_AGG FUNCTION
-- TO ASSIGN THE VALUES TO AN ARRAY.
-- 
DECLARE C1 CURSOR FOR SELECT ARRAY_AGG(NAME ORDER BY ID) FROM PERSONS
    WHERE NAME LIKE 'J%';
-- 
-- USE ARRAY CONSTRUCTORS TO INITIALIZE ARRAYS
-- 
SET IDS_ORDARRAYVAR = ARRAY[5,6,7];
SET NAMES_ORDARRAYVAR = ARRAY['BOB', 'ANN', 'SUE'];
SET CHAR_ASSOCARRAYVAR['001']='1';
SET CHAR_ASSOCARRAYVAR['002']='2';
SET CHAR_ASSOCARRAYVAR['003']='3';
SET CHAR_ASSOCARRAYVAR['004']='4';
SET CHAR_ASSOCARRAYVAR['005']='5';
SET CHAR_ASSOCARRAYVAR['006']='6';
SET INT_ORDARRAYVAR = ARRAY[1,INTEGER(2),3+0,4,5,6] ;
SET BIGINT_ASSOCARRAYVAR[1] = 9;
SET BIGINT_ASSOCARRAYVAR[3] = 10;
SET BIGINT_ASSOCARRAYVAR[5] = 11;
SET BIGINT_ASSOCARRAYVAR[7] = 12;
SET BIGINT_ASSOCARRAYVAR[9] = 13;
-- 
-- ASSIGN A CONSTANT TO AN ARRAY ELEMENT.
-- 
SET IDS_ORDARRAYVAR[4] = 8;
-- 
-- ASSIGN AN EXPRESSION TO AN ARRAY ELEMENT.
-- 
SET IDS_ORDARRAYVAR[5] = 8 * 4 ;
-- 
-- ASSIGN AN ARRAY ELEMENT TO ANOTHER ARRAY ELEMENT. USE AN EXPRESSION
-- TO IDENTIFY THE TARGET ARRAY ELEMENT.
-- 
SET NAMES_ORDARRAYVAR[1+INT_VAR] = NAMES_ORDARRAYVAR[5] ;
-- 
-- POPULATE THE PERSONS TABLE WITH AN INSERT STATEMENT WITH A SUBSELECT:
-- - USE UNNEST TO RETRIEVE VALUES FROM AN ARRAY INTO AN INTERMEDIATE RESULT
--   TABLE.
-- - INSERT THE VALUES FROM THE INTERMEDIATE RESULT TABLE INTO
--   THE PERSONS TABLE.
-- 
INSERT INTO PERSONS(ID, NAME)
    (SELECT T.I, T.N FROM UNNEST(IDS_ORDARRAYVAR, NAMES_ORDARRAYVAR) AS T(I, N));
-- 
-- USE THE ARRAY_AGG FUNCTION TO CREATE AN ARRAY FROM THE RESULT

```

```

-- TABLE OF A SELECT. THEN ASSIGN THAT ARRAY TO AN SQL OUT PARAMETER.
--
SET OUTSETARRAY = (SELECT ARRAY_AGG(NAME ORDER BY ID)
   FROM PERSONS
  WHERE NAME LIKE '%0%');

--
-- USE THE CARDINALITY FUNCTION TO CONTROL THE NUMBER OF TIMES THAT
-- AN INSERT STATEMENT IS EXECUTED TO POPULATE TABLE ARRAYTEST
-- WITH ARRAY ELEMENTS.
--
SET SMALLINT_VAR = 1;
WHILE SMALLINT_VAR <= CARDINALITY(INT_ORDARRAYVAR) DO
  INSERT INTO ARRAYTEST VALUES
    (CHAR_ASSOCARRAYVAR[SMALLINT_VAR],
     INT_ORDARRAYVAR[SMALLINT_VAR]);
  SET SMALLINT_VAR = SMALLINT_VAR+1;
END WHILE;

--
-- DYNAMICALLY EXECUTE AN SQL SELECT STATEMENT WITH A PARAMETER MARKER
-- FOR AN ARRAY, AND A PARAMETER MARKER FOR THE ARRAY INDEX.
--
SET INT_VAR = 3;
SET STMT_VAR =
  'SELECT INTCOL FROM ARRAYTEST WHERE INTCOL = ' ||
  'CAST(?) AS INTARRAY)[?]' ;
PREPARE S1 FROM STMT_VAR;
OPEN C2 USING INT_ORDARRAYVAR, INT_VAR;
FETCH C2 INTO INT0;
CLOSE C2;

--
-- USE A CURSOR TO FETCH AN ARRAY THAT IS CREATED WITH THE ARRAY_AGG FUNCTION
-- INTO AN ARRAY SQL OUT PARAMETER.
--
OPEN C1;
FETCH C1 INTO OUTSELECTWITHCURSOR;
CLOSE C1;

--
-- RETURN THE MAXIMUM CARDINALITY OF AN ARRAY USING THE MAX_CARDINALITY
-- FUNCTION, AND STORE THE VALUE IN AN SQL VARIABLE.
--
SET OUTMAXCARDINALITY = MAX_CARDINALITY(INT_ORDARRAYVAR);

--
-- IN A SELECT INTO STATEMENT, USE THE ARRAY_AGG FUNCTION TO
-- ASSIGN THE VALUES OF COLUMN INTCOL TO ARRAY ELEMENTS, AND ASSIGN
-- THOSE ELEMENTS TO ARRAY OUT PARAMETER OUTSELECTWITHARRAYAGG.
--
SELECT ARRAY_AGG(INTCOL) INTO OUTSELECTWITHARRAYAGG FROM ARRAYTEST;

--
-- IN AN UPDATE STATEMENT, ASSIGN ARRAY ELEMENTS TO COLUMNS.
--
SET SMALLINT_VAR = 1;
WHILE SMALLINT_VAR <= CARDINALITY(INT_ORDARRAYVAR) DO
  UPDATE ARRAYTEST
    SET CHARCOL =
      CHAR_ASSOCARRAYVAR[SMALLINT_VAR], INTCOL = INT_ORDARRAYVAR[SMALLINT_VAR];
  SET SMALLINT_VAR = SMALLINT_VAR +1;
END WHILE;
END#

```

Conceptos relacionados

[Comparaciones de tipos definidos por el usuario \(Db2 SQL\)](#)

[Asignaciones de tipos definidos por el usuario \(Db2 SQL\)](#)

Referencia relacionada

[Constructor de matrices \(Db2 SQL\)](#)

[Función agregada ARRAY_AGG \(SQL Db2\)](#)

[CARDINALITY función escalar \(Db2 SQL\)](#)

[Función escalar MAX_CARDINALITY \(Db2 SQL\)](#)

Creación de una función definida por el usuario

Puede ampliar la funcionalidad SQL de Db2 añadiendo sus propias definiciones de proveedor o de terceros.

Antes de empezar

Configure el entorno para las funciones definidas por el usuario, tal como se describe en [Paso de instalación 21 : Configurar Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario \(Db2 Instalación y migración\)](#).

Acerca de esta tarea

Una función definida por el usuario es un pequeño programa que puede escribir para realizar una operación, similar a un subprograma o función de lenguaje host. No obstante, una función definida por el usuario a menudo es la mejor opción para una aplicación de SQL ya que se puede invocar en una sentencia de SQL. Las funciones definidas por el usuario se crean mediante la instrucción CREATE FUNCTION y se registran en el catálogo Db2 .

Una función definida por el usuario se indica mediante un nombre de función seguido de cero o más operandos entre paréntesis. Al igual que una función incorporada, una función definida por el usuario representa una relación entre un conjunto de valores de entrada y un conjunto de valores de resultado. Los valores de entrada de una función se denominan *parámetros* en la definición de la función. Los valores de entrada de una función se denominan *argumentos* cuando se invoca la función. Por ejemplo, se puede pasar una función con dos argumentos de entrada que tengan tipos de datos de fecha y hora y devolver un valor con un tipo de datos de marca de tiempo como resultado.

Puede crear varios tipos diferentes de funciones definidas por el usuario, incluidas funciones definidas por el usuario externas, SQL y de origen. Las funciones definidas por el usuario también pueden clasificarse como funciones escalares, que devuelven un único valor, o funciones de tabla, que devuelven una tabla. En concreto, puede crear los siguientes tipos de funciones definidas por el usuario:

escalares externas

La función se escribe en un lenguaje de programación y devuelve un valor escalar. La rutina ejecutable externa (paquete) se registra en un servidor de base de datos junto con varios atributos de la función. Cada vez que se invoca la función, el paquete se ejecuta una o más veces. Consulte [Instrucción CREATE FUNCTION \(función escalar externa\) \(Db2 SQL\)](#).

Tabla externa

La función está escrita en un lenguaje de programación. Devuelve una tabla a la subselección desde la que se inició devolviendo una fila cada vez que se inicia la función. La rutina ejecutable externa (paquete) se registra en un servidor de base de datos junto con varios atributos de la función. Cada vez que se invoca la función, el paquete se ejecuta una o más veces. Consulte [Instrucción CREATE FUNCTION \(función de tabla externa\) \(Db2 SQL\)](#).

Derivadas

La función se implementa invocando otra función (ya sea integrada, externa, SQL o de origen) que existe en el servidor. La función hereda los atributos de la función de origen subyacente. Una función de origen no tiene un paquete asociado. Consulte [Instrucción CREATE FUNCTION \(función de origen\) \(Db2 SQL\)](#).

escalares de SQL

La función está escrita exclusivamente en sentencias SQL y devuelve un valor escalar. El cuerpo de una función escalar SQL se escribe en el lenguaje de procedimientos SQL (SQL PL). La función se define en el servidor actual junto con varios atributos de la función.

Db2 admite dos tipos de funciones escalares SQL, en línea y compiladas:

- *Las funciones escalares SQL en línea* contienen una sola instrucción RETURN, que devuelve el valor de una expresión simple. La función no se invoca como parte de una consulta; en su lugar, la expresión de la sentencia RETURN de la función se copia (en línea) en la propia consulta. Por lo tanto, no se genera un paquete para una función escalar SQL en línea.
- *Las funciones escalares SQL compiladas* admiten un conjunto más amplio de funcionalidades, incluidas todas las sentencias SQL PL. Se genera un paquete para una función escalar SQL compilada. Contiene el cuerpo de la función, incluidas las instrucciones de control. También puede contener declaraciones generadas por Db2. Cada vez que se invoca la función, el paquete se ejecuta una o más veces.

Cuando se procesa una instrucción CREATE FUNCTION para una función escalar SQL, Db2 intenta crear una función escalar SQL en línea. Si la función no se puede crear como una función en línea, Db2 intenta crear una función escalar SQL compilada. Para obtener más información sobre la sintaxis y las reglas de estos tipos de funciones, consulte [Instrucción CREATE FUNCTION \(función escalar SQL en línea\) \(Db2 SQL\)](#) y [Instrucción CREATE FUNCTION \(función escalar SQL compilada\) \(Db2 SQL\)](#).

Para determinar qué tipo de función escalar SQL se crea, consulte la columna INLINE de la tabla de catálogo de SYSIBM.SYSROUTINES.

tabla de SQL

La función se escribe exclusivamente como una instrucción SQL RETURN y devuelve un conjunto de filas. El cuerpo de una función de tabla SQL se escribe en el lenguaje de procedimiento SQL. La función se define en el servidor actual junto con varios atributos. La función no se invoca como parte de una consulta. En su lugar, la expresión de la sentencia RETURN de la función se copia (en línea) en la propia consulta. Por lo tanto, no se genera un paquete para una función de tabla SQL. Consulte [Instrucción CREATE FUNCTION \(función de tabla SQL\) \(Db2 SQL\)](#).

El entorno para las funciones definidas por el usuario incluye el espacio de direcciones de la aplicación, desde el cual un programa invoca una función definida por el usuario; un sistema de gestión de memoria (Db2), donde se ejecutan los paquetes de la función definida por el usuario; y un espacio de direcciones establecido por WLM, donde se puede ejecutar la función definida por el usuario; como se muestra en la siguiente figura.

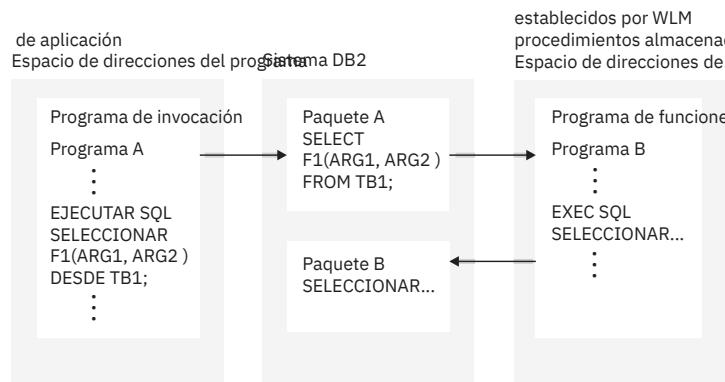


Figura 6. El entorno de funciones definido por el usuario

Para obtener información sobre las funciones definidas por el usuario de Java, consulte [Procedimientos almacenados y funciones definidas por el usuario de Java \(Db2 Application Programming for Java\)](#). Para funciones definidas por el usuario en otros idiomas, consulte las siguientes instrucciones.

Procedimiento

Para crear una función definida por el usuario:

1. Escriba y prepare la función definida por el usuario, como se describe en “[Escritura de una función externa definida por el usuario](#)” en la página 197.

Este paso es necesario solo para una función externa definida por el usuario.

2. Defina la función definida por el usuario en Db2 emitiendo una instrucción CREATE FUNCTION que especifique el tipo de función que desea crear.

Para obtener más información, consulte [Instrucción CREATE FUNCTION \(resumen\) \(Db2 SQL\)](#).

3. Invocar la función definida por el usuario desde una aplicación SQL, como se describe en “[Invocación de una función definida por el usuario](#)” en la página 476.

Definición de una función escalar SQL definida por el usuario

Puede definir una función SQL definida por el usuario para calcular la tangente de un valor utilizando las funciones integradas existentes SIN y COS:

```
CREATE FUNCTION TAN (X DOUBLE)
RETURNS DOUBLE
LANGUAGE SQL
CONTAINS SQL
DETERMINISTIC
RETURN SIN(X)/COS(X);
```

La lógica de la función está contenida en la definición de la función como la siguiente declaración:

```
RETURN SIN(X)/COS(X)
```

Qué hacer a continuación

Si después de definir la función descubre que necesita cambiar una parte de la definición, puede utilizar una instrucción ALTER FUNCTION para cambiar la definición. No puede utilizar ALTER FUNCTION para cambiar algunas de las características de una definición de función definida por el usuario.

Conceptos relacionados

[Funciones definidas por el usuario de ejemplo \(Db2 SQL\)](#)

Tareas relacionadas

[Control de funciones definidas por el usuario \(Db2 Administration Guide\)](#)

Referencia relacionada

[Instrucción CREATE FUNCTION \(resumen\) \(Db2 SQL\)](#)

Funciones externas

Una función externa definida por el usuario es una función que está escrita en un lenguaje de programación. Una función externa se define en la base de datos con una referencia a un programa externo que contiene la lógica que se ejecuta cuando se invoca la función.

Una función externa definida por el usuario que devuelve un único valor es una función escalar. Una función externa definida por el usuario que devuelve una tabla es una función de tabla.

Puede escribir una función externa definida por el usuario en ensamblador, C, C++, COBOL, PL/I o Java. Las funciones definidas por el usuario que están escritas en COBOL pueden incluir extensiones orientadas a objetos, al igual que otros programas COBOL e Db2 . Las funciones definidas por el usuario que están escritas en Java siguen las directrices de codificación y las restricciones específicas de Java. Para obtener información sobre cómo escribir funciones Java definidas por el usuario, consulte [Procedimientos almacenados y funciones definidas por el usuario de Java \(Db2 Application Programming for Java\)](#).

ejemplos

Ejemplo 1: Definición de una función escalar externa definida por el usuario

Un programador desarrolla una función definida por el usuario que busca una cadena de longitud máxima 200 en un valor CLOB cuya longitud máxima es 500 KB. Esta declaración CREATE FUNCTION define la función definida por el usuario:

```
CREATE FUNCTION FINDSTRING (CLOB(500K), VARCHAR(200))
RETURNS INTEGER
CAST FROM FLOAT
SPECIFIC FINDSTRINGCLOB
EXTERNAL NAME 'FINDSTR'
LANGUAGE C
PARAMETER STYLE SQL
NO SQL
DETERMINISTIC
NO EXTERNAL ACTION;
```

La función devuelve un código de estado como un entero. La cláusula CAST FROM se especifica porque la operación de la función da como resultado un valor de punto flotante, y los usuarios esperan un resultado entero para sus sentencias SQL. La función definida por el usuario está escrita en C y no contiene instrucciones SQL.

Supongamos que desea que una función definida por el usuario FINDSTRING funcione en tipos de datos BLOB, así como en tipos CLOB. Puede definir otra instancia de una función definida por el usuario FINDSTRING que especifique un tipo BLOB como entrada:

```
CREATE FUNCTION FINDSTRING (BLOB(500K), VARCHAR(200))
RETURNS INTEGER
CAST FROM FLOAT
SPECIFIC FINDSTRINGBLOB
EXTERNAL NAME 'FNDBLOB'
LANGUAGE C
PARAMETER STYLE SQL
NO SQL
DETERMINISTIC;
```

Cada instancia de FINDSTRING utiliza un programa de aplicación diferente para implementar la lógica de la función definida por el usuario.

Ejemplo 2: Definición de una función escalar externa definida por el usuario

Un programador ha escrito una función definida por el usuario para la división. Es decir, esta función definida por el usuario se invoca cuando un programa de aplicación ejecuta una instrucción utilizando el operador de división (/), como la siguiente instrucción:

```
UPDATE TABLE1 SET INTCOL1= "/" (INTCOL2, INTCOL3);
```

La función definida por el usuario toma dos valores enteros como entrada. El resultado de la función definida por el usuario es de tipo entero. La función definida por el usuario está en el esquema MATH, está escrita en ensamblador y no contiene sentencias SQL. Esta declaración CREATE FUNCTION define la función definida por el usuario:

```
CREATE FUNCTION MATH."/" (INT, INT)
RETURNS INTEGER
SPECIFIC DIVIDE
EXTERNAL NAME 'DIVIDE'
LANGUAGE ASSEMBLE
PARAMETER STYLE SQL
NO SQL
DETERMINISTIC;
```

Ejemplo 3: Definición de una función de tabla externa definida por el usuario

Un programador de aplicaciones desarrolla una función definida por el usuario que recibe dos valores de entrada y devuelve una tabla. Los dos valores de entrada son:

- Una cadena de caracteres de una longitud máxima de 30 que describe un asunto
- Una cadena de caracteres de una longitud máxima de 255 que contiene texto para buscar

La función definida por el usuario escanea los documentos del tema en busca de la cadena de búsqueda y devuelve una lista de documentos que coinciden con los criterios de búsqueda, con un resumen de cada documento. La lista tiene la forma de una tabla de dos columnas. La primera columna es una columna de caracteres de longitud 16 que contiene ID de documentos. La segunda columna es una columna de caracteres variables de una longitud máxima de 5000 que contiene resúmenes de documentos.

La función definida por el usuario está escrita en COBOL, utiliza SQL solo para realizar consultas y siempre produce el mismo resultado para una entrada determinada. La opción CARDINALITY especifica que debe esperar una invocación de la función definida por el usuario para devolver unas 20 filas.

La siguiente instrucción CREATE FUNCTION define la función definida por el usuario:

```
CREATE FUNCTION DOCMATCH (VARCHAR(30), VARCHAR(255))
RETURNS TABLE (DOC_ID CHAR(16), DOC_ABSTRACT VARCHAR(5000))
EXTERNAL NAME 'DOCMTCH'
```

```
LANGUAGE COBOL  
PARAMETER STYLE SQL  
READS SQL DATA  
DETERMINISTIC  
CARDINALITY 20;
```

Funciones escalares SQL

Una función escalar SQL es una función definida por el usuario escrita en SQL y devuelve un único valor cada vez que se invoca. Las funciones escalares SQL contienen el código fuente para la función definida por el usuario en la definición de función definida por el usuario. Hay dos tipos de funciones escalares de SQL, incorporadas y compiladas.

Todas las funciones escalares SQL que se crearon antes de la versión DB2 10 son funciones escalares SQL integradas. Empezando por DB2 10, las funciones escalares SQL pueden crearse como integradas o compiladas.

Db2 determina si una función escalar SQL está en línea o compilada según si la instrucción CREATE FUNCTION que define la función hace uso o no de características mejoradas. Visite [Instrucción CREATE FUNCTION \(función escalar SQL en línea\) \(Db2 SQL\)](#) y [Instrucción CREATE FUNCTION \(función escalar SQL compilada\) \(Db2 SQL\)](#) para obtener más información.

Una función escalar SQL en línea tiene un cuerpo con una sola instrucción RETURN. La sentencia RETURN puede devolver un valor NULL o una expresión simple que no haga referencia a un fullselect escalar. No se generará ningún paquete para una función escalar SQL en línea. Durante la preparación de una instrucción SQL que hace referencia a la función (cuando se invoca la función), la expresión especificada en la instrucción RETURN de la función simplemente se integra en esa instrucción SQL.

Una función escalar SQL compilada puede tener un cuerpo con lógica escrita en lenguaje SQL PL. Puede hacer uso de cualquiera de las características mejoradas para la sentencia CREATE FUNCTION, incluyendo el soporte para el tipo de datos TABLE LOCATOR para parámetros, varias opciones y una sentencia RETURN mejorada que permite la referencia a un fullselect escalar. Se crea un paquete para una función escalar SQL compilada.

Las funciones escalares SQL compiladas incluyen compatibilidad con versiones y gestión de código fuente. Puede utilizar funciones escalares SQL compiladas para las siguientes tareas:

- Definir varias versiones de una función escalar SQL, donde una versión se considera la versión "activa".
- Activar una versión concreta de una función escalar SQL.
- Modificar las opciones de rutina que están asociadas con una versión de una función escalar SQL.
- Definir una nueva versión de una función escalar SQL especificando la misma firma de función que la versión actual, y diferentes opciones de rutina y cuerpo de función.
- Reemplazar la definición de una versión existente especificando la misma firma de función que la versión actual, y diferentes opciones de rutina y cuerpo de función.
- Dejar una versión de una función escalar SQL.
- Volver a una versión anterior sin necesidad de volver a vincular o recompilar explícitamente, activando la versión anterior.

Puede implementar funciones escalares SQL compiladas en varios servidores para permitir que una comunidad más amplia utilice funciones que se han probado exhaustivamente, sin el riesgo de cambiar la lógica en el cuerpo de la rutina. Utilice el depurador de funciones escalares SQL (Unified Debugger) para depurar de forma remota funciones escalares SQL compiladas que se ejecutan en servidores de Db2 for z/OS .

Para preparar una función escalar SQL para su ejecución, ejecute la instrucción CREATE FUNCTION, ya sea de forma estática o dinámica.

Ejemplo: Definición de una función escalar SQL compilada definida por el usuario

El siguiente ejemplo define una función escalar que devuelve el texto de una cadena de entrada, en orden inverso. El ejemplo también explica cómo determinar por qué se permiten varias sentencias SQL en una función escalar SQL compilada.

Una instrucción SQL escalar CREATE FUNCTION compilada contiene *un cuerpo de rutina SQL*, tal como se define en Instrucción CREATE FUNCTION (función escalar SQL compilada) (Db2 SQL). El diagrama de sintaxis para *el cuerpo de la rutina SQL* define el cuerpo de la función como una sola instrucción de control SQL. El diagrama de sintaxis para *la instrucción de control SQL* en Lenguaje de procedimiento de SQL (SQL PL) (Db2 SQL) identifica las instrucciones de control que se pueden especificar, incluida una instrucción RETURN.

Una función SQL puede contener varias sentencias SQL si la sentencia SQL más externa es *una sentencia de control SQL* que incluye otras sentencias SQL. Estas sentencias se definen como sentencias de procedimiento de SQL. El diagrama de sintaxis en Declaración de procedimiento SQL (SQL PL) (Db2 SQL) identifica las sentencias SQL que se pueden especificar dentro de una sentencia de control. Las notas de sintaxis para *la instrucción de procedimiento SQL* aclaran las instrucciones SQL que están permitidas en una función SQL.

```
CREATE FUNCTION REVERSE(INSTR VARCHAR(4000))
RETURNS VARCHAR(4000)
DETERMINISTIC NO EXTERNAL ACTION
CONTAINS SQL
BEGIN A
DECLARE REVSTR, RESTSTR VARCHAR(4000) DEFAULT '';
DECLARE LEN INT; B
IF INSTR IS NULL THEN C
RETURN NULL; D
END IF;
SET (RESTSTR, LEN) = (INSTR, LENGTH(INSTR)); E
WHILE LEN > 0 DO F
SET (REVSTR, RESTSTR, LEN) E
= (SUBSTR(RESTSTR, 1, 1) CONCAT REVSTR,
SUBSTR(RESTSTR, 2, LEN - 1),
LEN - 1);
END WHILE;
RETURN REVSTR; D
END# A
```

La función SQL tiene las siguientes palabras clave y sentencias:

- Las palabras clave BEGIN y END (**A**) indican el principio y el final de una instrucción compuesta.
- Las declaraciones DECLARE (**B**) son componentes de una sentencia compuesta y definen variables SQL dentro de la sentencia compuesta. Para obtener más información sobre las declaraciones compuestas, consulte [sentencia compuesta](#) (Db2 SQL).
- La sentencia IF (**C**), las sentencias RETURN (**D**), y la instrucción WHILE (**F**) son instrucciones de control SQL.
- Las declaraciones de asignación SET (**E**) son instrucciones de control SQL que asignan valores a variables SQL.

Es posible hacer referencia a las variables de SQL en cualquier parte de la sentencia compuesta en la que están declaradas, incluyendo una sentencia SQL que está anidada de forma directa o indirecta dentro de dicha sentencia compuesta. Consulte [Referencias a parámetros y variables SQL en SQL PL](#) (Db2 SQL) para obtener más información.

Tareas relacionadas

[Creación de una función definida por el usuario](#)

Puede ampliar la funcionalidad SQL de Db2 añadiendo sus propias definiciones de proveedor o de terceros.

Referencia relacionada

[Instrucción CREATE FUNCTION \(resumen\)](#) (Db2 SQL)

funciones de tabla de SQL

Una función de tabla SQL es una función que se escribe exclusivamente en sentencias SQL y devuelve una única tabla de resultados.

Una función de tabla SQL puede definir un parámetro como un tipo distinto, definir un parámetro para una tabla de transición (por ejemplo, la TABLE LIKE... Sintaxis AS LOCATOR) e incluir una única instrucción SQL PL RETURN que devuelva una tabla de resultados

La sentencia CREATE para una función de tabla de SQL es una sentencia ejecutable que puede prepararse dinámicamente sólo si el comportamiento de ejecución de DYNAMICRULES se ha especificado implícita o explícitamente.

La sentencia ALTER para una función de tabla de SQL puede incorporarse a un programa de aplicación o bien emitirse de forma interactiva. La sentencia ALTER es una sentencia ejecutable que puede prepararse dinámicamente sólo si el comportamiento de ejecución de DYNAMICRULES se ha especificado implícita o explícitamente.

Función derivada

Una función de origen es una función que invoca otra función que ya existe en el servidor. La función hereda los atributos de la función de origen subyacente. La función de origen puede ser incorporada, externa, SQL o de origen. Las funciones de origen se pueden utilizar para ampliar las funciones de agregación y escalar incorporadas para su uso en tipos distintos.

Puede utilizar funciones de origen para ampliar las funciones integradas existentes u otras funciones definidas por el usuario. Las funciones de origen son útiles para ampliar las funciones agregadas y escalares integradas para su uso en tipos distintos.

Para implementar una función de origen, emita una instrucción CREATE FUNCTION e identifique la función en la que desea basar la función de origen en la cláusula SOURCE.

Ejemplo: Definición de una función definida por el usuario obtenida

Supongamos que necesita una función definida por el usuario que encuentre una cadena en un valor con un tipo distinto de BARCO. BOAT es un tipo distinto basado en un tipo de datos BLOB. La función definida por el usuario FINDSTRINGBLOB ya se ha definido para tomar un tipo de datos BLOB como entrada y realizar la función requerida, pero no se puede invocar con un valor del tipo de datos BOAT. El nombre específico de FINDSTRING es FINDSTRINGBLOB.

Puede definir una función definida por el usuario basada en FINDSTRING para realizar la búsqueda de cadenas en valores de tipo BOAT. Db2 implícitamente convierte el argumento BOAT a un BLOB cuando se invoca la función de origen, FINDSTRING, que acepta un valor BLOB. Esta declaración CREATE FUNCTION define la función definida por el usuario de origen:

```
CREATE FUNCTION FINDSTRING (BOAT, VARCHAR(200))
RETURNS INTEGER
SPECIFIC FINDSTRINGBOAT
SOURCE SPECIFIC FINDSTRINGBLOB;
```

Referencia relacionada

[Instrucción CREATE FUNCTION \(función de origen\) \(Db2 SQL\)](#)

Pasos para crear y utilizar una función definida por el usuario

Una función definida por el usuario es similar a un subprograma o función del lenguaje principal. No obstante, una función definida por el usuario a menudo es la mejor opción para una aplicación de SQL ya que se puede invocar en una sentencia de SQL.

Esta sección contiene información que se aplica a todas las funciones definidas por el usuario e información específica sobre funciones definidas por el usuario en lenguajes distintos de Java.

La creación y el uso de una función definida por el usuario implica estos pasos:

- Configuración del entorno para funciones definidas por el usuario

Probablemente un administrador de sistemas realice este paso. El entorno de funciones definido por el usuario se muestra en la siguiente figura.

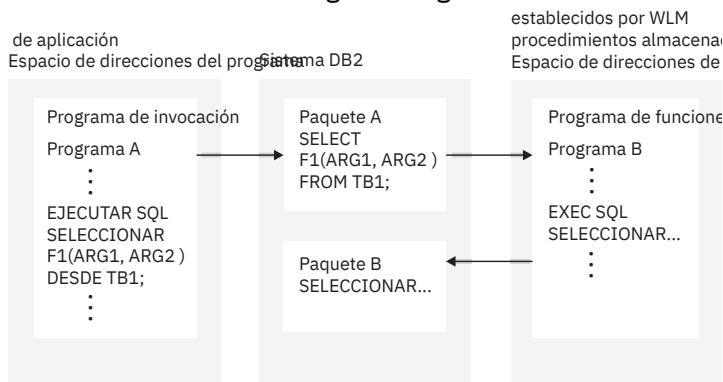


Figura 7. El entorno de funciones definido por el usuario

Contiene un espacio de direcciones de aplicación, desde el cual un programa invoca una función definida por el usuario; un sistema de gestión de memoria (Db2), donde se ejecutan los paquetes de la función definida por el usuario; y un espacio de direcciones establecido por WLM, donde se ejecuta la función definida por el usuario. Los pasos para configurar y mantener el entorno de funciones definido por el usuario son los mismos que para configurar y mantener el entorno de procedimientos almacenados en espacios de direcciones establecidos por WLM.

- Escribir y preparar la función definida por el usuario

Este paso es necesario solo para una función externa definida por el usuario.

La persona que realiza este paso se denomina *implementador de la función definida por el usuario*.

- Definir la función definida por el usuario a Db2

La persona que realiza este paso se denomina *definidor de funciones definidas por el usuario*.

- Invocar la función definida por el usuario desde una aplicación SQL

La persona que realiza este paso se denomina *invocador de funciones definidas por el usuario*.

Conceptos relacionados

[Procedimientos almacenados y funciones definidas por el usuario de Java \(Db2 Application Programming for Java\)](#)

Escritura de una función externa definida por el usuario

Una función externa definida por el usuario se escribe en un lenguaje de programación y es similar a otros programas SQL. Se pueden incluir sentencias de SQL estático o dinámico, llamadas a IFI y mandatos de Db2 que se emiten a través de llamadas a IFI.

Procedimiento

Puede escribir una función externa definida por el usuario en ensamblador, C, C++, COBOL, PL/I o Java.

Las funciones definidas por el usuario que están escritas en COBOL pueden incluir extensiones orientadas a objetos, al igual que otros programas COBOL e Db2. Las funciones definidas por el usuario que están escritas en Java siguen las directrices de codificación y las restricciones específicas de Java.

Su función definida por el usuario también puede acceder a datos remotos mediante el acceso DRDA utilizando las sentencias CONNECT o SET CONNECTION.

Restricciones en programas de funciones definidas por el usuario

Observe estas restricciones cuando escriba una función definida por el usuario:

- Debido a que Db2 utiliza el RRSAF (Resource Recovery Services attachment facility , Formato de respuesta de solicitud de registro) como interfaz con su función definida por el usuario, no debe incluir llamadas RRSAF en su función definida por el usuario. Db2 rechaza cualquier llamada RRSAF que encuentre en una función definida por el usuario.
- Si su función definida por el usuario no está definida con los parámetros SCRATCHPAD o EXTERNAL ACTION, no se garantiza que la función definida por el usuario se ejecute en la misma tarea cada vez que se invoque.
- No puede ejecutar instrucciones COMMIT o ROLLBACK en su función definida por el usuario.
- Debe cerrar todos los curosres que se abrieron dentro de una función escalar definida por el usuario. Db2 devuelve un error SQL si una función escalar definida por el usuario no cierra todos los curosres que abrió antes de completarse.
- Cuando elija el idioma en el que escribir un programa de funciones definido por el usuario, tenga en cuenta las restricciones en cuanto al número de parámetros que se pueden pasar a una rutina en ese idioma. Las funciones de tabla definidas por el usuario en particular pueden requerir un gran número de parámetros. Consulte la guía de programación para el idioma en el que planea escribir la función definida por el usuario para obtener información sobre el número de parámetros que se pueden pasar.
- No puede pasar variables de referencia de archivos LOB como parámetros a funciones definidas por el usuario.
- Las funciones definidas por el usuario no pueden devolver variables de referencia de archivos LOB.
- No se pueden pasar parámetros con el tipo XML a funciones definidas por el usuario. Puede especificar tablas o vistas que contengan columnas XML como parámetros de localización de tablas. Sin embargo, no puede hacer referencia a las columnas XML en el cuerpo de la función definida por el usuario.

Codificar su función definida por el usuario como un programa principal o como un subprograma

Puede codificar su función definida por el usuario como programa principal o como subprograma. La forma en que codifica su programa debe coincidir con la forma en que definió la función definida por el usuario: con el parámetro PROGRAM TYPE MAIN o PROGRAM TYPE SUB. La principal diferencia es que, cuando se inicia un programa principal, el Entorno de lenguaje asigna el almacenamiento del programa de aplicación que utiliza la función externa definida por el usuario. Cuando finaliza un programa principal, el Entorno de idioma cierra los archivos y libera el almacenamiento asignado dinámicamente.

Si codifica su función definida por el usuario como un subprograma y gestiona usted mismo el almacenamiento y los archivos, puede obtener un mejor rendimiento. La función definida por el usuario siempre debe liberar cualquier almacenamiento asignado antes de salir. Para mantener los datos entre invocaciones de la función definida por el usuario, utilice un bloc de notas.

Debe codificar como subprograma una función de tabla definida por el usuario que acceda a recursos externos. Asegúrese también de que el definidor especifique el parámetro EXTERNAL ACTION en la sentencia CREATE FUNCTION o ALTER FUNCTION. Las variables de programa para un subprograma persisten entre las invocaciones de la función definida por el usuario, y el uso del parámetro EXTERNAL ACTION garantiza que la función definida por el usuario permanezca en el mismo espacio de direcciones de una invocación a otra.

Consideraciones de paralelismo

Si el definidor especifica el parámetro ALLOW PARALLEL en la definición de una función escalar definida por el usuario, y la sentencia SQL invocadora se ejecuta en paralelo, la función puede ejecutarse en una tarea paralela. Db2 ejecuta una instancia independiente de la función definida por el usuario para cada tarea paralela. Cuando escriba su programa de funciones, debe comprender cómo interactúan los siguientes valores de parámetros con ALLOW PARALLEL para evitar resultados inesperados:

- SCRATCHPAD

Cuando una instrucción SQL invoca una función definida por el usuario que está definida con el parámetro ALLOW PARALLEL, Db2 asigna un borrador para cada tarea paralela de cada referencia a la función. Esto puede conducir a resultados impredecibles o incorrectos.

Por ejemplo, supongamos que la función definida por el usuario utiliza el bloc de notas para contar el número de veces que se invoca. Si se asigna un bloc de notas para cada tarea paralela, este recuento es el número de invocaciones realizadas por *la tarea paralela* y no para toda la instrucción SQL, que no es el resultado deseado.

- FINAL CALL

Si una función definida por el usuario realiza una acción externa, como enviar una nota, por cada llamada final a la función, se envía una nota por cada tarea paralela en lugar de una sola por la invocación de la función.

- EXTERNAL ACTION

Algunas funciones definidas por el usuario con acciones externas pueden recibir resultados incorrectos si la función se ejecuta mediante tareas paralelas.

Por ejemplo, si la función envía una nota por cada llamada inicial a la función, se envía una nota por cada tarea paralela en lugar de una por la invocación de la función.

- NOT DETERMINISTIC

Una función definida por el usuario que no es determinista puede generar resultados incorrectos si se ejecuta en una tarea paralela.

Por ejemplo, supongamos que ejecuta la siguiente consulta en tareas paralelas:

```
SELECT * FROM T1 WHERE C1 = COUNTER();
```

CONTADOR es una función definida por el usuario que incrementa una variable en el bloc de notas cada vez que se invoca. El contador no es determinista porque la misma entrada no siempre produce la misma salida. La tabla T1 contiene una columna, C1, que tiene los siguientes valores:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Cuando la consulta se ejecuta sin paralelismo, Db2 invoca COUNTER una vez por cada fila de la tabla T1, y hay un borrador para el contador, que Db2 inicializa la primera vez que COUNTER se ejecuta. COUNTER devuelve 1 la primera vez que se ejecuta, 2 la segunda vez, y así sucesivamente. La tabla de resultados de la consulta tiene los siguientes valores:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Ahora supongamos que la consulta se ejecuta con paralelismo y Db2 crea tres tareas paralelas. Db2 ejecuta el predicado WHERE C1 = COUNTER() para cada tarea paralela. Esto significa que cada tarea paralela invoca su propia instancia de la función definida por el usuario y tiene su propio bloc de notas. Db2 inicializa el bloc de notas a cero en la primera llamada a la función definida por el usuario para cada tarea paralela.

Si la tarea paralela 1 procesa las filas 1 a 3, la tarea paralela 2 procesa las filas 4 a 6 y la tarea paralela 3 procesa las filas 7 a 10, se producen los siguientes resultados:

- Cuando se ejecuta la tarea paralela 1, C1 tiene valores 1, 2 y 3, y COUNTER devuelve valores 1, 2 y 3, por lo que la consulta devuelve valores 1, 2 y 3.
- Cuando se ejecuta la tarea paralela 2, C1 tiene valores 4, 5 y 6, pero COUNTER devuelve valores 1, 2 y 3, por lo que la consulta no devuelve filas.
- Cuando se ejecuta la tarea paralela 3, C1 tiene valores 7, 8, 9 y 10, pero COUNTER devuelve valores 1, 2, 3 y 4, por lo que la consulta no devuelve filas.

Por lo tanto, en lugar de devolver las 10 filas que cabría esperar de la consulta, Db2 devuelve solo 3 filas.

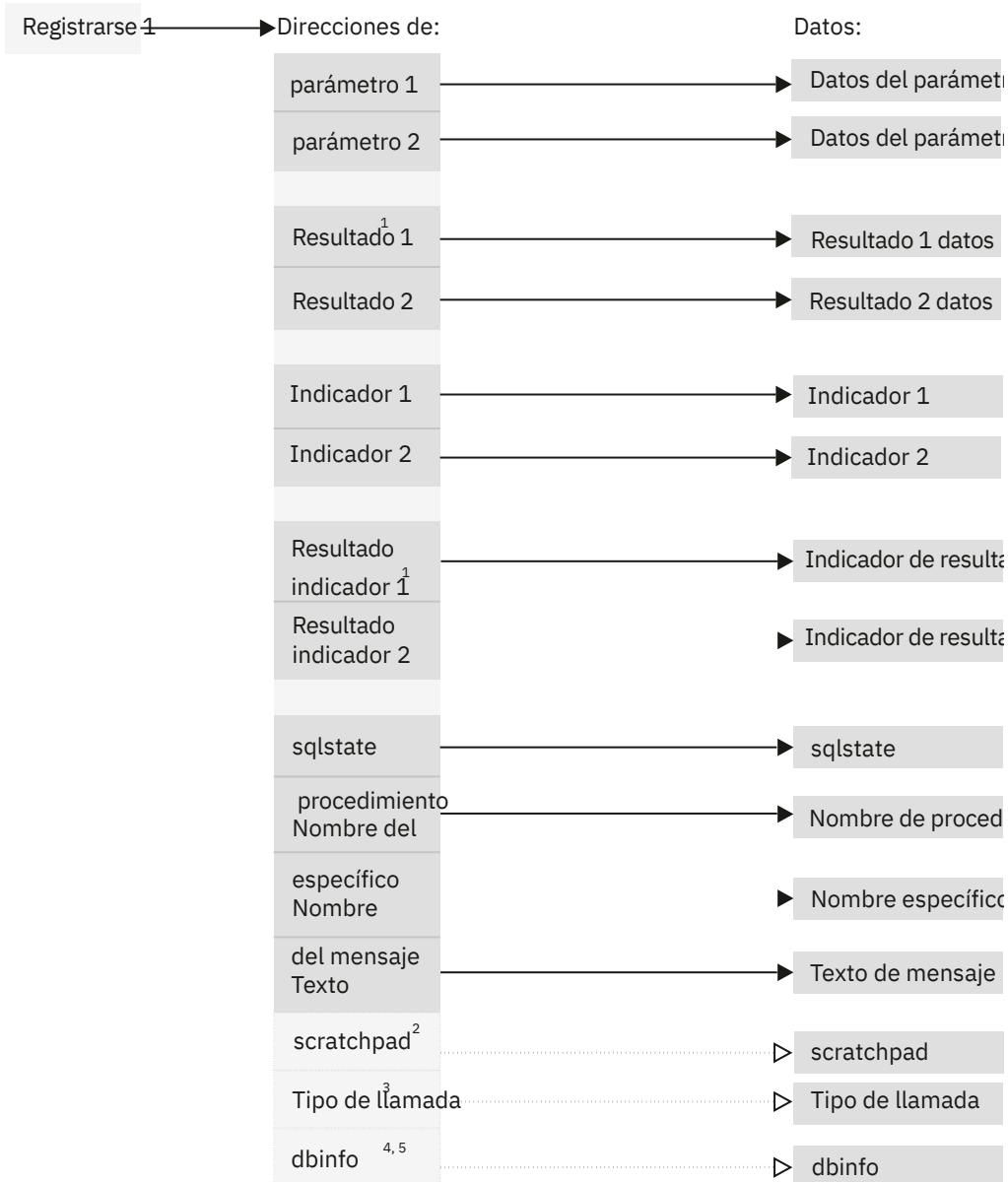
Conceptos relacionados

[Procedimientos almacenados y funciones definidas por el usuario de Java \(Db2 Application Programming for Java\)](#)

Parámetros para funciones externas definidas por el usuario

Para recibir y pasar parámetros a un invocador de una función externa definida por el usuario, debe comprender la estructura de la lista de parámetros. También debe comprender el significado de cada parámetro y si Db2 o su función definida por el usuario establece el valor de cada parámetro.

La siguiente figura muestra la estructura de la lista de parámetros que Db2 pasa a una función definida por el usuario. A continuación se explica cada parámetro.



1. Para una función escalar definida por el usuario, solo se transmiten un resultado y un indicador.
2. Se aprueba si la opción SCRATCHPAD se especifica en la definición de la función definida por el usuario.
3. Se pasa si la opción FINAL CALL se especifica en una definición de función escalar definida por el usuario. Siempre se pasa para una función de tabla definida por el usuario.
4. Para PL/I, este valor es la dirección de un puntero a los datos DBINFO.
5. Se aprueba si la opción DBINFO se especifica en la definición de la función definida por el usuario.

Figura 8. Convenciones de parámetros para una función definida por el usuario

Introducir valores de parámetros

Db2 obtiene los parámetros de entrada de la lista de parámetros del invocador, y su función definida por el usuario recibe esos parámetros de acuerdo con las reglas del lenguaje anfitrión en el que está escrita la función definida por el usuario. El número de parámetros de entrada es el mismo que el número de parámetros en la invocación de la función definida por el usuario. Si uno de los parámetros en la invocación de la función es una expresión, Db2 evalúa la expresión y asigna el resultado de la expresión al parámetro.

Para todos los tipos de datos excepto LOB, ROWID, localizadores y VARCHAR (con el lenguaje C), consulte las tablas enumeradas en “[Compatibilidad de SQL y tipos de datos de lenguaje](#)” en la página 511 para los

tipos de datos de host que son compatibles con los tipos de datos en definiciones de funciones definidas por el usuario.

Para LOB, ROWID y localizadores, consulte las tablas enumeradas en la siguiente tabla para los tipos de datos de host que son compatibles con los tipos de datos en definiciones de funciones definidas por el usuario.

Tabla 43. Lista de tablas de tipos de datos compatibles para LOBS, ROWID y localizadores

Idioma	Ubicación de la tabla de tipos de datos compatibles
Assembler	“Tipos de datos SQL y ensamblador equivalentes” en la página 594
C	“SQL equivalente y tipos de datos C” en la página 644
COBOL	“SQL equivalente y tipos de datos COBOL” en la página 714
PL/I	“SQL equivalente y tipos de datos PL/I” en la página 760

Parámetros de resultado : Establezca estos valores en su función definida por el usuario antes de salir.

Para una función escalar definida por el usuario, devuelve un parámetro de resultado. Para una función de tabla definida por el usuario, se devuelve el mismo número de parámetros que columnas en la cláusula RETURNS TABLE de la sentencia CREATE FUNCTION. Db2 asigna un búfer para cada valor de parámetro de resultado y pasa la dirección del búfer a la función definida por el usuario. Su función definida por el usuario coloca cada valor de parámetro de resultado en su búfer. Debe asegurarse de que la longitud del valor que coloca en cada búfer de salida no exceda la longitud del búfer. Utilice el tipo de datos SQL y la longitud en la instrucción CREATE FUNCTION para determinar la longitud del búfer.

Consulte “Parámetros para funciones externas definidas por el usuario” en la página 200 para determinar el tipo de datos de host que se debe utilizar para cada valor de parámetro de resultado. Si la instrucción CREATE FUNCTION contiene una cláusula CAST FROM, utilice un tipo de datos que se corresponda con el tipo de datos SQL de la cláusula CAST FROM. De lo contrario, utilice un tipo de datos que se corresponda con el tipo de datos SQL en la cláusula RETURNS o RETURNS TABLE.

Para mejorar el rendimiento de las funciones de tabla definidas por el usuario que devuelven muchas columnas, puede pasar valores de un subconjunto de columnas al invocador. Por ejemplo, una función de tabla definida por el usuario puede definirse para devolver 100 columnas, pero el invocador solo necesita valores para dos columnas. Utilice el parámetro DBINFO para indicar a Db2 las columnas para las que devolverá valores. A continuación, devuelva valores solo para esas columnas. Consulte DBINFO para obtener información sobre cómo indicar las columnas de interés.

Indicadores de parámetros de entrada : Son valores SMALLINT, que establece Db2 antes de pasar el control a la función definida por el usuario. Usted utiliza los indicadores para determinar si los parámetros de entrada correspondientes son nulos. El número y el orden de los indicadores son los mismos que el número y el orden de los parámetros de entrada. Al entrar en la función definida por el usuario, cada indicador contiene uno de estos valores:

0

El valor del parámetro de entrada no es nulo.

negativo

El valor del parámetro de entrada es nulo.

Codifique la función definida por el usuario para comprobar todos los indicadores de valores nulos, a menos que la función definida por el usuario esté definida con RETURNS NULL ON NULL INPUT. Una función definida por el usuario con RETURNS NULL ON NULL INPUT se ejecuta solo si todos los parámetros de entrada no son nulos.

Indicadores de resultado : Estos son valores SMALLINT, que debe establecer antes de que finalice la función definida por el usuario para indicar al programa invocador si cada valor del parámetro de resultado es nulo. Una función escalar definida por el usuario tiene un indicador de resultado. Una función de tabla definida por el usuario tiene el mismo número de indicadores de resultados que el número

de parámetros de resultados. El orden de los indicadores de resultados es el mismo que el de los parámetros de resultados. Establezca cada indicador de resultado en uno de estos valores:

0 o positivo

El parámetro result no es nulo.

negativo

El parámetro result es nulo.

Valor SQLSTATE : Este valor CHAR(5) representa el SQLSTATE que se pasa al programa desde el gestor de la base de datos. El valor inicial se establece en «00000». Aunque el programa no suele establecer el SQLSTATE, se puede establecer como el resultado SQLSTATE que se utiliza para devolver un error o una advertencia. Los valores devueltos que comiencen con cualquier otra cifra que no sea «00», «01» o «02» son condiciones de error.

Nombre de función definida por el usuario : Db2 establece este valor en la lista de parámetros antes de que se ejecute la función definida por el usuario. Este valor es VARCHAR(257): 128 bytes para el nombre del esquema, 1 byte para un punto y 128 bytes para el nombre de la función definida por el usuario. Si utiliza el mismo código para implementar varias versiones de una función definida por el usuario, puede utilizar este parámetro para determinar qué versión de la función desea ejecutar el invocador.

Nombre específico : Db2 establece este valor en la lista de parámetros antes de que se ejecute la función definida por el usuario. Este valor es VARCHAR(128) y es el nombre específico de la instrucción CREATE FUNCTION o un nombre específico que genera Db2 . Si utiliza el mismo código para implementar varias versiones de una función definida por el usuario, puede utilizar este parámetro para determinar qué versión de la función desea ejecutar el invocador.

Mensaje de diagnóstico : Su función definida por el usuario puede establecer este valor CHAR o VARCHAR en una cadena de caracteres de hasta 1000 bytes antes de salir. Utilice esta área para transmitir información descriptiva sobre un error o advertencia al solicitante.

Db2 asigna un búfer para esta área y le pasa la dirección del búfer en la lista de parámetros. Al menos los primeros 17 bytes del valor que pones en el búfer aparecen en el campo SQLERRMC del SQLCA que se devuelve al invocador. El número exacto de bytes depende del número de otros tokens en SQLERRMC. No utilice X'FF' en su mensaje de diagnóstico. Db2 utiliza este valor para delimitar tokens.

Scratchpad : Si el definidor especificó SCRATCHPAD en la declaración CREATE FUNCTION, Db2 asigna un búfer para el área de bloc de notas y pasa su dirección a la función definida por el usuario. Antes de que la función definida por el usuario se invoque por primera vez en una instrucción SQL, Db2 establece la longitud del bloc de notas en los primeros 4 bytes del búfer y, a continuación, establece el área del bloc de notas en X'00'. Db2 no reinicializa el bloc de notas entre invocaciones de una subconsulta correlacionada.

Debe asegurarse de que su función definida por el usuario no escriba más bytes en el bloc de notas que la longitud del mismo.

Tipo de llamada : Para una función escalar definida por el usuario, si el definidor especificó FINAL CALL en la sentencia CREATE FUNCTION, Db2 pasa este parámetro a la función definida por el usuario. Para una función de tabla definida por el usuario, Db2 siempre pasa este parámetro a la función definida por el usuario.

Al entrar en *una función escalar definida por el usuario*, el parámetro de tipo de llamada tiene uno de los siguientes valores:

-1

Esta es *la primera llamada* a la función definida por el usuario para la instrucción SQL. Para una primera llamada, todos los parámetros de entrada se pasan a la función definida por el usuario. Además, el bloc de notas, si se asigna, se establece en ceros binarios.

0

Esta es *una llamada normal*. Para una llamada normal, todos los parámetros de entrada se pasan a la función definida por el usuario. Si también se pasa un bloc de notas, Db2 no lo modifica.

1

Esta es *la última llamada*. Para una llamada final, no se pasan parámetros de entrada a la función definida por el usuario. Si también se pasa un bloc de notas, Db2 no lo modifica.

Este tipo de llamada final se produce cuando la aplicación invocadora cierra explícitamente un cursor. Cuando se pasa un valor de 1 a una función definida por el usuario, la función definida por el usuario puede ejecutar sentencias SQL.

255

Esta es *la última llamada*. Para una llamada final, no se pasan parámetros de entrada a la función definida por el usuario. Si también se pasa un bloc de notas, Db2 no lo modifica.

Este tipo de llamada final se produce cuando la aplicación que la invoca ejecuta una instrucción COMMIT o ROLLBACK, o cuando la aplicación que la invoca finaliza de forma anormal. Cuando se pasa un valor de 255 a la función definida por el usuario, la función definida por el usuario no puede ejecutar ninguna instrucción SQL, excepto CLOSE CURSOR. Si la función definida por el usuario ejecuta cualquier instrucción de cierre de cursor durante este tipo de llamada final, la función definida por el usuario debe tolerar SQLCODE -501 porque Db2 podría haber cerrado ya los cursores antes de la llamada final.

Durante la primera llamada, su función escalar definida por el usuario debe adquirir todos los recursos del sistema que necesite. Durante la última llamada, la función escalar definida por el usuario debe liberar todos los recursos que haya adquirido durante la primera llamada. La función escalar definida por el usuario debe devolver un valor de resultado solo durante las llamadas normales. Db2 ignora cualquier resultado que se devuelva durante una llamada final. Sin embargo, la función escalar definida por el usuario puede establecer el SQLSTATE y el área de mensajes de diagnóstico durante la llamada final.

Si una sentencia SQL invocadora contiene más de una función escalar definida por el usuario, y una de esas funciones definidas por el usuario devuelve un error SQLSTATE, Db2 invoca todas las funciones definidas por el usuario para una llamada final, y la sentencia SQL invocadora recibe el SQLSTATE de la primera función definida por el usuario con un error.

Al entrar en *una función de tabla definida por el usuario*, el parámetro de tipo de llamada tiene uno de los siguientes valores:

-2

Esta es *la primera llamada* a la función definida por el usuario para la instrucción SQL. La primera llamada se produce solo si se especifica la palabra clave FINAL CALL en la definición de la función definida por el usuario. Para una primera llamada, todos los parámetros de entrada se pasan a la función definida por el usuario. Además, el bloc de notas, si se asigna, se establece en ceros binarios.

-1

Esta es *la llamada abierta* a la función definida por el usuario mediante una instrucción SQL. Si no se especifica FINAL CALL en la definición de la función definida por el usuario, todos los parámetros de entrada se pasan a la función definida por el usuario, y el bloc de notas, si se ha asignado, se establece en ceros binarios durante la llamada abierta. Si se especifica FINAL CALL para la función definida por el usuario, Db2 no modifica el bloc de notas.

0

Esta es *una llamada de recuperación* a la función definida por el usuario mediante una instrucción SQL. Para una llamada de recuperación, todos los parámetros de entrada se pasan a la función definida por el usuario. Si también se pasa un bloc de notas, Db2 no lo modifica.

1

Esto está *muy reñido*. En caso de error, no se transmiten parámetros de entrada a la función definida por el usuario. Si también se pasa un bloc de notas, Db2 no lo modifica.

2

Esta es *la última llamada*. Este tipo de llamada final solo se produce si se especifica LLAMADA FINAL en la definición de la función definida por el usuario. Para una llamada final, no se pasan parámetros de entrada a la función definida por el usuario. Si también se pasa un bloc de notas, Db2 no lo modifica.

Este tipo de llamada final se produce cuando la aplicación invocadora ejecuta una instrucción CLOSE CURSOR.

255

Esta es *la última llamada*. Para una llamada final, no se pasan parámetros de entrada a la función definida por el usuario. Si también se pasa un bloc de notas, Db2 no lo modifica.

Este tipo de llamada final se produce cuando la aplicación que la invoca ejecuta una instrucción COMMIT o ROLLBACK, o cuando la aplicación que la invoca finaliza de forma anormal. Cuando se pasa un valor de 255 a la función definida por el usuario, la función definida por el usuario no puede ejecutar ninguna instrucción SQL, excepto CLOSE CURSOR. Si la función definida por el usuario ejecuta cualquier instrucción de cierre de cursor durante este tipo de llamada final, la función definida por el usuario debe tolerar SQLCODE -501 porque Db2 podría haber cerrado ya los cursores antes de la llamada final.

Si se define una función de tabla definida por el usuario con FINAL CALL, la función definida por el usuario debe asignar los recursos que necesite durante la primera llamada y liberar esos recursos durante la llamada final que establece un valor de 2.

Si una función de tabla definida por el usuario se define SIN LLAMADA FINAL, la función definida por el usuario debe asignar los recursos que necesite durante la llamada abierta y liberar esos recursos durante la llamada cerrada.

Durante una llamada de recuperación, la función de tabla definida por el usuario debe devolver una fila. Si la función definida por el usuario no tiene más filas que devolver, debe establecer el SQLSTATE en 02000.

Durante la llamada de cierre, una función de tabla definida por el usuario puede establecer el SQLSTATE y el área de mensajes de diagnóstico.

Si se invoca una función de tabla definida por el usuario desde una subconsulta, la función de tabla definida por el usuario recibe una llamada CLOSE por cada invocación de la subconsulta dentro de la consulta de nivel superior, y una llamada OPEN posterior para la siguiente invocación de la subconsulta dentro de la consulta de nivel superior.

DBINFO : Si el definidor especificó DBINFO en la instrucción CREATE FUNCTION, Db2 pasa la estructura DBINFO a la función definida por el usuario. DBINFO contiene información sobre el entorno del llamador de la función definida por el usuario. Contiene los siguientes campos, en el orden mostrado:

Nombre de la ubicación longitud

Un campo entero de 2 bytes sin firmar. Contiene la longitud del nombre de la ubicación en el siguiente campo.

Nombre de la ubicación

Un campo de caracteres de 128 bytes. Contiene el nombre de la ubicación a la que el invocador está conectado actualmente.

Longitud del ID de autorización

Un campo entero de 2 bytes sin firmar. Contiene la longitud del ID de autorización en el siguiente campo.

ID de autorización

Un campo de caracteres de 128 bytes. Contiene el ID de autorización de la aplicación desde la que se invoca la función definida por el usuario, rellenado a la derecha con espacios en blanco. Si esta función definida por el usuario está anidada dentro de otras funciones definidas por el usuario, este valor es el ID de autorización de la aplicación que invocó la función definida por el usuario de nivel más alto.

Página de códigos del subsistema

Una estructura de 48 bytes que consta de 10 campos enteros y un área reservada de ocho bytes. Estos campos proporcionan información sobre los CCSID del subsistema desde el que se invoca la función definida por el usuario.

Longitud del calificador de tabla

Un campo entero de 2 bytes sin firmar. Contiene la longitud del calificador de tabla en el siguiente campo. Si no se utiliza el campo de nombre de la tabla, este campo contiene 0.

Calificador de tabla

Un campo de caracteres de 128 bytes. Contiene el calificador de la tabla que se especifica en el campo de nombre de la tabla.

Longitud del nombre de tabla

Un campo entero de 2 bytes sin firmar. Contiene la longitud del nombre de la tabla en el siguiente campo. Si no se utiliza el campo de nombre de la tabla, este campo contiene 0.

Nombre de tabla

Un campo de caracteres de 128 bytes. Este campo contiene el nombre de la tabla para la operación de actualización o inserción si la referencia a la función definida por el usuario en la instrucción SQL de invocación se encuentra en uno de los siguientes lugares:

- El lado derecho de una cláusula SET en una operación de actualización
- En la lista VALORES de una operación de inserción

De lo contrario, este campo está en blanco.

Longitud del nombre de la columna

Un campo entero de 2 bytes sin firmar. Contiene la longitud del nombre de la columna en el siguiente campo. Si no se pasa ningún nombre de columna a la función definida por el usuario, este campo contiene 0.

Nombre de columna

Un campo de caracteres de 128 bytes. Este campo contiene el nombre de la columna que la operación de actualización o inserción modifica si la referencia a la función definida por el usuario en la instrucción SQL de invocación se encuentra en uno de los siguientes lugares:

- El lado derecho de una cláusula SET en una operación de actualización
- En la lista VALORES de una operación de inserción

De lo contrario, este campo está en blanco.

Información sobre el producto

Un campo de caracteres de 8 bytes que identifica el producto en el que se ejecuta la función definida por el usuario.

El valor del identificador de producto (PRDID) es un valor de carácter de 8 bytes en formato *ppvvrrm*, donde: *ppp* es un código de producto de 3 letras; *vv* es la versión; *rr* es el release; y *m* es el nivel de modificación. En Db2 12 for z/OS, el nivel de modificación indica un rango de niveles de función:

DSN12015 para V12R1M500 o superior.

DSN12010 para V12R1M100.

Para obtener más información, consulte [Valores del identificador de producto \(PRDID\)](#) en la guía de administración de Db2 for z/OS (Db2).

Área reservada

2 bytes.

Sistema operativo

Un campo entero de 4 bytes. Identifica el sistema operativo en el que se ejecuta el programa que invoca la función definida por el usuario. El valor es uno de estos:

0	Desconocido
1	OS/2
3	Windows
4	AIX
5	Windows NT

- 6** HP-UX
- 7** Solaris
- 8** z/OS
- 13** Siemens Nixdorf
- 15** Windows 95
- 16** UNIX
- 18** Linux®
- 19** DYNIX/ptx
- 24** Linux para S/390
- 25** Linux en IBM zSystems
- 26** Linux /IA64
- 27** Linux /PPC
- 28** Linux /PPC64
- 29** Linux /AMD64
- 400** iSeries

Número de entradas en la lista de columnas de la función de tabla

Un campo entero de 2 bytes sin firmar.

Área reservada

26 bytes.

Tabla de funciones de columna de lista de puntero

Si se define una función de tabla, este campo es un puntero a una matriz que contiene 1000 enteros de 2 bytes. Db2 asigna dinámicamente la matriz. Si no se define una función de tabla, este puntero es nulo.

Solo son de interés las primeras n entradas, donde n es el valor en el campo titulado número de entradas en la lista de columnas de la función de tabla. n es mayor o igual que 0 y menor o igual que el número de columnas de resultados definido para la función definida por el usuario en la cláusula RETURNS TABLE de la sentencia CREATE FUNCTION. Los valores corresponden a los números de las columnas que la instrucción de invocación necesita de la función de tabla. Un valor de 1 significa la primera columna de resultados definida, 2 significa la segunda columna de resultados definida, y así sucesivamente. Los valores pueden estar en cualquier orden. Si n es igual a 0, el primer elemento de la matriz es 0. Este es el caso de una sentencia como la siguiente, donde la sentencia de invocación no necesita valores de columna.

```
SELECT COUNT(*) FROM TABLE(TF(...)) AS QQ
```

Esta matriz representa una oportunidad de optimización. La función definida por el usuario no necesita devolver todos los valores para todas las columnas de resultados de la función de tabla. En

su lugar, la función definida por el usuario puede devolver solo aquellas columnas que se necesitan en el contexto particular, que usted identifica por número en la matriz. Sin embargo, si esta optimización complica la lógica de la función definida por el usuario lo suficiente como para anular la ventaja de rendimiento, puede optar por devolver todas las columnas definidas.

Identificador único de la aplicación

Este campo es un puntero a una cadena que identifica de forma única la conexión de la aplicación a Db2. La cadena se regenera para cada conexión a Db2.

La cadena es el LUWID, que consiste en un nombre de red LU totalmente cualificado seguido de un punto y un número de instancia LUW. El nombre de red LU consta de un identificador de red de 1 a 8 caracteres, un punto y un nombre de red LU de 1 a 8 caracteres. El número de instancia LUW consta de 12 caracteres hexadecimales que identifican de forma única la unidad de trabajo.

Área reservada

20 bytes.

Si escribe su función definida por el usuario en C o C++, puede utilizar las declaraciones en el miembro SQLUDF de DSN1210. SDSNC.H para muchos de los parámetros pasados. Para incluir SQLUDF, realice estos cambios en su programa:

- Incluya esta declaración en su código fuente:

```
#include <sqludf.h>
```

- Incluya el conjunto de datos DSN1210. SDSNC.H en la concatenación SYSLIB para el paso del compilador de su trabajo de preparación de programas.
- Especifique las opciones NOMARGINS y NOSEQUENCE en el paso del compilador de su trabajo de preparación del programa.

Ejemplos de parámetros de recepción en una función definida por el usuario:

Los siguientes ejemplos muestran cómo una función definida por el usuario que está escrita en cada uno de los lenguajes de host admitidos recibe la lista de parámetros que es pasada por Db2.

Estos ejemplos suponen que la función definida por el usuario se define con los parámetros SCRATCHPAD, FINAL CALL y DBINFO.

Ensamblador : La siguiente figura muestra las convenciones de parámetros para una función escalar definida por el usuario que se escribe como un programa principal que recibe dos parámetros y devuelve un resultado. Para una función definida por el usuario en lenguaje ensamblador que sea un subprograma, las convenciones son las mismas. En cualquier caso, debe incluir las macros CEEENTRY y CEEEXIT.

```
MYMAIN    CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS
          USING PROGAREA,R13

          L    R7,0(R1)           GET POINTER TO PARM1
          MVC PARM1(4),0(R7)      MOVE VALUE INTO LOCAL COPY OF PARM1
          L    R7,4(R1)           GET POINTER TO PARM2
          MVC PARM2(4),0(R7)      MOVE VALUE INTO LOCAL COPY OF PARM2
          L    R7,12(R1)          GET POINTER TO INDICATOR 1
          MVC F_IND1(2),0(R7)     MOVE PARM1 INDICATOR TO LOCAL STORAGE
          LH   R7,F_IND1         MOVE PARM1 INDICATOR INTO R7
          LTR  R7,R7              CHECK IF IT IS NEGATIVE
          BM   NULLIN             IF SO, PARM1 IS NULL
          L    R7,16(R1)          GET POINTER TO INDICATOR 2
          MVC F_IND2(2),0(R7)     MOVE PARM2 INDICATOR TO LOCAL STORAGE
          LH   R7,F_IND2         MOVE PARM2 INDICATOR INTO R7
          LTR  R7,R7              CHECK IF IT IS NEGATIVE
          BM   NULLIN             IF SO, PARM2 IS NULL
          :
          NULLIN    L    R7,8(R1)           GET ADDRESS OF AREA FOR RESULT
          MVC 0(9,R7),RESULT      MOVE A VALUE INTO RESULT AREA
          L    R7,20(R1)           GET ADDRESS OF AREA FOR RESULT IND
          MVC 0(2,R7),=H'0'        MOVE A VALUE INTO INDICATOR AREA
          :
          CEETERM  RC=0

*****
* VARIABLE DECLARATIONS AND EQUATES
*****
```

```

R1      EQU  1          REGISTER 1
R7      EQU  7          REGISTER 7
PPA     CEEPPA ,        CONSTANTS DESCRIBING THE CODE BLOCK
         LTORG ,
PROGAREA DSECT
         ORG   *+CEEDSASZ  LEAVE SPACE FOR DSA FIXED PART
PARM1   DS   F          PARAMETER 1
PARM2   DS   F          PARAMETER 2
RESULT  DS   CL9        RESULT
F_IND1  DS   H          INDICATOR FOR PARAMETER 1
F_IND2  DS   H          INDICATOR FOR PARAMETER 2
F_INDR  DS   H          INDICATOR FOR RESULT

PROGSIZE EQU  *-PROGAREA
CEEDSA  ,            MAPPING OF THE DYNAMIC SAVE AREA
CEECAA  ,            MAPPING OF THE COMMON ANCHOR AREA
END    MYMAIN

```

C o C++ : Para las funciones definidas por el usuario en C o C++, las convenciones para pasar parámetros son diferentes para los programas principales y los subprogramas.

Para los subprogramas, se pasan los parámetros directamente. Para los programas principales, se utilizan las variables estándar argc y argv para acceder a los parámetros de entrada y salida:

- La variable argv contiene una matriz de punteros a los parámetros que se pasan a la función definida por el usuario. Todos los parámetros de cadena que se devuelven a Db2 deben terminar en nulo.
 - argv[0] contiene la dirección del nombre del módulo de carga para la función definida por el usuario.
 - argv[1] a argv[n] contienen las direcciones de los parámetros 1 a n.
- La variable argc contiene el número de parámetros que se pasan a la función externa definida por el usuario, incluido argv[0].

La siguiente figura muestra las convenciones de parámetros para una función escalar definida por el usuario que se escribe como un programa principal que recibe dos parámetros y devuelve un resultado.

```

#include <stdlib.h>
#include <stdio.h>

main(argc,argv)
  int argc;
  char *argv[];
{
  /*****
  /* Assume that the user-defined function invocation*/
  /* included 2 input parameters in the parameter */
  /* list. Also assume that the definition includes */
  /* the SCRATCHPAD, FINAL CALL, and DBINFO options, */
  /* so DB2 passes the scratchpad, calltype, and */
  /* dbinfo parameters. */
  /* The argv vector contains these entries: */
  /*     argv[0]      1  load module name */
  /*     argv[1-2]     2  input parms */
  /*     argv[3]      1  result parm */
  /*     argv[4-5]     2  null indicators */
  /*     argv[6]      1  result null indicator */
  /*     argv[7]      1  SQLSTATE variable */
  /*     argv[8]      1  qualified func name */
  /*     argv[9]      1  specific func name */
  /*     argv[10]     1  diagnostic string */
  /*     argv[11]     1  scratchpad */
  /*     argv[12]     1  call type */
  /*     argv[13]     + 1 dbinfo */
  /*               ----- */
  /*               14  for the argc variable */
  *****/
  if argc<>14
  {
  :
  /*****
  /* This section would contain the code executed if the */
  /* user-defined function is invoked with the wrong number */
  /* of parameters. */
  *****/
}

```

```

/*****************/
/* Assume the first parameter is an integer.      */
/* The following code shows how to copy the integer*/
/* parameter into the application storage.        */
/*****************/
int parm1;
parm1 = *(int *) argv[1];

/*****************/
/* Access the null indicator for the first      */
/* parameter on the invoked user-defined function */
/* as follows:                                     */
/*****************/
short int ind1;
ind1 = *(short int *) argv[4];

/*****************/
/* Use the following expression to assign          */
/* 'xxxxx' to the SQLSTATE returned to caller on   */
/* the SQL statement that contains the invoked     */
/* user-defined function.                         */
/*****************/
strcpy(argv[7],"xxxxx");

/*****************/
/* Obtain the value of the qualified function      */
/* name with this expression.                     */
/*****************/
char f_func[28];
strcpy(f_func,argv[8]);

/*****************/
/* Obtain the value of the specific function       */
/* name with this expression.                     */
/*****************/
char f_spec[19];
strcpy(f_spec,argv[9]);

/*****************/
/* Use the following expression to assign          */
/* 'yyyyyyy' to the diagnostic string returned    */
/* in the SQLCA associated with the invoked       */
/* user-defined function.                         */
/*****************/
strcpy(argv[10],"yyyyyyy");

/*****************/
/* Use the following expression to assign the      */
/* result of the function.                        */
/*****************/
char l_result[11];
strcpy(argv[3],l_result);

:
}

```

La siguiente figura muestra las convenciones de parámetros para una función escalar definida por el usuario escrita como un subprograma C que recibe dos parámetros y devuelve un resultado.

```

#pragma runopts(plist(os))
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sqludf.h>

void myfunc(long *parm1, char parm2[11], char result[11],
            short *f_ind1, short *f_ind2, short *f_indr,
            char udf_sqlstate[6], char udf_fname[138],
            char udf_specname[129], char udf_mgtext[71],
            struct sqludf_scratchpad *udf_scratchpad,
            long *udf_call_type,
            struct sql_dbinfo *udf_dbinfo);
{
    /*************************************************************************/
    /* Declare local copies of parameters                                */
    /*************************************************************************/
    int l_p1;
    char l_p2[11];
    short int l_ind1;
    short int l_ind2;

```

```

char ludf_sqlstate[6];      /* SQLSTATE          */
char ludf_fname[138];       /* function name    */
char ludf_specname[129];    /* specific function name */
char ludf_msgrtext[71]      /* diagnostic message text*/
sqludf_scratchpad *ludf_scratchpad; /* scratchpad      */
long *ludf_call_type;       /* call type        */
sqludf_dbinfo *ludf_dbinfo /* dbinfo           */
/*****************************************/
/* Copy each of the parameters in the parameter */
/* list into a local variable to demonstrate   */
/* how the parameters can be referenced.       */
/*****************************************/

l_p1 = *parm1;
strcpy(l_p2,parm2);
l_ind1 = *f_ind1;
l_ind1 = *f_ind2;
strcpy(ludf_sqlstate,udf_sqlstate);
strcpy(ludf_fname,udf_fname);
strcpy(ludf_specname,udf_specname);
l_udf_call_type = *udf_call_type;
strcpy(ludf_msgrtext,udf_msgrtext);
memcpy(&ludf_scratchpad,udf_scratchpad,sizeof(ludf_scratchpad));
memcpy(&ludf_dbinfo,udf_dbinfo,sizeof(ludf_dbinfo));
:
}

```

La siguiente figura muestra las convenciones de parámetros para una función escalar definida por el usuario que se escribe como un subprograma C++ que recibe dos parámetros y devuelve un resultado. Este ejemplo demuestra que debe utilizar un modificador externo "C" para indicar que desea que el subprograma C++ reciba parámetros de acuerdo con la convención de vinculación C. Db2 Este modificador es necesario porque la interfaz CEEPIPI CALL_SUB, que utiliza PHP para llamar a la función definida por el usuario, pasa parámetros utilizando la convención de vinculación C.

```

#pragma runopts(plist(os))
#include <stdlib.h>
#include <stdio.h>
#include <sqludf.h>

extern "C" void myfunc(long *parm1, char parm2[11],
                      char result[11], short *f_ind1, short *f_ind2, short *f_indr,
                      char udf_sqlstate[6], char udf_fname[138],
                      char udf_specname[129], char udf_msgrtext[71],
                      struct sqludf_scratchpad *udf_scratchpad,
                      long *udf_call_type,
                      struct sql_dbinfo *udf_dbinfo);

{
/*****************************************/
/* Define local copies of parameters. */
/*****************************************/
int l_p1;
char l_p2[11];
short int l_ind1;
short int l_ind2;
char ludf_sqlstate[6];      /* SQLSTATE          */
char ludf_fname[138];       /* function name    */
char ludf_specname[129];    /* specific function name */
char ludf_msgrtext[71]      /* diagnostic message text*/
sqludf_scratchpad *ludf_scratchpad; /* scratchpad      */
long *ludf_call_type;       /* call type        */
sqludf_dbinfo *ludf_dbinfo /* dbinfo           */
/*****************************************/
/* Copy each of the parameters in the parameter */
/* list into a local variable to demonstrate   */
/* how the parameters can be referenced.       */
/*****************************************/
l_p1 = *parm1;
strcpy(l_p2,parm2);
l_ind1 = *f_ind1;
l_ind1 = *f_ind2;
strcpy(ludf_sqlstate,udf_sqlstate);
strcpy(ludf_fname,udf_fname);
strcpy(ludf_specname,udf_specname);
l_udf_call_type = *udf_call_type;
strcpy(ludf_msgrtext,udf_msgrtext);
memcpy(&ludf_scratchpad,udf_scratchpad,sizeof(ludf_scratchpad));
memcpy(&ludf_dbinfo,udf_dbinfo,sizeof(ludf_dbinfo));

```

COBOL : La siguiente figura muestra las convenciones de parámetros para una función de tabla definida por el usuario que se escribe como un programa principal que recibe dos parámetros y devuelve dos resultados. Para una función definida por el usuario COBOL que sea un subprograma, las convenciones son las mismas.

```
CBL APOST,RES,RENT
IDENTIFICATION DIVISION.
:
DATA DIVISION.
:
LINKAGE SECTION.
*****
* Declare each of the parameters *
*****
01 UDFPARM1 PIC S9(9) USAGE COMP.
01 UDFPARM2 PIC X(10).
:
*****
* Declare these variables for result parameters *
*****
01 UDFRESULT1 PIC X(10).
01 UDFRESULT2 PIC X(10).
:
*****
* Declare a null indicator for each parameter *
*****
01 UDF-IND1 PIC S9(4) USAGE COMP.
01 UDF-IND2 PIC S9(4) USAGE COMP.
:
*****
* Declare a null indicator for result parameter *
*****
01 UDF-RIND1 PIC S9(4) USAGE COMP.
01 UDF-RIND2 PIC S9(4) USAGE COMP.
:
*****
* Declare the SQLSTATE that can be set by the *
* user-defined function *
*****
01 UDF-SQLSTATE PIC X(5).
*****
* Declare the qualified function name *
*****
01 UDF-FUNC.
49 UDF-FUNC-LEN PIC 9(4) USAGE BINARY.
49 UDF-FUNC-TEXT PIC X(137).
*****
* Declare the specific function name *
*****
01 UDF-SPEC.
49 UDF-SPEC-LEN PIC 9(4) USAGE BINARY.
49 UDF-SPEC-TEXT PIC X(128).
*****
* Declare SQL diagnostic message token *
*****
01 UDF-DIAG.
49 UDF-DIAG-LEN PIC 9(4) USAGE BINARY.
49 UDF-DIAG-TEXT PIC X(1000).

*****
* Declare the scratchpad *
*****
01 UDF-SCRATCHPAD.
49 UDF-SPAD-LEN PIC 9(9) USAGE BINARY.
49 UDF-SPAD-TEXT PIC X(100).
*****
* Declare the call type *
*****
01 UDF-CALL-TYPE PIC 9(9) USAGE BINARY.
*****
* CONSTANTS FOR DB2-EBCODING-SCHEME. *
*****
77 SQLUDF-ASCII PIC 9(9) VALUE 1.
77 SQLUDF-EBCDIC PIC 9(9) VALUE 2.
```

```

77 SQLUDF-UNICODE PIC 9(9) VALUE 3.
*****
* Structure used for DBINFO
*****
01 SQLUDF-DBINFO.
*   location name length
  05 DBNAMELEN PIC 9(4) USAGE BINARY.
*   location name
  05 DBNAME PIC X(128).
*   authorization ID length
  05 AUTHIDLEN PIC 9(4) USAGE BINARY.
*   authorization ID
  05 AUTHID PIC X(128).
*   environment CCSID information
  05 CODEPG PIC X(48).
  05 CDPG-DB2 REDEFINES CODEPG.
  10 DB2-CCSID OCCURS 3 TIMES.
    15 DB2-SBCS  PIC 9(9) USAGE BINARY.
    15 DB2-DBCS  PIC 9(9) USAGE BINARY.
    15 DB2-MIXED PIC 9(9) USAGE BINARY.
    10 ENCODING-SCHEME PIC 9(9) USAGE BINARY.
    10 RESERVED   PIC X(8).
* other platform-specific deprecated CCSID structures not included here
* schema name length
  05 TBSCHAMELEN PIC 9(4) USAGE BINARY.
* schema name
  05 TBSHEMA PIC X(128).
* table name length
  05 TBNAMELEN PIC 9(4) USAGE BINARY.
* table name
  05 TBNAME PIC X(128).
* column name length
  05 COLNAMELEN PIC 9(4) USAGE BINARY.
* column name
  05 COLNAME PIC X(128).
* product information
  05 VER-REL PIC X(8).
* reserved for expansion
  05 RESD0 PIC X(2).
* platform type
  05 PLATFORM PIC 9(9) USAGE BINARY.
* number of entries in tfcolumn list array (tfcolumn, below)
  05 NUMTFCOL PIC 9(4) USAGE BINARY.

*   reserved for expansion
  05 RESD1 PIC X(26).
*   tfcolumn will be allocated dynamically if TF is defined
*   otherwise this will be a null pointer
  05 TFCOLUMN USAGE IS POINTER.
*   Application identifier
  05 APP-ID USAGE IS POINTER.
*   reserved for expansion
  05 RESD2 PIC X(20).

*
PROCEDURE DIVISION USING UDFPARM1, UDFPARM2, UDFRESULT1,
                      UDFRESULT2, UDF-IND1, UDF-IND2,
                      UDF-RIND1, UDF-RIND2,
                      UDF-SQLSTATE, UDF-FUNC, UDF-SPEC,
                      UDF-DIAG, UDF-SCRATCHPAD,
                      UDF-CALL-TYPE, SQLUDF-DBINFO.

```

PL/I: La siguiente figura muestra las convenciones de parámetros para una función escalar definida por el usuario que se escribe como un programa principal que recibe dos parámetros y devuelve un resultado. Para una función definida por el usuario PL/I que sea un subprograma, las convenciones son las mismas.

```

*PROCESS SYSTEM(MVS);
MYMAIN: PROC(UDF_PARM1, UDF_PARM2, UDF_RESULT,
            UDF_IND1, UDF_IND2, UDF_INDR,
            UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
            UDF_DIAG_MSG, UDF_SCRATCHPAD,
            UDF_CALL_TYPE, UDF_DBINFO)
OPTIONS(MAIN NOEXECOPS REENTRANT);

DCL UDF_PARM1 BIN FIXED(31);      /* first parameter */
DCL UDF_PARM2 CHAR(10);          /* second parameter */
DCL UDF_RESULT CHAR(10);          /* result parameter */
DCL UDF_IND1 BIN FIXED(15);       /* indicator for 1st parm */
DCL UDF_IND2 BIN FIXED(15);       /* indicator for 2nd parm */

```

```

DCL UDF_INDR BIN FIXED(15);      /* indicator for result      */
DCL UDF_SQLSTATE CHAR(5);        /* SQLSTATE returned to DB2   */
DCL UDF_NAME CHAR(137) VARYING; /* Qualified function name    */
DCL UDF_SPEC_NAME CHAR(128) VARYING; /* Specific function name   */
DCL UDF_DIAG_MSG CHAR(70) VARYING; /* Diagnostic string          */
DCL 01 UDF_SCRATCHPAD           /* Scratchpad                 */
  03 UDF_SPAD_LEN BIN FIXED(31),
  03 UDF_SPAD_TEXT CHAR(100);
DCL UDF_CALL_TYPE BIN FIXED(31); /* Call Type                  */
DCL DBINFO PTR;
/* CONSTANTS FOR DB2_ENCODING_SCHEME */
DCL SQLUDF_ASCII BIN FIXED(15) INIT(1);
DCL SQLUDF_EBCDIC BIN FIXED(15) INIT(2);
DCL SQLUDF_MIXED BIN FIXED(15) INIT(3);

DCL 01 UDF_DBINFO BASED(DBINFO),           /* Dbinfo                    */
  03 UDF_DBINFO_LLEN BIN FIXED(15),        /* location length          */
  03 UDF_DBINFO_LOC CHAR(128),             /* location name            */
  03 UDF_DBINFO_ALEN BIN FIXED(15),        /* auth ID length          */
  03 UDF_DBINFO_AUTH CHAR(128),            /* authorization ID        */
  03 UDF_DBINFO_CDPG,                      /* environment CCSID info */
  05 DB2_CCSIDS(3),
    07 R1          BIN FIXED(15), /* Reserved                */
    07 DB2_SBCS   BIN FIXED(15), /* SBCS CCSID              */
    07 R2          BIN FIXED(15), /* Reserved                */
    07 DB2_DBCS   BIN FIXED(15), /* DBCS CCSID              */
    07 R3          BIN FIXED(15), /* Reserved                */
    07 DB2_MIXED   BIN FIXED(15), /* MIXED CCSID             */
  05 DB2_ENCODING_SCHEME BIN FIXED(31),
  05 DB2_CCSID_RESERVED CHAR(8),
  03 UDF_DBINFO_SLEN BIN FIXED(15), /* schema length           */
  03 UDF_DBINFO_SCHEMA CHAR(128),          /* schema name              */
  03 UDF_DBINFO_TLEN BIN FIXED(15), /* table length             */
  03 UDF_DBINFO_TABLE CHAR(128),           /* table name               */
  03 UDF_DBINFO_CLEN BIN FIXED(15), /* column length            */
  03 UDF_DBINFO_COLUMN CHAR(128),          /* column name              */
  03 UDF_DBINFO_RELVER CHAR(8),            /* DB2 release level       */
  03 UDF_DBINFO_RESERVED CHAR(2),          /* reserved                */
  03 UDF_DBINFO_PLATFORM BIN FIXED(31), /* database platform        */
  03 UDF_DBINFO_NUMTFCOL BIN FIXED(15), /* # of TF columns used   */
  03 UDF_DBINFO_RESERVED1 CHAR(26),         /* reserved                */
  03 UDF_DBINFO_TFCOLUMN PTR,             /* -> TFcolumn list       */
  03 UDF_DBINFO_APPLID PTR,              /* -> application id     */
  03 UDF_DBINFO_RESERVED2 CHAR(20);        /* reserved                */
:

```

Referencia relacionada

[Instrucción CREATE FUNCTION \(función escalar externa\) \(Db2 SQL\)](#)

Hacer que una función definida por el usuario sea reentrante

Una función reentrante definida por el usuario es una función para la que dos o más procesos pueden utilizar simultáneamente una única copia de la función.

Procedimiento

Se recomienda compilar y editar el enlace de su función definida por el usuario como reentrante. (Para un programa ensamblador, también debe codificar la función definida por el usuario para que sea reentrante)

Las funciones reentrantes definidas por el usuario tienen las siguientes ventajas:

- El sistema operativo no necesita cargar la función definida por el usuario en el almacenamiento cada vez que se llama a la función definida por el usuario.
- Varias tareas en un espacio de direcciones de procedimientos almacenados establecidos por WLM pueden compartir una única copia de la función definida por el usuario. Esto disminuye la cantidad de almacenamiento virtual que se necesita para el código en el espacio de direcciones.

Si su función definida por el usuario consta de varios programas, debe vincular cada programa que contenga instrucciones SQL en un paquete independiente. El definidor de la función definida por el usuario debe tener autoridad EXECUTE para todos los paquetes que forman parte de la función definida por el usuario.

Cuando el programa principal de una función definida por el usuario llama a otro programa, Db2 utiliza el registro especial CURRENT PACKAGE PATH para determinar la lista de colecciones en las que buscar el paquete del programa llamado. El programa principal puede cambiar este ID de colección ejecutando la instrucción SET CURRENT PACKAGE PATH.

Si el valor de CURRENT PACKAGE PATH está en blanco o es una cadena vacía, Db2 utiliza el registro especial CURRENT PACKAGESET para determinar la colección en la que buscar el paquete del programa llamado. El programa principal puede cambiar este valor ejecutando la instrucción SET CURRENT PACKAGESET.

Si los registros especiales CURRENT PACKAGE PATH y CURRENT PACKAGESET contienen un valor en blanco, Db2 utiliza el método descrito en “Enlace de un plan de aplicación” en la página 933 para buscar el paquete.

Registros especiales en una función definida por el usuario o un procedimiento almacenado

Puede utilizar todos los registros especiales en una función definida por el usuario o un procedimiento almacenado. Sin embargo, puede modificar solo algunos de estos registros especiales.

Después de que se complete una función definida por el usuario o un procedimiento almacenado, Db2 restaura todos los registros especiales a los valores que tenían antes de la invocación.

La siguiente tabla muestra la información que necesita cuando utiliza registros especiales en una función definida por el usuario o un procedimiento almacenado.

Tabla 44. Características de los registros especiales en una función definida por el usuario o un procedimiento almacenado

Registro especial	Valor inicial cuando se especifica la opción HEREDAR REGISTROS ESPECIALES	Valor inicial cuando se especifica la opción REGISTROS ESPECIALES POR DEFECTO	¿Puede Routine utilizar la instrucción SET para modificar?
CURRENT ACCELERATOR	Heredado de la aplicación invocadora ⁶ ; de lo contrario, no se utiliza ningún acelerador preferido y Db2 determinará el acelerador de destino	El valor de la opción de enlace ACCELERATOR si se especifica para el paquete de funciones definidas por el usuario o de procedimientos almacenados; de lo contrario, no se utiliza ningún acelerador preferido y Db2 determinará el acelerador de destino	Sí
CURRENT APPLICATION COMPATIBILITY	El valor de la opción de enlace APPLCOMPAT para la función definida por el usuario o el paquete de procedimientos almacenados	El valor de la opción de enlace APPLCOMPAT para la función definida por el usuario o el paquete de procedimientos almacenados	Sí
CURRENT APPLICATION ENCODING SCHEME	El valor de la opción de enlace ENCODING para la función definida por el usuario o el paquete de procedimientos almacenados	El valor de la opción de enlace ENCODING para la función definida por el usuario o el paquete de procedimientos almacenados	Sí
CURRENT CLIENT_ACCTNG	Heredado de la aplicación de invocación	Heredado de la aplicación de invocación	No aplicable ⁵

Tabla 44. Características de los registros especiales en una función definida por el usuario o un procedimiento almacenado (continuación)

Registro especial	Valor inicial cuando se especifica la opción HEREDAR REGISTROS ESPECIALES	Valor inicial cuando se especifica la opción REGISTROS ESPECIALES POR DEFECTO	¿Puede Routine utilizar la instrucción SET para modificar?
CURRENT CLIENT_APPLNAME	Heredado de la aplicación de invocación	Heredado de la aplicación de invocación	No aplicable ⁵
CURRENT CLIENT_USERID	Heredado de la aplicación de invocación	Heredado de la aplicación de invocación	No aplicable ⁵
CURRENT CLIENT_WRKSTNNNAME	Heredado de la aplicación de invocación	Heredado de la aplicación de invocación	No aplicable ⁵
CURRENT DATE	Nuevo valor para cada instrucción SQL en el ^{pa} quete de funciones definidas por el usuario o procedimientos almacenados ¹	Nuevo valor para cada instrucción SQL en el ^{pa} quete de funciones definidas por el usuario o procedimientos almacenados ¹	No aplicable ⁵
CURRENT DEBUG MODE	Heredado de la aplicación de invocación	DISALLOW	Sí
CURRENT DECFLOAT ROUNDING MODE	Heredado de la aplicación de invocación	El valor de la opción de enlace REDONDEAR para el paquete de funciones definidas por el usuario o procedimientos almacenados	Sí
CURRENT DEGREE	GRADO ACT UAL2	El valor del campo CURRENT DEGREE en el panel de instalación DSNTIP8	Sí
CURRENT EXPLAIN MODE	Heredado de la aplicación de invocación	NEE	Sí
GET_ACCEL_ARCHIVE actual	Heredado de la aplicación invocadora ⁶ ; de lo contrario, se utilizará el valor del parámetro del subsistema	El valor de la opción de enlace GETACCELARCHIVE si se especifica para el paquete de funciones definidas por el usuario o de procedimientos almacenados; de lo contrario, se utilizará el valor del parámetro del subsistema	Sí
CURRENT LOCALE LC_CTYPE	Heredado de la aplicación de invocación	El valor del campo CURRENT LC_CTYPE en el panel de instalación DSNTIPF	Sí
CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION	Heredado de la aplicación de invocación	Valor predeterminado del sistema	Sí
CURRENT MEMBER	Nuevo valor para cada instrucción SET <i>host-variable</i> =CURRENT MEMBER	Nuevo valor para cada instrucción SET <i>host-variable</i> =CURRENT MEMBER	No aplicable ⁵

Tabla 44. Características de los registros especiales en una función definida por el usuario o un procedimiento almacenado (continuación)

Registro especial	Valor inicial cuando se especifica la opción HEREDAR REGISTROS ESPECIALES	Valor inicial cuando se especifica la opción REGISTROS ESPECIALES POR DEFECTO	¿Puede Routine utilizar la instrucción SET para modificar?
CURRENT OPTIMIZATION HINT	El valor de la opción de enlace OPTHINT para el paquete de funciones definidas por el usuario o procedimiento almacenado o heredado de la aplicación que lo invoca ⁶	El valor de la opción de enlace OPTHINT para la función definida por el usuario o el paquete de procedimientos almacenados	Sí
CURRENT PACKAGE PATH	Una cadena vacía si la rutina se definió con un valor COLLID; de lo contrario, se hereda de la aplicación que invoca ⁴	Una cadena vacía, independientemente de si se especificó un valor COLLID para la rutina ⁴	Sí
CURRENT PACKAGESET	Heredado de la aplicación invocadora ³	Heredado de la aplicación invocadora ³	Sí
CURRENT PATH	El valor de la opción de enlace PATH para el paquete de funciones definidas por el usuario o procedimiento almacenado o heredado de la aplicación que lo invoca ⁶	El valor de la opción de enlace PATH para la función definida por el usuario o el paquete de procedimientos almacenados	Sí
CURRENT PRECISION	Heredado de la aplicación de invocación	El valor del campo ARITMÉTICA DECIMAL en el panel de instalación DSNTIP4	Sí
CURRENT QUERY ACCELERATION	Heredado de la aplicación invocadora ⁶ ; de lo contrario, se utilizará el valor del parámetro del subsistema	El valor de la opción de enlace QUERYACCELERATION si se especifica para el paquete de funciones definidas por el usuario o de procedimientos almacenados; de lo contrario, se utilizará el valor del parámetro del subsistema	Sí
CURRENT QUERY ACCELERATION WAITFORDATA	Heredado de la aplicación invocadora ⁶ ; de lo contrario, se utilizará el valor del parámetro del subsistema	El valor de la opción ACCELERATIONWAITFORDATA si se especifica para el paquete de funciones definidas por el usuario o de procedimientos almacenados; de lo contrario, se utilizará el valor del parámetro del subsistema	Sí
CURRENT REFRESH AGE	Heredado de la aplicación de invocación	Valor predeterminado del sistema	Sí
CURRENT ROUTINE VERSION	Heredado de la aplicación de invocación	La cadena vacía	Sí

Tabla 44. Características de los registros especiales en una función definida por el usuario o un procedimiento almacenado (continuación)

Registro especial	Valor inicial cuando se especifica la opción HEREDAR REGISTROS ESPECIALES	Valor inicial cuando se especifica la opción REGISTROS ESPECIALES POR DEFECTO	¿Puede Routine utilizar la instrucción SET para modificar?
CURRENT RULES	Heredado de la aplicación de invocación	El valor de la opción de enlace SQLRULES para el plan que invoca una función definida por el usuario o un procedimiento almacenado	Sí
CURRENT SCHEMA	Heredado de la aplicación de invocación	El valor de CURRENT SCHEMA cuando se introduce la rutina	Sí
CURRENT SERVER	Heredado de la aplicación de invocación	Heredado de la aplicación de invocación	Sí
CURRENT SQLID	El ID de autorización principal del proceso de aplicación o heredado de la aplicación que lo invoca ⁷	El ID de autorización principal del proceso de solicitud	Sí ⁸
CURRENT TEMPORAL BUSINESS_TIME	Heredado de la aplicación de invocación	Nulo	Sí
CURRENT TEMPORAL SYSTEM_TIME	Heredado de la aplicación de invocación	Nulo	Sí
CURRENT TIME	Nuevo valor para cada instrucción SQL en el ^{pa} quete de funciones definidas por el usuario o procedimientos almacenados ¹	Nuevo valor para cada instrucción SQL en el ^{pa} quete de funciones definidas por el usuario o procedimientos almacenados ¹	No aplicable ⁵
CURRENT TIMESTAMP	Nuevo valor para cada instrucción SQL en el ^{pa} quete de funciones definidas por el usuario o procedimientos almacenados ¹	Nuevo valor para cada instrucción SQL en el ^{pa} quete de funciones definidas por el usuario o procedimientos almacenados ¹	No aplicable ⁵
FECHA Y HORA ACTUAL CON ZONA HORARIA	Nuevo valor para cada instrucción SQL en el ^{pa} quete de funciones definidas por el usuario o procedimientos almacenados ¹	Nuevo valor para cada instrucción SQL en el ^{pa} quete de funciones definidas por el usuario o procedimientos almacenados ¹	No aplicable ⁵
ZONA HORARIA ACTUAL	Heredado de la aplicación de invocación	Heredado de la aplicación de invocación	No aplicable ⁵
ENCRYPTION PASSWORD	Heredado de la aplicación de invocación	Heredado de la aplicación de invocación	Sí
SESSION TIME ZONE	Heredado de la aplicación de invocación	El valor de ZONA HORARIA ACTUAL cuando se introduce la rutina	Sí
SESSION_USER o USER	ID de autorización principal del proceso de solicitud	ID de autorización principal del proceso de solicitud	No aplicable ⁵

Tabla 44. Características de los registros especiales en una función definida por el usuario o un procedimiento almacenado (continuación)

Registro especial	Valor inicial cuando se especifica la opción HEREDAR REGISTROS ESPECIALES	Valor inicial cuando se especifica la opción REGISTROS ESPECIALES POR DEFECTO	¿Puede Routine utilizar la instrucción SET para modificar?
-------------------	---	---	--

Notas:

1. Si la función definida por el usuario o el procedimiento almacenado se invoca dentro del ámbito de un desencadenador, Db2 utiliza la marca de tiempo de la instrucción SQL desencadenante como marca de tiempo para todas las instrucciones SQL del paquete.
2. Db2 permite el paralelismo en un solo nivel de una sentencia SQL anidada. Si establece el valor del registro especial CURRENT DEGREE en ANY y el paralelismo está desactivado, Db2 ignora el valor CURRENT DEGREE.
3. Si la definición de rutina incluye una especificación para COLLID, Db2 establece CURRENT PACKAGESET en el valor de COLLID. Si se especifican tanto CURRENT PACKAGE PATH como COLLID, el valor CURRENT PACKAGE PATH tiene prioridad y COLLID se ignora.
4. Si la definición de la función incluye una especificación para PACKAGE PATH, Db2 establece CURRENT PACKAGE PATH en el valor de PACKAGE PATH.
5. No aplicable porque no existe ninguna declaración SET para el registro especial.
6. Si un programa dentro del ámbito del programa invocador emite una instrucción SET para el registro especial antes de invocar la función definida por el usuario o el procedimiento almacenado, el registro especial hereda el valor de la instrucción SET. De lo contrario, el registro especial contiene el valor establecido por la opción de enlace para el paquete de funciones definidas por el usuario o procedimientos almacenados.
7. Si un programa dentro del ámbito del programa invocador emite una instrucción SET CURRENT SQLID antes de que se invoque la función definida por el usuario o el procedimiento almacenado, el registro especial hereda el valor de la instrucción SET. De lo contrario, CURRENT SQLID contiene el ID de autorización del proceso de solicitud.
8. Si la función definida por el usuario o el paquete de procedimientos almacenados utiliza un valor distinto de RUN para la opción de enlace DYNAMICRULES, se puede ejecutar la instrucción SET CURRENT SQLID. Sin embargo, no afecta al ID de autorización que se utiliza para las sentencias SQL dinámicas en el paquete. El valor DYNAMICRULES determina el ID de autorización que se utiliza para las sentencias SQL dinámicas.

Conceptos relacionados

[Opciones de reglas dinámicas para sentencias de SQL dinámico](#)

La opción de vinculación DYNAMICRULES y el entorno de ejecución determinan las reglas para los atributos de SQL dinámico.

Referencia relacionada

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2 \)](#)

[Registros especiales \(Db2 SQL\)](#)

Acceso a tablas de transición en una función o un procedimiento almacenado definido por el usuario

Si desea hacer referencia al conjunto completo de filas que modifica una sentencia SQL desencadenante, en lugar de filas individuales, utilice una tabla de transición. Puede hacer referencia a una tabla de transición en funciones y procedimientos definidos por el usuario que se invocan desde un desencadenante.

Acerca de esta tarea

Este tema describe cómo acceder a las variables de transición en una función definida por el usuario, pero las mismas técnicas se aplican a un procedimiento almacenado.

Para acceder a las tablas de transición en una función definida por el usuario, utilice localizadores de tablas, que son punteros a las tablas de transición. Usted declara los localizadores de tablas como parámetros de entrada en la instrucción CREATE FUNCTION utilizando la cláusula TABLE LIKE *table-name* AS LOCATOR.

Procedimiento

Para acceder a las tablas de transición en una función definida por el usuario o un procedimiento almacenado:

1. Declare los parámetros de entrada para recibir localizadores de tabla. Debe definir cada parámetro que reciba un localizador de tabla como un entero sin signo de 4 bytes.
2. Declare los localizadores de mesa. Puede declarar localizadores de tablas en ensamblador, C, C++, COBOL, PL/I y en una instrucción compuesta de procedimiento SQL.
3. Declare un cursor para acceder a las filas de cada tabla de transición.
4. Asignar los valores de los parámetros de entrada a los localizadores de la tabla.
5. Acceda a las filas de las tablas de transición utilizando los cursosres que se declaran para las tablas de transición.

Resultados

Los siguientes ejemplos muestran cómo una función definida por el usuario que está escrita en C, C++, COBOL o PL/I accede a una tabla de transición para un disparador. La tabla de transición, NEWEMP, contiene filas modificadas de la tabla de muestra de empleados. El desencadenante se define así:

```
CREATE TRIGGER EMPRAISE
  AFTER UPDATE ON EMP
  REFERENCING NEW TABLE AS NEWEMPS
  FOR EACH STATEMENT MODE DB2SQL
  BEGIN ATOMIC
    VALUES (CHECKEMP(TABLE NEWEMPS));
  END;
```

La definición de la función definida por el usuario tiene este aspecto:

```
CREATE FUNCTION CHECKEMP(TABLE LIKE EMP AS LOCATOR)
  RETURNS INTEGER
  EXTERNAL NAME 'CHECKEMP'
  PARAMETER STYLE SQL
  LANGUAGE language;
```

Ensamblador: *El siguiente ejemplo* muestra cómo un programa ensamblador accede a las filas de la tabla de transición NEWEMPS.

CHECKEMP CSECT	
SAVE (14,12)	ANY SAVE SEQUENCE
LR R12,R15	CODE ADDRESSABILITY
USING CHECKEMP,R12	TELL THE ASSEMBLER
LR R7,R1	SAVE THE PARM POINTER
USING PARMAREA,R7	SET ADDRESSABILITY FOR PARMS
USING SOLDSECT,R8	ESTABLISH ADDRESSIBILITY TO SQLDSECT
L R6,PROGSIZE	GET SPACE FOR USER PROGRAM
GETMAIN R,LV=(6)	GET STORAGE FOR PROGRAM VARIABLES
LR R10,R1	POINT TO THE ACQUIRED STORAGE
LR R2,R10	POINT TO THE FIELD
LR R3,R6	GET ITS LENGTH
SR R4,R4	CLEAR THE INPUT ADDRESS
SR R5,R5	CLEAR THE INPUT LENGTH
MVCL R2,R4	CLEAR OUT THE FIELD
ST R13,FOUR(R10)	CHAIN THE SAVEAREA PTRS
ST R10,EIGHT(R13)	CHAIN SAVEAREA FORWARD
LR R13,R10	POINT TO THE SAVEAREA
USING PROGAREA,R13	SET ADDRESSABILITY

```

ST      R6,GETLENGTH      SAVE THE LENGTH OF THE GETMAIN
:
*****
* Declare table locator host variable TRIGTBL          *
*****
TRIGTBL SQL TYPE IS TABLE LIKE EMP AS LOCATOR
*****
* Declare a cursor to retrieve rows from the transition  *
* table                                                 *
*****
EXEC SQL DECLARE C1 CURSOR FOR                      X
      SELECT LASTNAME FROM TABLE(:TRIGTBL LIKE EMP)      X
      WHERE SALARY > 100000
*****
* Copy table locator for trigger transition table    *
*****
L      R2,TABLOC           GET ADDRESS OF LOCATOR
L      R2,0(0,R2)          GET LOCATOR VALUE
ST     R2,TRIGTBL
EXEC SQL OPEN C1
EXEC SQL FETCH C1 INTO :NAME
:
EXEC SQL CLOSE C1
:PROGAREA DSECT           WORKING STORAGE FOR THE PROGRAM
SAVEAREA DS    18F          THIS ROUTINE'S SAVE AREA
GETLENGTH DS    A           GETMAIN LENGTH FOR THIS AREA
:
NAME     DS    CL24
:
DS    0D
PROGSIZE EQU  *--PROGAREA   DYNAMIC WORKAREA SIZE
PARMAREA DSECT
TABLOC  DS    A           INPUT PARAMETER FOR TABLE LOCATOR
:
END    CHECKEMP

```

C o C++: El siguiente ejemplo muestra cómo un programa C o C++ accede a las filas de la tabla de transición NEWEMPS.

```

int CHECK_EMP(int trig_tbl_id)
{
:
/*
* Declare table locator host variable trig_tbl_id  */
/*
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS TABLE LIKE EMP AS LOCATOR trig_tbl_id;
  char name[25];
EXEC SQL END DECLARE SECTION;
:
/*
* Declare a cursor to retrieve rows from the transition  */
* table                                                 */
/*
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT NAME FROM TABLE(:trig_tbl_id LIKE EMPLOYEE)
  WHERE SALARY > 100000;
/*
* Fetch a row from transition table                  */
/*
EXEC SQL OPEN C1;
EXEC SQL FETCH C1 INTO :name;
:
EXEC SQL CLOSE C1;
:
}

```

COBOL : El siguiente ejemplo muestra cómo un programa COBOL accede a las filas de la tabla de transición NEWEMPS.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. CHECKEMP.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  NAME    PIC X(24).
:
LINKAGE SECTION.

```

```

*****
* Declare table locator host variable TRIG-TBL-ID      *
*****
01 TRIG-TBL-ID SQL TYPE IS TABLE LIKE EMP AS LOCATOR.

.
PROCEDURE DIVISION USING TRIG-TBL-ID.
.

*****
* Declare cursor to retrieve rows from transition table *
*****
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT NAME FROM TABLE(:TRIG-TBL-ID LIKE EMP)
  WHERE SALARY > 100000 END-EXEC.
*****
* Fetch a row from transition table                      *
*****
EXEC SQL OPEN C1 END-EXEC.
EXEC SQL FETCH C1 INTO :NAME END-EXEC.
.
EXEC SQL CLOSE C1 END-EXEC.
.

PROG-END.
GOBACK.

```

PL/I: El siguiente ejemplo muestra cómo un programa PL/I accede a las filas de la tabla de transición NEWEMPS.

```

CHECK_EMP: PROC(TRIG_TBL_ID) RETURNS(BIN FIXED(31))
  OPTIONS(MAIN NOEXECOPS REENTRANT);
/*
* Declare table locator host variable TRIG_TBL_ID  */
DECLARE TRIG_TBL_ID SQL TYPE IS TABLE LIKE EMP AS LOCATOR;
DECLARE NAME CHAR(24);
.

/*
* Declare a cursor to retrieve rows from the          */
/* transition table                                */
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT NAME FROM TABLE(:TRIG_TBL_ID LIKE EMP)
  WHERE SALARY > 100000;
/*
* Retrieve rows from the transition table           */
EXEC SQL OPEN C1;
EXEC SQL FETCH C1 INTO :NAME;
.
EXEC SQL CLOSE C1;
.
END CHECK_EMP;

```

Preparación de una función externa definida por el usuario para la ejecución

Debido a que una función externa definida por el usuario está escrita en un lenguaje de programación, prepararla es similar a preparar cualquier otro programa de aplicación.

Procedimiento

Para preparar una función externa definida por el usuario para su ejecución:

1. Precompilar el programa de funciones definido por el usuario y vincular el DBRM en un paquete. Solo debe hacerlo si su función definida por el usuario contiene instrucciones SQL. No es necesario vincular un plan para la función definida por el usuario.
2. Compilar el programa de funciones definido por el usuario y editar el enlace con Language Environment y RRSAF.

Debe compilar el programa con un compilador que admite el entorno de lenguaje y vincular y editar los componentes adecuados del entorno de lenguaje con la función definida por el usuario. También debe vincular y editar la función definida por el usuario con RRSAF.

Las muestras JCL de preparación de programas DSNHASM, DSNHC, DSNHCPP, DSNHICOB y DSNHPLI le muestran cómo precompilar, compilar y editar enlaces de ensamblador, C, C++, COBOL y PL/I

programas de Db2 . Para programas orientados a objetos en C++, consulte el ejemplo JCL DSNHCPP2 para obtener consejos de preparación del programa.

3. Para una función definida por el usuario que contenga sentencias SQL, conceda al definidor de la función la autoridad EXECUTE en el paquete de la función definida por el usuario.

Terminación anómala de una función externa definida por el usuario

Si una función externa definida por el usuario termina de forma anómala, el programa recibe SQLCODE -430 para invocar la sentencia.

Db2 también realiza las siguientes acciones:

- Coloca la unidad de trabajo que contiene la sentencia de invocación en un estado de reversión obligatoria.
- Detiene la función definida por el usuario y las llamadas posteriores fallan en cualquiera de las siguientes situaciones:
 - El número de terminaciones anormales es igual al valor DETENER DESPUÉS DE *n* FALLOS para la función definida por el usuario.
 - Si no se especifica la opción DETENER DESPUÉS DE *n* FALLOS, el número de terminaciones anormales es igual al valor predeterminado de RECUENTO MÁX. DE FALLOS para el subsistema.

Debe incluir código en su programa para comprobar si hay un fallo de función definida por el usuario y para revertir la unidad de trabajo que contiene la invocación de la función definida por el usuario.

Guardar información entre invocaciones de una función definida por el usuario mediante el uso de un bloc de notas

Si crea un bloc de notas para una función reentrante definida por el usuario, Db2 puede utilizarlo para conservar la información entre las invocaciones de la función.

Acerca de esta tarea

Puede utilizar un bloc de notas para guardar información entre invocaciones de una función definida por el usuario. Para indicar que se debe asignar un bloc de notas cuando se ejecute la función definida por el usuario, el definidor de la función especifica el parámetro SCRATCHPAD en la instrucción CREATE FUNCTION.

El bloc de notas consta de un campo de longitud de 4 bytes, seguido del área del bloc de notas. El definidor puede especificar la longitud del área de la libreta de notas en la instrucción CREATE FUNCTION. La longitud especificada no incluye el campo de longitud. El tamaño por omisión es 100 bytes. Db2 inicializa el bloc de notas para cada función a ceros binarios al comienzo de la ejecución para cada subconsulta de una instrucción SQL y no examina ni cambia el contenido a partir de entonces. Db2 , en cada invocación de la función definida por el usuario, pasa el bloc de notas a la función definida por el usuario. Por lo tanto, puede utilizar el bloc de notas para conservar la información entre las invocaciones de una función reentrante definida por el usuario.

El siguiente ejemplo muestra cómo introducir información en un bloc de notas para una función definida por el usuario definida de esta manera:

```
CREATE FUNCTION COUNTER()
RETURNS INT
SCRATCHPAD
FENCED
NOT DETERMINISTIC
NO SQL
NO EXTERNAL ACTION
LANGUAGE C
PARAMETER STYLE SQL
EXTERNAL NAME 'UDFCTR';
```

La longitud del bloc de notas no está especificada, por lo que el bloc de notas tiene la longitud predeterminada de 100 bytes, más 4 bytes para el campo de longitud. La función definida por el usuario incrementa un valor entero y lo almacena en el bloc de notas en cada ejecución.

```
#pragma linkage(ctr,fetchable)
#include <stdlib.h>
#include <stdio.h>
/* Structure scr defines the passed scratchpad for function ctr */
struct scr {
    long len;
    long countr;
    char not_used[96];
};
/*********************************************
/* Function ctr: Increments a counter and reports the value */
/* from the scratchpad. */
*/
/*
/* Input: None
/* Output: INTEGER out      the value from the scratchpad */
/*********************************************
void ctr(
    long *out,                      /* Output answer (counter) */
    short *outnull,                 /* Output null indicator */
    char *sqlstate,                /* SQLSTATE */
    char *funcname,                /* Function name */
    char *specname,                /* Specific function name */
    char *mesgtext,                 /* Message text insert */
    struct scr *scratchptr)        /* Scratchpad */
{
    *out = ++scratchptr->countr;   /* Increment counter and */
    /* copy to output variable */
    *outnull = 0;                  /* Set output null indicator*/
    return;
}
/* end of user-defined function ctr */
```

Ejemplo de creación y uso de una función escalar definida por el usuario

Puede crear una función escalar definida por el usuario que reciba la entrada de una tabla y ponga la salida en una tabla.

Supongamos que su organización necesita una función escalar definida por el usuario que calcule la bonificación que recibe cada empleado. Todos los datos de los empleados, incluidos los salarios, comisiones y bonificaciones, se guardan en la tabla de empleados, EMP. Los campos de entrada para la función de cálculo de bonificaciones son los valores de las columnas SALARIO y COMISIÓN. El resultado de la función va a la columna BONUS. Dado que esta función obtiene su entrada de una tabla de tipo "Db2" y pone la salida en una tabla de tipo "Db2", una forma conveniente de manipular los datos es a través de una función definida por el usuario.

El definidor y el invocador de la función definida por el usuario determinan que esta nueva función definida por el usuario debe tener estas características:

- El nombre de la función definida por el usuario es CALC_BONUS.
- Los dos campos de entrada son de tipo DECIMAL(9,2).
- El campo de salida es de tipo DECIMAL(9,2).
- El programa para la función definida por el usuario está escrito en COBOL y tiene un nombre de módulo de carga CBONUS.

Dado que no existe ninguna función integrada o definida por el usuario sobre la que construir una función definida por el usuario de origen, el implementador de la función debe codificar una función definida por el usuario externa. El implementador realiza los siguientes pasos:

- Escribe la función definida por el usuario, que es un programa COBOL
- Precompila, compila y vincula el programa
- Enlaza un paquete si la función definida por el usuario contiene sentencias SQL
- Prueba el programa a fondo
- Otorga autoridad de ejecución sobre el paquete de funciones definido por el usuario al definidor

El definidor de funciones definido por el usuario ejecuta esta instrucción CREATE FUNCTION para registrar CALC_BONUS en Db2:

```
CREATE FUNCTION CALC_BONUS(DECIMAL(9,2),DECIMAL(9,2))
RETURNS DECIMAL(9,2)
EXTERNAL NAME 'CBONUS'
PARAMETER STYLE SQL
LANGUAGE COBOL;
```

El definidor concede entonces autoridad de ejecución en CALC_BONUS a todos los invocadores.

Los invocadores de funciones definidos por el usuario escriben y preparan programas de aplicación que invocan CALC_BONUS. Un invocador podría escribir una declaración como esta, que utiliza la función definida por el usuario para actualizar el campo BONUS en la tabla de empleados:

```
UPDATE EMP
SET BONUS = CALC_BONUS(SALARY,COMM);
```

Un invocador puede ejecutar esta declaración de forma estática o dinámica.

Ejemplos de funciones definidas por el usuario que se suministran con Db2

Para ayudarle a definir, implementar e invocar las funciones definidas por usuario, Db2 proporciona varios ejemplos de funciones definidas por usuario. Todos los códigos de funciones definidas por el usuario de ejemplo están en el conjunto de datos DSN1210.SDSNSAMP.

La siguiente tabla resume las características de las funciones de ejemplo definidas por el usuario.

Tabla 45. Muestras de funciones definidas por el usuario enviadas con Db2

Nombre de función definida por el usuario	Idioma	Miembro que contiene código fuente	Finalidad
FECHAANTERIOR1	C	DSN8DUAD	Convierte la fecha actual a un formato especificado por el usuario
FECHA2 ANTERIOR	C	DSN8DUCD	Convierte una fecha de un formato a otro
ALTTIME3	C	DSN8DUAT	Convierte la hora actual a un formato especificado por el usuario
ALTTIME4	C	DSN8DUCT	Convierte una hora de un formato a otro
DAYNAME	C++	DSN8EUDN	Devuelve el día de la semana para una fecha especificada por el usuario
HDFS_READ	C++	DSN8HDFS	Lee datos de un archivo separado por delimitadores en el Sistema de archivos distribuidos de Linux (Hadoop, HDFS)
JAQL_SUBMIT	C++	DSN8JAQL	Invoca un IBMInfoSphereBigInsights Consulta Jaql
MONTHNAME	C++	DSN8EUMN	Devuelve el mes para una fecha especificada por el usuario
Moneda	C	DSN8DUCY	Da formato a un número de punto flotante como valor de moneda
TABLE_NAME	C	DSN8DUTI	Devuelve el nombre de tabla sin cualificar para una tabla, vista o alias
TABLE_QUALIF	C	DSN8DUTI	Devuelve el calificador de una tabla, vista o alias
TABLE_LOCATION	C	DSN8DUTI	Devuelve la ubicación de una tabla, vista o alias

Tabla 45. Muestras de funciones definidas por el usuario enviadas con Db2 (continuación)

Nombre de función definida por el usuario	Idioma	Miembro que contiene código fuente	Finalidad
Meteorología	C	DSN8DUWF	Devuelve una tabla de información meteorológica de un conjunto de datos EBCDIC

Notas:

1. Esta versión de ALTDATE tiene un parámetro de entrada, de tipo VARCHAR(13).
2. Esta versión de ALTDATE tiene tres parámetros de entrada, de tipo VARCHAR(17), VARCHAR(13) y VARCHAR(13).
3. Esta versión de ALTTIME tiene un parámetro de entrada, de tipo VARCHAR(14).
4. Esta versión de ALTTIME tiene tres parámetros de entrada, de tipo VARCHAR(11), VARCHAR(14) y VARCHAR(14).

DSN8DUWC, para miembros, contiene un programa cliente que muestra cómo invocar la función de tabla definida por el usuario WEATHER.

El miembro DSNEJBI le muestra cómo definir y preparar el IBMInfoSphereBigInsights muestra cómo definir y preparar las funciones de ejemplo definidas por el usuario.

DSNTEJ2U, miembro de la comunidad, le muestra cómo definir y preparar las otras funciones de ejemplo definidas por el usuario y el programa cliente.

Conceptos relacionados

[Trabajo DSNEJBI \(Db2 Installation and Migration\)](#)

[Trabajo DSNTEJ2U \(Db2 Installation and Migration\)](#)

[Funciones definidas por el usuario de ejemplo \(Db2 SQL\)](#)

Creación de procedimientos almacenados

Un archivo de extensión. *procedimiento almacenado*, es un código ejecutable que puede ser llamado por otros programas. El proceso de creación depende del tipo de procedimiento.

Antes de empezar

Debe completar algunas tareas de configuración para el entorno Db2 para poder utilizar cualquiera de los siguientes tipos de procedimientos:

- Procedimientos almacenados externos
- Procedimientos SQL nativos que cumplen cualquiera de las condiciones siguientes:
 - Llama al menos un procedimiento almacenado externo, un procedimiento de SQL externo o una función definida por el usuario.
 - Definido con ALLOW DEBUG MODE o DISALLOW DEBUG MODE.
- Procedimientos de SQL externo (en desuso)
- Procedimientos almacenados suministrados por Db2

Para obtener instrucciones, consulte [Paso de instalación 21 : Configurar Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario \(Db2 Instalación y migración\)](#) o [Paso de migración 23 : Configurar Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario \(opcional\) \(Instalación y migración Db2\)](#).

Procedimiento

Siga el proceso para el tipo de procedimiento almacenado que desea crear y emita una sentencia CREATE PROCEDURE para registrar el procedimiento almacenado con un servidor de bases de datos.

Puede crear los siguientes tipos de procedimientos almacenados:

Procedimientos de SQL nativos

El cuerpo del procedimiento se graba exclusivamente en sentencias de SQL, incluidas las sentencias de lenguaje de procedimientos de SQL (SQL PL). El cuerpo del procedimiento está contenido y se especifica en la definición de procedimiento junto con varios atributos del procedimiento. Se genera un paquete para un procedimiento de SQL nativo. Contiene el cuerpo del procedimiento, incluidas las sentencias de control. A veces puede incluir también sentencias generadas por Db2. Cada vez que se invoca el procedimiento, el paquete se ejecuta una o varias veces.

Todos los procedimientos SQL que se crean con una sentencia CREATE PROCEDURE que no especifica las opciones FENCED o EXTERNAL son procedimientos SQL nativos. Se da soporte a más prestaciones para los procedimientos de SQL nativo, normalmente funcionan mejor que los procedimientos SQL externos y no se genera ningún programa C asociado para ellos.

Para obtener más información, consulte [“Creación de procedimientos SQL nativos” en la página 244](#).

Procedimientos almacenados externos

El cuerpo del procedimiento es un programa externo que se escribe en un lenguaje de programación como C, C++, COBOL o Java y puede contener sentencias de SQL. El código fuente de un procedimiento almacenado externo está separado de la definición de procedimiento y está enlazado en un paquete. El nombre del ejecutable externo se especifica como parte de la definición de procedimiento junto con varios atributos del procedimiento. Todos los programas deben estar diseñados para ejecutarse utilizando Language Environment. Los procedimientos almacenados COBOL y C++ pueden contener extensiones orientadas a objetos. Cada vez que se invoca el procedimiento almacenado, la lógica del procedimiento controla si el paquete se ejecuta y cuántas veces.

Para obtener más información, consulte [“Creación de procedimientos almacenados externos” en la página 270](#).

Procedimientos de SQL externo (en desuso)

El cuerpo del procedimiento se graba exclusivamente en sentencias de SQL, incluidas las sentencias de lenguaje de procedimientos de SQL (SQL PL). El cuerpo del procedimiento se especifica en la definición del procedimiento junto con varios atributos del procedimiento. Se genera un programa C y un paquete asociado para un procedimiento de SQL externo. Contiene el cuerpo del procedimiento, incluidas las sentencias de control. A veces puede incluir también sentencias generadas por Db2. Cada vez que se invoca el procedimiento, el paquete se ejecuta una o varias veces.

Los procedimientos de SQL nativo son más compatibles y más fáciles de mantener y normalmente funcionan mejor que los procedimientos SQL externos, que están en desuso.

Para obtener más información, consulte [“Creación de procedimientos SQL externos \(en desuso\)” en la página 306](#).



Conceptos relacionados

Procedimientos almacenados

Un *procedimiento almacenado* es un programa compilado que puede ejecutar sentencias de SQL y que se almacena en un servidor Db2 local o remoto. Puede invocar un procedimiento almacenado desde un programa de aplicación o desde el Db2 command line processor. Una única llamada a un procedimiento almacenado desde una aplicación cliente puede acceder a la base de datos del servidor en varias ocasiones.

Procedimientos almacenados externos

Un *procedimiento almacenado externo* es un procedimiento que se escribe en un lenguaje de host y que puede contener sentencias SQL. El código fuente de los procedimientos externos es distinto de la definición.

Procedimientos de SQL

Un procedimiento de SQL es un procedimiento almacenado que solo contiene sentencias SQL.

Tareas relacionadas

[Ocultación del código fuente de procedimientos SQL, funciones SQL y desencadenantes \(Db2 Administration Guide\)](#)

Referencia relacionada

[Instrucción CREATE PROCEDURE \(resumen\) \(Db2 SQL\)](#)

[Db2 for z/OS Cambio](#)

Información relacionada

[Db2 para procedimientos almacenados de SQL Server \(z/OS Stored Procedures: Through the CALL and Beyond \(IBM Redbooks \)](#)

Procedimientos almacenados

Un *procedimiento almacenado* es un programa compilado que puede ejecutar sentencias de SQL y que se almacena en un servidor Db2 local o remoto. Puede invocar un procedimiento almacenado desde un programa de aplicación o desde el Db2 command line processor. Una única llamada a un procedimiento almacenado desde una aplicación cliente puede acceder a la base de datos del servidor en varias ocasiones.

Un procedimiento almacenado típico contiene dos o más sentencias de SQL y proceso de manipulación o lógico en un lenguaje principal o sentencias de procedimiento de SQL. Puede llamar a procedimientos almacenados desde otras aplicaciones o desde la línea de mandatos. Db2 proporciona varios procedimientos almacenados, pero también puede crear procedimientos almacenados propios.

Un procedimiento almacenado proporciona una parte común de código que sólo se escribe una única vez y que se mantiene en una única instancia a la que puede llamarse desde varias aplicaciones distintas. Los lenguajes de host pueden llamar fácilmente a procedimientos que existen en el sistema local, y SQL puede llamar a procedimientos almacenados que existen en sistemas remotos. De hecho, una de las principales ventajas de los procedimientos en SQL es que pueden utilizarse para mejorar las características de rendimiento de las aplicaciones distribuidas. Con los procedimientos almacenados, puede eludir la transferencia mediante la red de grandes cantidades de datos obtenidos como parte de los resultados intermedios de una secuencia de consultas larga.

En el diagrama siguiente se muestra el proceso para una aplicación que no utiliza procedimientos almacenados. La aplicación cliente incorpora sentencias de SQL y se comunica con el servidor de forma separada para cada sentencia. Este diseño de aplicación incrementa el tráfico de la red y los costes de procesador.

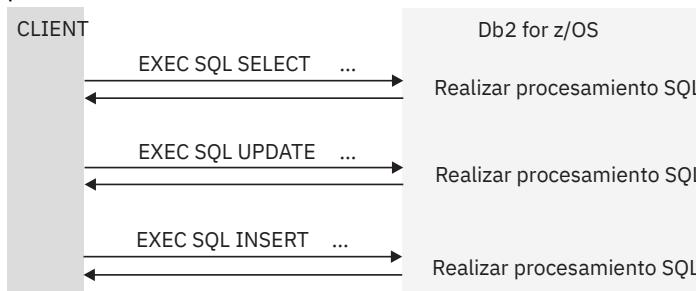


Figura 9. Proceso sin procedimientos almacenados

En el diagrama siguiente se muestra el proceso para una aplicación que utiliza procedimientos almacenados. Puesto que un procedimiento almacenado se utiliza en el servidor, puede ejecutarse una serie de sentencias de SQL con una única operación de envío y recepción, reduciendo el tráfico de la red y el coste del proceso de estas sentencias.

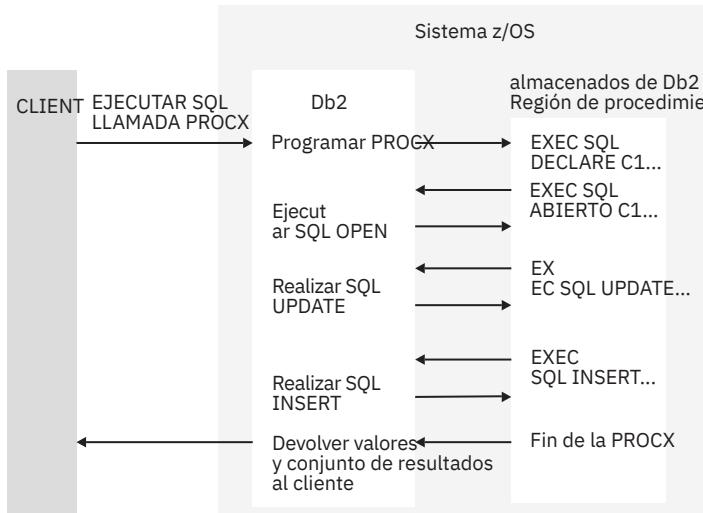


Figura 10. Proceso con procedimientos almacenados

Los procedimientos almacenados son útiles para las aplicaciones cliente/servidor que realizan, como mínimo, una de las acciones siguientes:

- Ejecutar varias sentencias de SQL remotas. Las sentencias de SQL remotas pueden crear muchas operaciones de envío y recepción, lo cual incrementa los costes de procesador. Los procedimientos almacenados pueden encapsular muchas de las sentencias de SQL de la aplicación en un único mensaje hacia el servidor Db2, con lo cual el tráfico en la red se reduce a una única operación de envío y recepción para una serie de sentencias de SQL. Los bloqueos sobre las tablas de Db2 no se mantienen entre transmisiones de red, lo cual reduce la contienda de recursos en el servidor.
- Acceder a tablas desde un entorno de SQL dinámico en el que no se desean utilizar privilegios de tabla para la aplicación que está en ejecución. Los procedimientos almacenados permiten autorización de SQL estático desde un entorno dinámico.
- Acceder a variables de lenguaje principal cuya seguridad e integridad se desea garantizar. Los procedimientos almacenados eliminan aplicaciones de SQL de la estación de trabajo, lo que impide que los usuarios de la estación de trabajo puedan manipular el contenido de sentencias de SQL y de variables de lenguaje principal sensibles.
- Crear un conjunto de resultados de filas para su devolución a la aplicación cliente.

Los procedimientos almacenados que se han escrito en SQL estático incorporado proporcionan las siguientes ventajas adicionales:

- Mejor rendimiento, pues el SQL estático se prepara durante la precompilación y no genera sobrecarga en tiempo de ejecución para la generación del plan de acceso (paquete).
- La encapsulación permite a los programadores escribir aplicaciones que accedan a los datos sin necesidad de conocer los detalles de los objetos de base de datos.
- Seguridad mejorada, pues los privilegios de acceso se encapsulan dentro de los paquetes que se asocian a los procedimientos almacenados. Puede otorgar acceso para ejecutar un procedimiento almacenado que selecciona datos de tablas, sin necesidad de otorgar al usuario el privilegio SELECT.

Puede crear los siguientes tipos de procedimientos almacenados:

Procedimientos de SQL nativos

El cuerpo del procedimiento se graba exclusivamente en sentencias de SQL, incluidas las sentencias de lenguaje de procedimientos de SQL (SQL PL). El cuerpo del procedimiento está contenido y se especifica en la definición de procedimiento junto con varios atributos del procedimiento. Se genera un paquete para un procedimiento de SQL nativo. Contiene el cuerpo del procedimiento, incluidas las sentencias de control. A veces puede incluir también sentencias generadas por Db2. Cada vez que se invoca el procedimiento, el paquete se ejecuta una o varias veces.

Todos los procedimientos SQL que se crean con una sentencia CREATE PROCEDURE que no especifica las opciones FENCED o EXTERNAL son procedimientos SQL nativos. Se da soporte a más prestaciones para los procedimientos de SQL nativo, normalmente funcionan mejor que los procedimientos SQL externos y no se genera ningún programa C asociado para ellos.

Para obtener más información, consulte [“Creación de procedimientos SQL nativos” en la página 244](#).

Procedimientos almacenados externos

El cuerpo del procedimiento es un programa externo que se escribe en un lenguaje de programación como C, C++, COBOL o Java y puede contener sentencias de SQL. El código fuente de un procedimiento almacenado externo está separado de la definición de procedimiento y está enlazado en un paquete. El nombre del ejecutable externo se especifica como parte de la definición de procedimiento junto con varios atributos del procedimiento. Todos los programas deben estar diseñados para ejecutarse utilizando Language Environment. Los procedimientos almacenados COBOL y C++ pueden contener extensiones orientadas a objetos. Cada vez que se invoca el procedimiento almacenado, la lógica del procedimiento controla si el paquete se ejecuta y cuántas veces.

Para obtener más información, consulte [“Creación de procedimientos almacenados externos” en la página 270](#).

Procedimientos de SQL externo (en desuso)

El cuerpo del procedimiento se graba exclusivamente en sentencias de SQL, incluidas las sentencias de lenguaje de procedimientos de SQL (SQL PL). El cuerpo del procedimiento se especifica en la definición del procedimiento junto con varios atributos del procedimiento. Se genera un programa C y un paquete asociado para un procedimiento de SQL externo. Contiene el cuerpo del procedimiento, incluidas las sentencias de control. A veces puede incluir también sentencias generadas por Db2. Cada vez que se invoca el procedimiento, el paquete se ejecuta una o varias veces.

Los procedimientos de SQL nativo son más compatibles y más fáciles de mantener y normalmente funcionan mejor que los procedimientos SQL externos, que están en desuso.

Para obtener más información, consulte [“Creación de procedimientos SQL externos \(en desuso\)” en la página 306](#).

Db2 también proporciona un conjunto de procedimientos almacenados que puede llamar en sus programas de aplicación para llevar a cabo diversas funciones de programa de utilidad, programación de aplicaciones y gestión del rendimiento. Estos procedimientos se denominan *Procedimientos almacenados proporcionados*. Generalmente, estos procedimientos se crean durante la instalación o la migración.

Conceptos relacionados

[Procedimientos almacenados de la API de SQL común \(Db2 Administration Guide\)](#)

Tareas relacionadas

[Implementación de procedimientos almacenados de Db2 \(procedimientos almacenados proporcionados por Db2\)](#)

Referencia relacionada

[Procedimientos que se suministran con Db2 \(Db2 SQL\)](#)

Parámetros de procedimientos almacenados

Puede pasar información entre un procedimiento almacenado y el programa de aplicación de llamada utilizando parámetros. Las aplicaciones pasan los parámetros necesarios en la sentencia CALL de SQL. Opcionalmente, la aplicación también puede incluir una variable de indicador con cada parámetro para permitir valores nulos o pasar valores gramdes de parámetro de salida.

Usted define los parámetros del procedimiento almacenado como parte de la definición del procedimiento almacenado en la instrucción CREATE PROCEDURE. Los parámetros del procedimiento almacenado pueden ser de uno de los siguientes tipos:

ENTRADA

Parámetros de solo entrada, que proporcionan valores al procedimiento almacenado.

OUT

Parámetros de solo salida, que devuelven valores del procedimiento almacenado al programa de llamada.

INOUT

Parámetros de entrada y salida, que proporcionan valores al procedimiento almacenado y devuelven valores desde él.

Si un procedimiento almacenado no establece uno o más de los parámetros OUT o INOUT, Db2 no devuelve un error. En su lugar, Db2 devuelve los parámetros de salida al programa de llamada, con los valores que se establecieron al entrar en el procedimiento almacenado.

Dentro de un cuerpo de procedimiento, las siguientes reglas se aplican a los parámetros IN, OUT e INOUT:

- Puede utilizar un parámetro que defina como IN en el lado izquierdo o derecho de una sentencia de asignación. Sin embargo, si asigna un valor a un parámetro IN, no puede devolver el nuevo valor al llamante. El parámetro IN tiene el mismo valor antes y después de que se llame al procedimiento SQL.
- Puede utilizar un parámetro que defina como OUT en el lado izquierdo o derecho de una sentencia de asignación. El último valor que asignas al parámetro es el valor que se devuelve al llamador. El valor inicial de un parámetro OUT es NULL.
- Puede utilizar un parámetro que defina como INOUT en el lado izquierdo o derecho de una instrucción de asignación. La persona que llama determina el primer valor del parámetro INOUT, y el último valor que se le asigna al parámetro es el valor que se devuelve a la persona que llama.

Restricciones:

- No puede pasar variables de referencia de archivo como parámetros de procedimiento almacenado.
- No se pueden pasar parámetros con el tipo XML a procedimientos almacenados. Puede especificar tablas o vistas que contengan columnas XML como parámetros de localización de tablas. Sin embargo, no puede hacer referencia a las columnas XML en el cuerpo del procedimiento almacenado.

Tareas relacionadas

[Invocación de un procedimiento almacenado desde una aplicación](#)

Para ejecutar un procedimiento almacenado, puede llamarlo desde un programa cliente o invocarlo desde el Db2 command line processor.

[Pase de parámetros de salida de gran tamaño a procedimientos almacenados mediante variables de indicador](#)

Si algún parámetro de salida ocupa un gran volumen de almacenamiento, el pase del área de almacenamiento completa a un procedimiento almacenado puede disminuir el rendimiento.

Referencia relacionada

[CALL declaración \(Db2 SQL\)](#)

[Instrucción CREATE PROCEDURE \(resumen\) \(Db2 SQL\)](#)

Ejemplo de un procedimiento almacenado simple

Cuando una aplicación que se ejecuta en una estación de trabajo llama a un procedimiento almacenado de un servidor Db2, el procedimiento almacenado actualiza una tabla en función de la información que recibe de la aplicación.

Supongamos que una aplicación se ejecuta en una estación de trabajo cliente y llama a un procedimiento almacenado A en el servidor Db2 en la ubicación LOCA. El procedimiento almacenado A realiza las siguientes operaciones:

1. Recibe un conjunto de parámetros que contienen los datos de una fila de la tabla de actividades proyectadas del empleado (DSN8C10.EMPPROJECT). Estos parámetros son parámetros de entrada en la instrucción SQL CALL:
 - Número de empleado (EMP)
 - PRJ: número de proyecto
 - ACT: ID de actividad

- EMT: porcentaje de tiempo del empleado requerido
 - EMS: fecha de inicio de la actividad
 - EME: fecha en la que debe finalizar la actividad
2. Declara un cursor, C1, con la opción WITH RETURN, que se utiliza para devolver un conjunto de resultados que contiene todas las filas de EMPPROJECT a la aplicación de la estación de trabajo que llamó al procedimiento almacenado.
 3. Consulta la tabla EMPPROJECT para determinar si existe una fila en la que las columnas PROJNO, ACTNO, EMSTDATE y EMPNO coincidan con los valores de los parámetros PRJ, ACT, EMS y EMP. (La tabla tiene un índice único en esas columnas. Hay como máximo una fila con esos valores)
 4. Si la fila existe, ejecuta una instrucción SQL UPDATE para asignar los valores de los parámetros EMT y EME a las columnas EMPTIME y EMENDATE.¹
 5. Si la fila no existe (SQLCODE +100), ejecuta una instrucción SQL INSERT para insertar una nueva fila con todos los valores de la lista de parámetros.¹
 6. Abre el cursor C1. Esto hace que el conjunto de resultados se devuelva al llamador cuando finalice el procedimiento almacenado.
 7. Devuelve dos parámetros, que contienen estos valores:
 - Un código para identificar el tipo de instrucción SQL ejecutada por última vez: UPDATE o INSERT.
 - El SQLCODE de esa declaración.

Nota:

1. De forma alternativa, los pasos 4 y 5 pueden realizarse con una sola instrucción MERGE.

La siguiente figura ilustra los pasos que intervienen en la ejecución de este procedimiento almacenado.

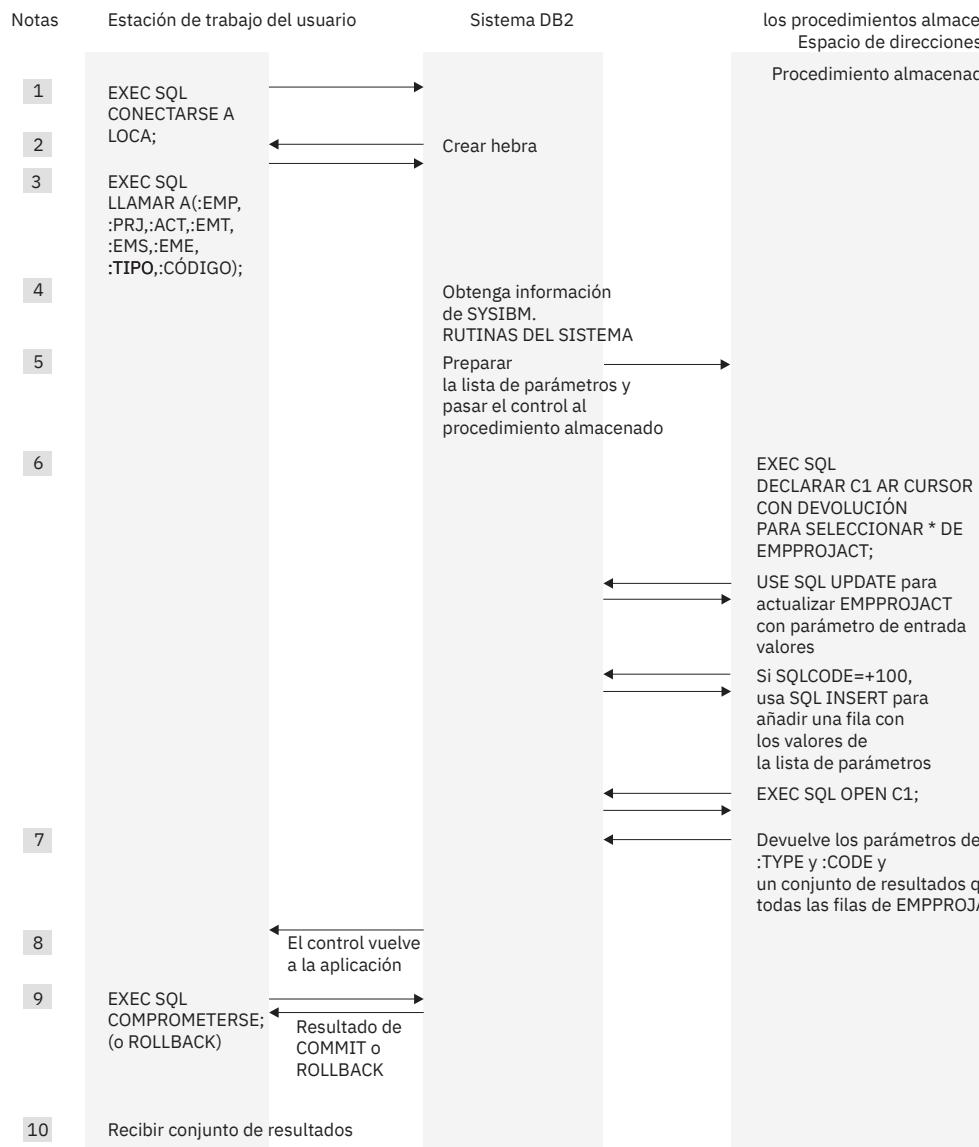


Figura 11. Descripción general del procedimiento almacenado

Notas:

1. La aplicación de estación de trabajo utiliza la sentencia CONNECT de SQL para crear una conversación con Db2.
2. Db2 crea una hebra de Db2 para procesar solicitudes de SQL.
3. La sentencia CALL de SQL indica al servidor Db2 que la aplicación va a ejecutar un procedimiento almacenado. La aplicación de llamada proporciona los parámetros necesarios.
4. El plan para la aplicación del cliente contiene información de la tabla del catálogo SYSIBM.SYSROUTINES sobre el procedimiento almacenado A.
5. Db2 pasa la información sobre la solicitud al espacio de direcciones de procedimientos almacenados, y el procedimiento almacenado comienza su ejecución.
6. El procedimiento almacenado ejecuta sentencias de SQL.

Db2 verifica que el propietario del paquete o plan que contiene la instrucción SQL CALL tiene autoridad EXECUTE para el paquete asociado con el procedimiento almacenado Db2 .

Una de las sentencias de SQL abre un cursor que se ha declarado como WITH RETURN. Esto hace que se devuelva un conjunto de resultados a la aplicación de la estación de trabajo cuando finaliza el procedimiento.

Cualquier SQLCODE que se emita dentro de un procedimiento almacenado **externo no se devuelve** a la aplicación de la estación de trabajo en el SQLCA (como resultado de la instrucción CALL).

7. Si no se encuentra ningún error, el procedimiento almacenado asigna valores a los parámetros de salida y sale.

El control vuelve al espacio de direcciones de procedimientos almacenados de Db2 , y de ahí al sistema de Db2 . Si la definición del procedimiento almacenado contiene COMMIT ON RETURN NO, Db2 no confirma ni revierte ningún cambio del SQL en el procedimiento almacenado hasta que el programa de llamada ejecute una instrucción COMMIT o ROLLBACK explícita. Si la definición del procedimiento almacenado contiene COMMIT ON RETURN YES, y el procedimiento almacenado se ejecuta correctamente, Db2 confirma todos los cambios. La sentencia COMMIT cierra el cursor a menos que se declare con la opción WITH HOLD.

8. Se devuelve el control a la aplicación que realiza la llamada, la cual recibe los parámetros de salida y el conjunto de resultados. Db2 y a continuación:

- Cierra todos los cursos que el procedimiento almacenado abrió, excepto aquellos que el procedimiento almacenado abrió para devolver conjuntos de resultados.
- Descarta todas las sentencias SQL que el procedimiento almacenado preparó.
- Recupera el almacenamiento de trabajo que utilizó el procedimiento almacenado.

La aplicación puede llamar a más procedimientos almacenados o puede ejecutar más sentencias SQL. Db2 recibe y procesa la solicitud COMMIT o ROLLBACK. La operación COMMIT o ROLLBACK cubre todas las operaciones SQL, ya sean ejecutadas por la aplicación o por procedimientos almacenados, para esa unidad de trabajo.

Si la solicitud implica IMS o CICS, se produce un procesamiento similar basado en el IMS o punto de sincronización en lugar de en una instrucción SQL COMMIT o ROLLBACK CICS punto de sincronización en lugar de en una instrucción SQL COMMIT o ROLLBACK.

9. Db2 devuelve un mensaje de respuesta a la aplicación que describe el resultado de la operación COMMIT o ROLLBACK.

10. La aplicación de la estación de trabajo ejecuta los siguientes pasos para recuperar el contenido de la tabla EMPPROJECT, que el procedimiento almacenado ha devuelto en un conjunto de resultados:

- a. Declara un localizador de conjunto de resultados para el conjunto de resultados devuelto.
- b. Ejecuta la sentencia ASSOCIATE LOCATORS para asociar el localizador del conjunto de resultados con el conjunto de resultados.
- c. Ejecuta la instrucción ALLOCATE CURSOR para asociar un cursor con el conjunto de resultados.
- d. Ejecuta la sentencia FETCH con el cursor asignado varias veces para recuperar las filas en el conjunto de resultados.
- e. Ejecuta la instrucción CLOSE para cerrar el cursor.

Procedimientos de SQL

Un procedimiento de SQL es un procedimiento almacenado que solo contiene sentencias SQL.

El código fuente de estos procedimientos (las sentencias SQL) se especifica en la sentencia CREATE PROCEDURE. La parte de la instrucción CREATE PROCEDURE que contiene instrucciones SQL se denomina *cuerpo del procedimiento*.

Tipos de procedimientos SQL

Db2 for z/OS admite los siguientes tipos de procedimientos SQL:

Procedimientos de SQL nativos

El cuerpo del procedimiento se graba exclusivamente en sentencias de SQL, incluidas las sentencias de lenguaje de procedimientos de SQL (SQL PL). El cuerpo del procedimiento está contenido y se especifica en la definición de procedimiento junto con varios atributos del procedimiento. Se genera un paquete para un procedimiento de SQL nativo. Contiene el cuerpo del procedimiento, incluidas las

sentencias de control. A veces puede incluir también sentencias generadas por Db2. Cada vez que se invoca el procedimiento, el paquete se ejecuta una o varias veces.

Todos los procedimientos SQL que se crean con una sentencia CREATE PROCEDURE que no especifica las opciones FENCED o EXTERNAL son procedimientos SQL nativos. Se da soporte a más prestaciones para los procedimientos de SQL nativo, normalmente funcionan mejor que los procedimientos SQL externos y no se genera ningún programa C asociado para ellos.

Para obtener más información, consulte “[Creación de procedimientos SQL nativos](#)” en la página 244.

Procedimientos de SQL externo (en desuso)

El cuerpo del procedimiento se graba exclusivamente en sentencias de SQL, incluidas las sentencias de lenguaje de procedimientos de SQL (SQL PL). El cuerpo del procedimiento se especifica en la definición del procedimiento junto con varios atributos del procedimiento. Se genera un programa C y un paquete asociado para un procedimiento de SQL externo. Contiene el cuerpo del procedimiento, incluidas las sentencias de control. A veces puede incluir también sentencias generadas por Db2. Cada vez que se invoca el procedimiento, el paquete se ejecuta una o varias veces.

Los procedimientos de SQL nativo son más compatibles y más fáciles de mantener y normalmente funcionan mejor que los procedimientos SQL externos, que están en desuso.

Para obtener más información, consulte “[Creación de procedimientos SQL externos \(en desuso\)](#)” en la página 306.

Procedimientos de SQL nativos

Un *procedimiento de SQL nativo* es un procedimiento cuyo cuerpo se ha escrito en SQL en su totalidad. El cuerpo está escrito en el lenguaje de procedimientos de SQL (SQL PL). Un procedimiento de SQL nativo se crea emitiendo una única sentencia SQL, CREATE PROCEDURE. Los procedimientos de SQL nativos no necesitan ninguna otra preparación de programa, como la precompilación, la compilación o la edición de enlaces del código fuente. Los procedimientos de SQL nativos se ejecutan como sentencias de SQL que se han vinculado en un paquete de Db2. Los procedimientos de SQL nativo no tienen un programa de aplicación externo asociado. Los procedimientos de SQL nativo son más compatibles y más fáciles de mantener y normalmente funcionan mejor que los procedimientos SQL externos, que están en desuso.

Los procedimientos nativos de SQL tienen las siguientes ventajas:

- Puede crearlos en un solo paso.
- No funcionan en un entorno WLM.
- Podrían ser elegibles para una redirección de " zIIP " si se invocan de forma remota a través de un cliente DRDA.
- Suelen funcionar mejor que los procedimientos SQL externos.
- Soportan más capacidades, como sentencias compuestas anidadas, que los procedimientos SQL externos.
- Db2 puede gestionar múltiples versiones de estos procedimientos por usted.
- Puede especificar que el procedimiento SQL se ejecute de forma autónoma, sin comprometer el trabajo de la aplicación que lo llama.

Todos los procedimientos SQL que se crean sin las opciones FENCED o EXTERNAL en la instrucción CREATE PROCEDURE son procedimientos SQL nativos.

Procedimientos de SQL externo (en desuso)

Un *procedimiento SQL externo* es un procedimiento cuyo cuerpo se ha escrito en SQL en su totalidad. El cuerpo está escrito en el lenguaje de procedimientos de SQL (SQL PL). No obstante, los procedimientos SQL externo se crean, implementan y ejecutan como cualquier otro procedimiento almacenado externo.

Función en desuso: Los procedimientos de SQL externo están en desuso y no están totalmente soportados como procedimientos SQL nativos. Para obtener mejores resultados, cree procedimientos de SQL nativo en su lugar. Para obtener más información, consulte “[Creación de procedimientos SQL](#)

nativos” en la página 244 y “Migración de un procedimiento de SQL externo a un procedimiento de SQL nativo” en la página 307.

Todos los procedimientos SQL que se crearon antes de DB2 9 son procedimientos SQL externos. A partir de la versión DB2 9, puede crear un procedimiento SQL externo especificando FENCED o EXTERNAL en la instrucción CREATE PROCEDURE.

Cuerpo de un procedimiento SQL

El cuerpo de un procedimiento SQL contiene una o más sentencias SQL. En el cuerpo del procedimiento SQL, también puede declarar y utilizar variables, condiciones, códigos de retorno, sentencias, cursores y manejadores.

Declaraciones que puede incluir en el cuerpo de un procedimiento SQL

Una declaración CREATE PROCEDURE para un procedimiento SQL nativo contiene *un cuerpo de rutina SQL*, tal como se define en [Instrucción CREATE PROCEDURE \(SQL - procedimiento nativo\) \(Db2 SQL\)](#). El diagrama de sintaxis para *el cuerpo de la rutina SQL* define el cuerpo del procedimiento como una sola instrucción SQL. La instrucción SQL puede ser una de las instrucciones SQL que se muestran en el diagrama de sintaxis para *el cuerpo de la rutina SQL*, o una instrucción de control SQL. El diagrama de sintaxis para *la instrucción de control SQL* en [Lenguaje de procedimiento de SQL \(SQL PL\) \(Db2 SQL\)](#) identifica las instrucciones de control que se pueden especificar.

Un procedimiento SQL nativo puede contener varias sentencias SQL si la sentencia SQL más externa es *una sentencia de control SQL* que incluye otras sentencias SQL. Estas sentencias se definen como sentencias de procedimiento de SQL. El diagrama de sintaxis en [Declaración de procedimiento SQL \(SQL PL\) \(Db2 SQL\)](#) identifica las sentencias SQL que se pueden especificar dentro de una sentencia de control. Las notas de sintaxis para *la instrucción de procedimiento SQL* aclaran las instrucciones SQL que están permitidas en un procedimiento SQL nativo.

Ejemplos

Los siguientes ejemplos muestran cómo determinar si una instrucción SQL está permitida en un procedimiento SQL.

Los diagramas de sintaxis para las instrucciones de control indican dónde se necesitan puntos y comas en un procedimiento SQL. Si el procedimiento contiene una sola instrucción que no sea de control, como en el Ejemplo 1, entonces no hay punto y coma en la instrucción CREATE PROCEDURE. Si el procedimiento consta de varias sentencias, como en el Ejemplo 2, utilice punto y coma para separar las sentencias SQL dentro del procedimiento SQL. No ponga punto y coma después de la declaración de control más externa.

Ejemplo 1

```
CREATE PROCEDURE UPDATE_SALARY_1
  (IN EMPLOYEE_NUMBER CHAR(10),
   IN RATE DECIMAL(6,2))
LANGUAGE SQL
MODIFIES SQL DATA
DETERMINISTIC
COMMIT ON RETURN YES
  UPDATE EMP [A]
    SET SALARY = SALARY * RATE
  WHERE EMPNO = EMPLOYEE_NUMBER
```

La instrucción UPDATE (**A**) es una instrucción SQL que está permitida porque aparece en el diagrama de sintaxis para *el cuerpo de la rutina SQL*.

Ejemplo 2

```
CREATE PROCEDURE GETWEEKENDS(IN MYDATES DATEARRAY, OUT WEEKENDS DATEARRAY)
BEGIN [A]
  -- ARRAY INDEX VARIABLES
  DECLARE DATEINDEX, WEEKENDINDEX INT DEFAULT 1; [B]
  -- VARIABLE TO STORE THE ARRAY LENGTH OF MYDATES,
  -- INITIALIZED USING THE CARDINALITY FUNCTION.
```

```

DECLARE DATESCOUNT INT; B
SET DATESCOUNT = CARDINALITY(MYDATES); C
-- FOR EACH DATE IN MYDATES, IF THE DATE IS A SUNDAY OR SATURDAY,
-- ADD IT TO THE OUTPUT ARRAY NAMED "WEEKENDS"
WHILE DATEINDEX <= DATESCOUNT DO D
  IF DAYOFWEEK(MYDATES[DATEINDEX]) IN (1, 7) THEN E
    SET WEEKENDS[WEEKENDINDEX] = MYDATES[DATEINDEX]; C
    SET WEEKENDINDEX = WEEKENDINDEX + 1; C
  END IF;
  SET DATEINDEX = DATEINDEX + 1; C
END WHILE;
END A

```

El procedimiento SQL tiene las siguientes palabras clave y sentencias:

- Las palabras clave BEGIN y END (**A**) indican el principio y el final de una instrucción compuesta.
- Las declaraciones DECLARE (**B**) son componentes de una sentencia compuesta y definen variables SQL dentro de la sentencia compuesta.
- Las declaraciones de asignación SET (**C**) son instrucciones de control SQL que asignan valores a variables SQL.
- La sentencia WHILE (**D**) y la instrucción IF (**E**) son instrucciones de control SQL.

Una sentencia compuesta es una sentencia de control SQL. Las sentencias de control SQL están permitidas en el cuerpo del procedimiento SQL porque *la sentencia de control SQL* aparece en el diagrama de sintaxis para *el cuerpo de la rutina SQL* de una sentencia CREATE PROCEDURE (SQL - nativa).

Conceptos relacionados

[Sentencias compuestas anidadas en procedimientos de SQL nativo](#)

Las sentencias compuestas anidadas son bloques de sentencias SQL que están contenidos por otros bloques de sentencias SQL en procedimientos SQL nativos. Utilice sentencias compuestas anidadas para definir manejadores de condiciones que ejecutan más de una sentencia y para definir diferentes ámbitos para variables y manejadores de condiciones.

Parámetros de procedimientos almacenados

Puede pasar información entre un procedimiento almacenado y el programa de aplicación de llamada utilizando parámetros. Las aplicaciones pasan los parámetros necesarios en la sentencia CALL de SQL. Opcionalmente, la aplicación también puede incluir una variable de indicador con cada parámetro para permitir valores nulos o pasar valores gramdes de parámetro de salida.

[Promoción de tipos de datos \(Db2 SQL\)](#)

Referencia relacionada

[sentencia compuesta \(Db2 SQL\)](#)

Variables en procedimientos de SQL

Para los datos que utilice únicamente dentro de un procedimiento SQL, puede declarar *variables SQL* y almacenar los valores en las variables. Las variables de SQL son similares a las variables de host en los procedimientos almacenados externos. Las variables de SQL se pueden definir con los mismos tipos de datos y longitudes que los parámetros de procedimiento de SQL.

Una declaración de variable SQL tiene la siguiente forma:

```
DECLARE SQL-variable-name data-type;
```

Una variable SQL se define en una instrucción compuesta. Es posible hacer referencia a las variables de SQL en cualquier parte de la sentencia compuesta en la que están declaradas, incluyendo una sentencia SQL que está anidada de forma directa o indirecta dentro de dicha sentencia compuesta. Para obtener más información, consulte [Referencias a parámetros y variables SQL en SQL PL \(Db2 SQL\)](#).

Puede realizar cualquier operación en variables SQL que pueda realizar en variables de host en sentencias SQL.

Conceptos relacionados

[Variables de host](#)

Utilice variables host para pasar un único elemento de datos entre Db2 y la aplicación.

Uso de variables host en sentencias SQL

Utilice variables de host escalares en sentencias de SQL incorporado para representar un único valor.

Las variables host son útiles para almacenar datos recuperados o para pasar valores que van a asignar o utilizar las comparaciones.

Tareas relacionadas

Controlar el ámbito de variables en un procedimiento de SQL

Utilice sentencias compuestas anidadas dentro de un procedimiento SQL para definir el ámbito de variables SQL. Puede hacer referencia a la variable solo dentro de la sentencia compuesta en la que se ha declarado y dentro de cualquier sentencia anidada.

Ejemplos de procedimientos SQL

Puede utilizar sentencias de tipo " CASE ", sentencias compuestas y sentencias anidadas dentro del cuerpo de un procedimiento SQL.

Ejemplo: instrucción " CASE ": El siguiente procedimiento SQL muestra cómo utilizar una instrucción " CASE ". El procedimiento recibe el número de identificación y la calificación de un empleado como parámetros de entrada. La sentencia " CASE " modifica el salario y la bonificación del empleado, utilizando una sentencia "UPDATE" diferente para cada una de las posibles calificaciones.

```
CREATE PROCEDURE UPDATESALARY2
  (IN EMPNUMBR CHAR(6),
   IN RATING INT)
LANGUAGE SQL
MODIFIES SQL DATA
CASE RATING
  WHEN 1 THEN
    UPDATE CORPDATA.EMPLOYEE
      SET SALARY = SALARY * 1.10, BONUS = 1000
      WHERE EMPNO = EMPNUMBR;
  WHEN 2 THEN
    UPDATE CORPDATA.EMPLOYEE
      SET SALARY = SALARY * 1.05, BONUS = 500
      WHERE EMPNO = EMPNUMBR;
  ELSE
    UPDATE CORPDATA.EMPLOYEE
      SET SALARY = SALARY * 1.03, BONUS = 0
      WHERE EMPNO = EMPNUMBR;
END CASE
```

Ejemplo: Sentencia compuesta con sentencias IF y WHILE anidadas : El siguiente ejemplo muestra una sentencia compuesta que incluye una sentencia IF, una sentencia WHILE y sentencias de asignación. El ejemplo también muestra cómo declarar variables SQL, cursos y controladores para clases de códigos de error.

El procedimiento recibe un número de departamento como parámetro de entrada. Una sentencia WHILE en el cuerpo del procedimiento obtiene el salario y la bonificación de cada empleado del departamento y utiliza una variable SQL para calcular un total acumulado de los salarios de los empleados del departamento. Una instrucción IF dentro de la instrucción WHILE comprueba si hay bonificaciones positivas y aumenta una variable SQL que cuenta el número de bonificaciones en el departamento. Cuando se han procesado todos los registros de empleados del departamento, se produce una condición de NO ENCONTRADO. Un controlador de condición NOT FOUND hace que la condición de búsqueda para la sentencia WHILE sea falsa, por lo que la ejecución de la sentencia WHILE finaliza. Las declaraciones de asignación asignan entonces el total de los salarios de los empleados y el número de bonificaciones del departamento a los parámetros de salida del procedimiento almacenado.

Si alguna instrucción SQL en la instrucción compuesta P1 recibe un error, el control pasa al controlador SQLEXCEPTION. La acción del controlador establece el parámetro de salida DEPTSALARY en NULL. Una vez que la acción del controlador se ha completado correctamente, se resuelve la condición de error original (SQLSTATE '00000', SQLCODE 0). Debido a que este controlador es un controlador EXIT, la ejecución pasa al final de la sentencia compuesta y el procedimiento SQL finaliza.

```
CREATE PROCEDURE RETURNDEPTSALARY
  (IN DEPTNUMBER CHAR(3),
   OUT DEPTSALARY DECIMAL(15,2),
```

```

OUT DEPTBONUSCNT INT)
LANGUAGE SQL
READS SQL DATA
P1: BEGIN
    DECLARE EMPLOYEE_SALARY DECIMAL(9,2);
    DECLARE EMPLOYEE_BONUS DECIMAL(9,2);
    DECLARE TOTAL_SALARY DECIMAL(15,2) DEFAULT 0;
    DECLARE BONUS_CNT INT DEFAULT 0;
    DECLARE END_TABLE INT DEFAULT 0;
    DECLARE C1 CURSOR FOR
        SELECT SALARY, BONUS FROM CORPDATA.EMPLOYEE
        WHERE WORKDEPT = DEPTNUMBER;
    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET END_TABLE = 1;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        SET DEPTSALARY = NULL;
    OPEN C1;
    FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
    WHILE END_TABLE = 0 DO
        SET TOTAL_SALARY = TOTAL_SALARY + EMPLOYEE_SALARY + EMPLOYEE_BONUS;
        IF EMPLOYEE_BONUS > 0 THEN
            SET BONUS_CNT = BONUS_CNT + 1;
        END IF;
        FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
    END WHILE;
    CLOSE C1;
    SET DEPTSALARY = TOTAL_SALARY;
    SET DEPTBONUSCNT = BONUS_CNT;
END P1

```

Ejemplo: Instrucción compuesta con instrucciones SQL dinámicas: El siguiente ejemplo muestra una sentencia compuesta que incluye sentencias SQL dinámicas.

El procedimiento recibe un número de departamento (P_DEPT) como parámetro de entrada. En la declaración compuesta, se construyen, preparan y ejecutan tres cadenas de declaración:

- La primera cadena de instrucciones ejecuta una instrucción DROP para garantizar que la tabla que se va a crear no existe ya. El nombre de la tabla es la concatenación del valor constante TABLE_PREFIX, el valor del parámetro P_DEPT y el valor constante TABLE_SUFFIX.
- La siguiente cadena de instrucciones ejecuta una instrucción CREATE para crear *DEPT_deptno_T*.
- La tercera cadena de instrucciones inserta filas para empleados en el departamento *deptno* en *DEPT_deptno_T*.

Al igual que las cadenas de instrucciones que se preparan en los programas de lenguaje de host no pueden contener variables de host, las cadenas de instrucciones en los procedimientos SQL no pueden contener variables SQL ni parámetros de procedimientos almacenados. Por lo tanto, la tercera cadena de la declaración contiene un marcador de parámetro que representa P_DEPT. Cuando se ejecuta la sentencia preparada, el marcador de parámetro se sustituye por el parámetro P_DEPT.

```

CREATE PROCEDURE CREATEDEPTTABLE (IN P_DEPT CHAR(3))
LANGUAGE SQL
BEGIN
    DECLARE STMT CHAR(1000);
    DECLARE MESSAGE CHAR(20);
    DECLARE TABLE_NAME CHAR(30);
    DECLARE TABLE_PREFIX VARCHAR(15) CONSTANT 'DEPT_';
    DECLARE TABLE_SUFFIX VARCHAR(15) CONSTANT '_T';
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
        SET MESSAGE = 'ok';
    SET TABLE_NAME = TABLE_PREFIX||P_DEPT||TABLE_SUFFIX;
    SET STMT = 'DROP TABLE '||TABLE_NAME;
    PREPARE S1 FROM STMT;
    EXECUTE S1;
    SET STMT = 'CREATE TABLE '||TABLE_NAME||
        '( EMPNO CHAR(6) NOT NULL, ||
        'FIRSTNME VARCHAR(6) NOT NULL, ||
        'MIDINIT CHAR(1) NOT NULL, ||
        'LASTNAME CHAR(15) NOT NULL, ||
        'SALARY DECIMAL(9,2))';
    PREPARE S2 FROM STMT;
    EXECUTE S2;
    SET STMT = 'INSERT INTO TABLE '||TABLE_NAME ||
        'SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY ||
        'FROM EMPLOYEE ||'

```

```
'WHERE WORKDEPT = ?';  
PREPARE S3 FROM STMT;  
EXECUTE S3 USING P_DEPT;  
END
```

Procedimientos autónomos

Los procedimientos autónomos se ejecutan bajo sus propias unidades de trabajo, de forma separada al programa de llamada, y se confirman cuando finalizan sin confirmar el trabajo del programa de llamada.

Los procedimientos autónomos se ejecutan como unidades de trabajo independientes de los programas de la aplicación que los invocan. Los procedimientos autónomos siguen las reglas de la opción COMMIT ON RETURN YES para sus cambios antes de volver a la persona que llama. Sin embargo, su confirmación no afecta a los cambios realizados por el programa de la aplicación que realiza la llamada. El programa de aplicación de llamada controla cuándo se confirman o se revierten sus propias actualizaciones.

Si la aplicación que llama revierte sus propios cambios, los cambios confirmados del procedimiento autónomo no se ven afectados. Por lo tanto, los procedimientos autónomos son útiles para registrar información sobre las condiciones de error encontradas por un programa de aplicación. Cuando la aplicación encuentra el error y revierte sus propios cambios, los cambios confirmados del procedimiento autónomo permanecen disponibles.

Los programas de aplicación normales, otros procedimientos almacenados, funciones definidas por el usuario o activadores pueden invocar procedimientos autónomos. Los procedimientos autónomos pueden completar los siguientes tipos de trabajo:

- Ejecutar instrucciones SQL
- Invocar otro procedimiento, función o activador, siempre que el número de niveles anidados no supere los 64 y el procedimiento llamado no sea autónomo.
- Ejecutar instrucciones COMMIT y ROLLBACK que se apliquen a las operaciones SQL ejecutadas por procesos anidados dentro del procedimiento autónomo.

Las siguientes restricciones se aplican a los procedimientos autónomos:

- Solo los procedimientos SQL nativos pueden definirse como autónomos.
- Los procedimientos autónomos y los procedimientos anidados, los desencadenantes y las funciones dentro de los procedimientos autónomos no pueden invocar otros procedimientos autónomos.
- Los procedimientos autónomos no pueden ver los cambios no confirmados de la aplicación de llamada.
- Cuando existen varias versiones de un procedimiento, todas las versiones deben definirse como autónomas.
- Los procedimientos autónomos no comparten bloqueos con la aplicación que los llama, lo que significa que el procedimiento autónomo podría agotar el tiempo de espera debido a la contención de bloqueo con la aplicación que lo llama.
- El paralelismo está desactivado para los procedimientos autónomos. Todas las declaraciones en un procedimiento autónomo y para cualquier nivel anidado dentro se ejecutan en modo de procesamiento secuencial.
- CONJUNTOS DE RESULTADOS DINÁMICOS Se debe especificar 0 cuando se utilizan procedimientos autónomos.
- Los parámetros de procedimiento almacenado no deben definirse como un tipo de datos LOB, ni como ningún otro tipo de datos distinto que se base en un valor LOB o XML.

Tareas relacionadas

[Control de procedimientos autónomos \(Db2 Administration Guide\)](#)

Procedimientos almacenados externos

Un *procedimiento almacenado externo* es un procedimiento que se escribe en un lenguaje de host y que puede contener sentencias SQL. El código fuente de los procedimientos externos es distinto de la definición.

Un procedimiento almacenado externo es muy parecido a cualquier otra aplicación SQL. Puede incluir sentencias de SQL estático o dinámico, llamadas IFI y mandatos de Db2 que se emiten a través de IFI. Los procedimientos almacenados externos se preparan como prepararía normalmente los programas de aplicación. Ha de precompilarlos, compilarlos y editar los enlaces de éstos. A continuación, ha de vincular el DBRM en un paquete. También debe definir el procedimiento para Db2 mediante la sentencia CREATE PROCEDURE. De este modo, el código fuente para un procedimiento almacenado externo es distinto de la definición para el procedimiento almacenado.

Requisitos de idioma para el procedimiento almacenado externo y su llamador

Puede escribir un procedimiento almacenado externo en Assembler, C, C++, COBOL, Java, REXX o PL/I. Todos los programas deben estar diseñados para ejecutarse utilizando el entorno de lenguaje. Los procedimientos almacenados COBOL y C++ pueden contener extensiones orientadas a objetos.

El programa que llama al procedimiento almacenado puede estar en cualquier lenguaje que dé soporte a la sentencia SQL CALL. ODBC las aplicaciones pueden utilizar una cláusula de escape para pasar una llamada de procedimiento almacenado a Db2.

Conceptos relacionados

Extensiones orientadas a objetos en COBOL

Cuando utiliza extensiones orientadas a objetos en una aplicación COBOL, debe considerar dónde colocar sentencias SQL, SQLCA, SQLDA y declaraciones de variables host. También debe considerar las reglas para las variables host.

Procedimientos almacenados de REXX

Un procedimiento almacenado REXX es similar a cualquier otro procedimiento REXX y sigue las mismas reglas que los procedimientos almacenados en otros lenguajes. Un procedimiento almacenado REXX recibe parámetros de entrada, ejecuta comandos REXX, ejecuta opcionalmente sentencias SQL y devuelve como máximo un parámetro de salida. Sin embargo, hay algunas diferencias.

Procedimientos almacenados y funciones definidas por el usuario de Java (Db2 Application Programming for Java)

Diferencias entre procedimientos SQL nativos y procedimientos externos

Los procedimientos SQL se escriben completamente en sentencias SQL. Los procedimientos externos están escritos en un lenguaje host y pueden contener instrucciones SQL. Puede invocar ambos tipos de procedimientos con una instrucción SQL CALL. Sin embargo, debe tener en cuenta varias diferencias importantes en cuanto a comportamiento y preparación.

Los procedimientos nativos de SQL y los procedimientos externos difieren de las siguientes maneras:

Cómo gestionan los errores

- Para un procedimiento SQL, Db2 devuelve automáticamente las condiciones SQL en el SQLCA cuando el procedimiento no incluye una instrucción RETURN o un controlador. Para obtener información sobre las diversas formas de manejar errores en un procedimiento SQL, consulte [“Gestión de condiciones SQL en un procedimiento SQL” en la página 250](#).
- Para un procedimiento almacenado externo, Db2 no devuelve las condiciones SQL en el SQLCA a la aplicación que lo invoca. Si utiliza PARAMETER STYLE SQL al definir un procedimiento externo, puede configurar SQLSTATE para indicar un error antes de que finalice el procedimiento. Para valores válidos de SQLSTATE, consulte [Valores de SQLSTATE y códigos de error comunes \(Db2 Codes\)](#).

Cómo especifican el código para el procedimiento almacenado

Las definiciones de procedimientos SQL contienen el código fuente del procedimiento almacenado. Una definición de procedimiento almacenado externo especifica el nombre del programa de procedimiento almacenado.

Cómo define el procedimiento almacenado.

Tanto para los procedimientos SQL nativos como para los procedimientos externos, se define el procedimiento almacenado a Db2 ejecutando la instrucción CREATE PROCEDURE. Para los procedimientos externos, también debe vincular por separado el código fuente del procedimiento

en un paquete. Puede hacerlo antes o después de emitir la instrucción CREATE PROCEDURE para definir el procedimiento externo.

ejemplos

Creación de un procedimiento de SQL nativo

El siguiente ejemplo muestra una definición para un procedimiento SQL.

```
CREATE PROCEDURE UPDATESALARY1  
  (IN EMPNUMBR CHAR(10),  
   IN RATE DECIMAL(6,2))  
LANGUAGE SQL  
  UPDATE EMP  
    SET SALARY = SALARY * RATE  
  WHERE EMPNO = EMPNUMBR
```

1
2
3
4

Notas:

- 1** El nombre del procedimiento almacenado es UPDATESALARY1.
- 2** Los dos parámetros tienen tipos de datos CHAR(10) y DECIMAL(6,2). Ambos son parámetros de entrada.
- 3** IDIOMA SQL indica que se trata de un procedimiento SQL, por lo que un cuerpo de procedimiento sigue a los otros parámetros.
- 4** El cuerpo del procedimiento consiste en una única instrucción SQL UPDATE, que actualiza las filas de la tabla de empleados.

Creación de un procedimiento almacenado externo

El siguiente ejemplo muestra una definición para un procedimiento almacenado externo equivalente que está escrito en COBOL. El programa de procedimiento almacenado, que actualiza los salarios de los empleados, se llama UPDSAL.

```
CREATE PROCEDURE UPDATESALARY1  
  (IN EMPNUMBR CHAR(10),  
   IN RATE DECIMAL(6,2))  
LANGUAGE COBOL  
  EXTERNAL NAME UPDSAL;
```

1
2
3
4

Notas:

- 1** El nombre del procedimiento almacenado es UPDATESALARY1.
- 2** Los dos parámetros tienen tipos de datos CHAR(10) y DECIMAL(6,2). Ambos son parámetros de entrada.
- 3** IDIOMA COBOL indica que se trata de un procedimiento externo, por lo que el código del procedimiento almacenado se encuentra en un programa COBOL independiente.
- 4** El nombre del módulo de carga que contiene el programa de procedimiento almacenado ejecutable es UPDSAL.

Referencia relacionada

[Instrucción CREATE PROCEDURE \(resumen\) \(Db2 SQL\)](#)

[Instrucción CREATE PROCEDURE \(procedimiento externo\) \(Db2 SQL\)](#)

[Instrucción CREATE PROCEDURE \(SQL - procedimiento nativo\) \(Db2 SQL\)](#)

Sentencias COMMIT y ROLLBACK en un procedimiento almacenado

Cuando emite sentencias COMMIT o ROLLBACK en su procedimiento almacenado, Db2 confirma o retrotrae todos los cambios en la unidad de trabajo.

Para los procedimientos que no se definen como autónomos, los cambios confirmados o retrotraídos incluyen cambios que puede realizar la aplicación cliente antes de llamar el procedimiento almacenado y el trabajo Db2 que realiza el procedimiento almacenado. Para los procedimientos autónomos, los cambios confirmados o revertidos incluyen solo el trabajo realizado por la unidad de trabajo almacenada para el procedimiento almacenado.

Si su procedimiento almacenado incluye sentencias COMMIT o ROLLBACK, defínalo con una de las siguientes cláusulas:

- CONTAINS SQL
- READS SQL DATA
- MODIFIES SQL DATA

La cláusula COMMIT ON RETURN en una definición de procedimiento almacenado no tiene efecto en las sentencias COMMIT o ROLLBACK en el código de procedimiento almacenado. Si especifica COMMIT ON RETURN YES al definir el procedimiento almacenado, Db2 emite una instrucción COMMIT cuando el control vuelve del procedimiento almacenado. Esta acción se produce independientemente de si el procedimiento almacenado contiene instrucciones COMMIT o ROLLBACK.

Si especifica AUTÓNOMO al definir el procedimiento almacenado, el procedimiento autónomo es una unidad de trabajo independiente de la aplicación que lo llama. Db2 emite una instrucción COMMIT cuando el control regresa del procedimiento almacenado, pero solo se confirman los cambios completados por el procedimiento autónomo. Del mismo modo, las sentencias COMMIT o ROLLBACK en el código de procedimiento autónomo tampoco tienen efecto en el trabajo realizado por la aplicación que llama.

Una sentencia ROLLBACK tiene el mismo efecto en los cursos de un procedimiento almacenado que en los cursos de programas independientes. Una instrucción ROLLBACK cierra todos los cursos abiertos. Una declaración COMMIT en un procedimiento almacenado cierra los cursos que no están declarados WITH HOLD y deja abiertos los cursos que están declarados WITH HOLD. El efecto de COMMIT o ROLLBACK en los cursos se aplica a los cursos que se declaran en la aplicación de llamada y a los cursos que se declaran en el procedimiento almacenado.

Restricción: No puede incluir instrucciones COMMIT o ROLLBACK en un procedimiento almacenado si se cumple alguna de las siguientes condiciones:

- El procedimiento almacenado está anidado dentro de un desencadenador o función definida por el usuario.
- El procedimiento almacenado es llamado por un cliente que utiliza el procesamiento de confirmación en dos fases.
- El programa cliente utiliza una conexión de tipo 2 para conectarse al servidor remoto que contiene el procedimiento almacenado.
- Db2 no es el coordinador de confirmación.

Si una instrucción COMMIT o ROLLBACK en un procedimiento almacenado infringe alguna de estas condiciones, Db2 pone la transacción en un estado de reversión obligatoria. Además, en este caso, la instrucción CALL falla.

Referencia relacionada

[CALL declaración \(Db2 SQL\)](#)

[COMMIT declaración \(Db2 SQL\)](#)

[ROLLBACK declaración \(Db2 SQL\)](#)

Registros especiales en un procedimiento almacenado

Puede utilizar todos los registros especiales en un procedimiento almacenado. Sin embargo, puede modificar solo algunos de estos registros especiales. Después de que se complete un procedimiento almacenado, Db2 restaura todos los registros especiales a los valores que tenían antes de la invocación.

Creación de procedimientos SQL nativos

Un *procedimiento de SQL nativo* es un procedimiento cuyo cuerpo se ha escrito en SQL en su totalidad y que se crea emitiendo una única sentencia SQL, CREATE PROCEDURE.

Antes de empezar

Antes de crear un procedimiento SQL nativo, configure Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario durante la instalación o configure Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario durante la migración si el procedimiento SQL nativo cumple al menos una de las siguientes condiciones:

- El procedimiento SQL nativo llama al menos a un procedimiento almacenado externo, un procedimiento SQL externo o una función definida por el usuario.
- El procedimiento SQL nativo se define con ALLOW DEBUG MODE o DISALLOW DEBUG MODE. Si especifica DISABLE DEBUG MODE, no necesita configurar el entorno de procedimiento almacenado.

Acerca de esta tarea

Un *procedimiento de SQL nativo* es un procedimiento cuyo cuerpo se ha escrito en SQL en su totalidad. El cuerpo está escrito en el lenguaje de procedimientos de SQL (SQL PL). Un procedimiento de SQL nativo se crea emitiendo una única sentencia SQL, CREATE PROCEDURE. Los procedimientos de SQL nativos no necesitan ninguna otra preparación de programa, como la precompilación, la compilación o la edición de enlaces del código fuente. Los procedimientos de SQL nativos se ejecutan como sentencias de SQL que se han vinculado en un paquete de Db2. Los procedimientos de SQL nativo no tienen un programa de aplicación externo asociado. Los procedimientos de SQL nativo son más compatibles y más fáciles de mantener y normalmente funcionan mejor que los procedimientos SQL externos, que están en desuso.

Procedimiento

Para crear un procedimiento SQL nativo, realice una de las siguientes acciones:

- Utilice una herramienta como Db2 Developer Extension para especificar las sentencias de origen para el procedimiento SQL e implemente el procedimiento SQL en Db2. Para obtener más información, consulte [IBM Db2 for z/OS Developer Extension for Visual Studio Code](#).
- Utilice IBM Data Studio para especificar las sentencias de origen para el procedimiento SQL e implemente el procedimiento SQL en Db2.
IBM Data Studio también le permite crear copias del paquete de procedimientos según sea necesario e implementar el procedimiento en servidores remotos.
- Implemente manualmente el procedimiento SQL nativo completando los siguientes pasos:
 - a) Emitir la declaración CREATE PROCEDURE:
 - Incluir un cuerpo de procedimiento escrito íntegramente en el lenguaje de procedimientos SQL (SQL PL). Para obtener más información sobre lo que puede hacer dentro del cuerpo del procedimiento, consulte *Cuerpo de la rutina SQL* en [Instrucción CREATE PROCEDURE \(SQL - procedimiento nativo\) \(Db2 SQL\)](#), *Instrucción de control SQL* en [Lenguaje de procedimiento de SQL \(SQL PL\) \(Db2 SQL\)](#) y la siguiente información:
 - “[Controlar el ámbito de variables en un procedimiento de SQL](#)” en la página 245
 - “[Declaración de cursos en un procedimiento SQL con sentencias compuestas anidadas](#)” en la página 249
 - “[Gestión de condiciones SQL en un procedimiento SQL](#)” en la página 250

- “Generación de una condición dentro de un procedimiento SQL utilizando las sentencias SIGNAL o RESIGNAL” en la página 259
- No incluya las palabras clave FENCED o EXTERNAL, que especifican la creación de procedimientos SQL externos, que están en desuso.
- Puede especificar la palabra clave AUTONOMOUS para permitir que el procedimiento se comprometa sin comprometer el trabajo de la aplicación que llama. Los procedimientos autónomos no pueden ver los cambios no confirmados de la aplicación de llamada, y no pueden llamar a otros procedimientos autónomos.

Cuando emite esta declaración CREATE PROCEDURE, la primera versión de este procedimiento se define como Db2, y un paquete se vincula implícitamente con las opciones que especifique en la declaración CREATE PROCEDURE.

- b) Si el procedimiento SQL nativo contiene una o más de las siguientes declaraciones o referencias, haga copias del paquete de procedimientos SQL nativo, según sea necesario:
 - CONNECT
 - SET CURRENT PACKAGESET
 - SET CURRENT PACKAGE PATH
 - Una referencia de tabla con un nombre de tres partes que hace referencia a una ubicación distinta del servidor actual o hace referencia a un alias que se resuelve en dicho nombre.
- c) Si planea llamar al procedimiento nativo de SQL en otro servidor de Db2 , implemente el procedimiento en otro servidor de Db2 for z/OS . Puede personalizar las opciones de encuadernación al mismo tiempo.
- d) Autorizar a los usuarios adecuados a llamar al procedimiento almacenado.

Qué hacer a continuación

Después de crear un procedimiento SQL nativo, puede crear versiones adicionales del procedimiento según sea necesario. Para obtener más información, consulte “[Creación de nuevas versiones de procedimientos SQL nativos](#)” en la página 264.

Conceptos relacionados

[Procedimientos de SQL](#)

Un procedimiento de SQL es un procedimiento almacenado que solo contiene sentencias SQL.

[Cuerpo de un procedimiento SQL](#)

El cuerpo de un procedimiento SQL contiene una o más sentencias SQL. En el cuerpo del procedimiento SQL, también puede declarar y utilizar variables, condiciones, códigos de retorno, sentencias, cursores y manejadores.

Tareas relacionadas

[Implementación de procedimientos almacenados de Db2 \(Db2 Administration Guide\)](#)

[Desarrollo de rutinas de base de datos \(IBM Data Studio, IBM Optim Database Administrator, IBM infoSphere Data Architect, IBM Optim Development Studio\)](#)

Referencia relacionada

[Instrucción CREATE PROCEDURE \(SQL - procedimiento nativo\) \(Db2 SQL\)](#)

Controlar el ámbito de variables en un procedimiento de SQL

Utilice sentencias compuestas anidadas dentro de un procedimiento SQL para definir el ámbito de variables SQL. Puede hacer referencia a la variable solo dentro de la sentencia compuesta en la que se ha declarado y dentro de cualquier sentencia anidada.

Procedimiento

Para controlar el alcance de una variable en un procedimiento SQL:

1. Declare la variable dentro de la sentencia compuesta en la que desea hacer referencia a ella. Asegúrese de que el nombre de la variable sea único dentro de la sentencia compuesta, sin

incluir ninguna sentencia anidada. Puede definir variables con el mismo nombre en otras sentencias compuestas en el mismo procedimiento SQL.

2. Haga referencia a la variable dentro de esa sentencia compuesta o de cualquier sentencia anidada.

Recomendación: Si existen varias variables con el mismo nombre dentro de un procedimiento SQL, califique la variable con la etiqueta de la sentencia compuesta en la que se declaró. De lo contrario, podría referirse accidentalmente a la variable incorrecta.

Si el nombre de la variable no está cualificado y existen varias variables con ese nombre dentro del mismo ámbito, Db2 utiliza la variable de la instrucción compuesta más interna.

Ejemplo

El siguiente ejemplo contiene tres declaraciones de la variable A. Un ejemplo se declara en la sentencia outer compound, que tiene la etiqueta " OUTER1 ". Los otros casos se declaran en las sentencias compuestas internas con las etiquetas " INNER1 " y " INNER2 ". En la sentencia compuesta " INNER1 ", " Db2 " asume que las referencias no cualificadas a "A" en la sentencia "assignment" y en la sentencia "UPDATE" se refieren a la instancia de "A" que se declara en la sentencia compuesta " INNER1 ". Para referirse a la instancia de A que se declara en la sentencia compuesta de tipo " OUTER1 ", califique la variable como " OUTER1.A ".

```
CREATE PROCEDURE P2 ()  
LANGUAGE SQL  
  
-- Outermost compound statement -----  
OUTER1: BEGIN [1]  
    DECLARE A INT DEFAULT 100;  
  
    -- Inner compound statement with label INNER1 ---  
    INNER1: BEGIN [2]  
        DECLARE A INT DEFAULT NULL;  
        DECLARE W INT DEFAULT NULL;  
  
        SET A = A + OUTER1.A; [3]  
  
        UPDATE T1 SET T1.B = 5  
        WHERE T1.B = A; [4]  
  
        SET OUTER1.A = 100; [5]  
  
        SET INNER1.A = 200; [6]  
    END INNER1; [7]  
    -- End of inner compound statement INNER1 -----  
  
    -- Inner compound statement with label INNER2 ---  
    INNER2: BEGIN [8]  
        DECLARE A INT DEFAULT NULL;  
        DECLARE Z INT DEFAULT NULL;  
  
        SET A = A + OUTER1.A;  
  
    END INNER2; [9]  
    -- End of inner compound statement INNER2 -----  
  
    SET OUTER1.A = 100; [10]  
  
END OUTER1 [11]
```

El ejemplo anterior tiene las siguientes partes:

1. El comienzo de la sentencia compuesta más externa, que tiene la etiqueta " OUTER1 ".
2. El comienzo de la sentencia del compuesto interno con la etiqueta INNER1.
3. La variable no cualificada A se refiere a INNER1.A.
4. La variable no cualificada A se refiere a INNER1.A.
5. OUTER1.A es una referencia válida, porque se hace referencia a esta variable en una sentencia compuesta anidada.

6. INNER1.A es una referencia válida, porque se hace referencia a esta variable en la misma sentencia compuesta en la que se declara. No puede hacer referencia a INNER2.A, porque esta variable no está en el ámbito de esta instrucción compuesta.
7. El final de la sentencia compuesta interna con la etiqueta INNER1.
8. El comienzo de la sentencia del compuesto interno con la etiqueta INNER2.
9. El final de la sentencia compuesta interna con la etiqueta INNER2.
10. OUTER1.A es una referencia válida, porque se hace referencia a esta variable en la misma sentencia compuesta en la que se declara. No puede hacer referencia a INNER1.A, porque esta variable se declara en una instrucción anidada y no se puede hacer referencia a ella en la instrucción externa.
11. El final de la sentencia compuesta más externa, que tiene la etiqueta " OUTER1 ".

Conceptos relacionados

Variables en procedimientos de SQL

Para los datos que utilice únicamente dentro de un procedimiento SQL, puede declarar *variables SQL* y almacenar los valores en las variables. Las variables de SQL son similares a las variables de host en los procedimientos almacenados externos. Las variables de SQL se pueden definir con los mismos tipos de datos y longitudes que los parámetros de procedimiento de SQL.

Referencias a parámetros y variables SQL en SQL PL (Db2 SQL)

Sentencias compuestas anidadas en procedimientos de SQL nativo

Las sentencias compuestas anidadas son bloques de sentencias SQL que están contenidos por otros bloques de sentencias SQL en procedimientos SQL nativos. Utilice sentencias compuestas anidadas para definir manejadores de condiciones que ejecutan más de una sentencia y para definir diferentes ámbitos para variables y manejadores de condiciones.

El siguiente pseudocódigo muestra una estructura básica de un procedimiento SQL con sentencias compuestas anidadas:

```
CREATE PROCEDURE...
OUTERMOST: BEGIN
  ...
    INNER1: BEGIN
      ...
        INNERMOST: BEGIN
          ...
            END INNERMOST;
        END INNER1;
    INNER2: BEGIN
      ...
        ...
      END INNER2;
  END OUTERMOST
```

En el código anterior, la sentencia compuesta OUTERMOST contiene dos sentencias compuestas anidadas: " INNER1 " y " INNER2 ". INNER1 contiene una sentencia compuesta anidada: INNERMOST.

Conceptos relacionados

Manejadores en un procedimiento SQL

Si se produce un error cuando se ejecuta el procedimiento SQL, el procedimiento finaliza a menos que incluya sentencias para indicar al procedimiento que realice alguna otra acción. Estas sentencias se denominan manejadores.

Tareas relacionadas

Definición de los manejadores de condiciones que ejecutan más de una sentencia

Un controlador de condiciones define la acción que un procedimiento SQL realiza cuando se produce una condición concreta. Debe especificar la acción como una sentencia de procedimiento SQL única.

Etiquetas de sentencias para sentencias compuestas anidadas en procedimientos SQL nativos

Puede definir una etiqueta para cada sentencia compuesta en un procedimiento SQL. Esta etiqueta le permite hacer referencia a este bloque de sentencias en otras sentencias, como las sentencias de control

GOTO, LEAVE e ITERATE SQL PL. También puede utilizar la etiqueta para calificar una variable cuando sea necesario. No se requieren etiquetas.

El nombre de una etiqueta debe cumplir los siguientes criterios:

- Sea único dentro de la sentencia compuesta, incluyendo cualquier sentencia compuesta que esté anidada dentro de la sentencia compuesta.
- No ser el mismo que el nombre del procedimiento SQL.

Puede hacer referencia a una etiqueta dentro de la sentencia compuesta en la que está definida, incluidas las sentencias compuestas anidadas dentro de esa sentencia compuesta.

Ejemplo de etiquetas de declaración: El siguiente ejemplo muestra varias etiquetas de declaración y su alcance:

```
CREATE PROCEDURE P1 ()  
LANGUAGE SQL  
  
--Outermost compound statement -----  
OUTER1: BEGIN [1]  
  
    --Inner compound statement with label INNER1 ---  
    INNER1: BEGIN [2]  
        IF...  
            ABC: LEAVE INNER1; [3]  
        ELSEIF  
            XYZ: LEAVE OUTER1; [4]  
        END IF  
  
    END INNER1;  
    --End of inner compound statement INNER1 -----  
  
    --Inner compound statement with label INNER2---  
    INNER2: BEGIN [5]  
        XYZ:...statement [6]  
    END INNER2;  
    -- End of inner compound statement INNER2 -----  
  
END OUTER1 [7]
```

El ejemplo anterior tiene las siguientes partes:

1. El comienzo de la declaración compuesta más externa, que está etiquetada OUTER1
2. El comienzo de una sentencia compuesta interna que está etiquetada INNER1
3. Una declaración LEAVE que se define con la etiqueta ABC. Esta instrucción LEAVE especifica que Db2 debe finalizar el procesamiento de la instrucción compuesta INNER1 y comenzar a procesar la siguiente instrucción, que es INNER2. Esta declaración LEAVE no puede especificar INNER2, porque esa etiqueta no está dentro del alcance de la declaración compuesta INNER1.
4. Una instrucción LEAVE que se define con la etiqueta XYZ. Esta sentencia LEAVE especifica que Db2 debe finalizar el procesamiento de la sentencia compuesta OUTER1 y comenzar a procesar la siguiente sentencia, si existe. Este ejemplo no muestra la siguiente declaración.
5. El comienzo de una sentencia compuesta interna que se etiqueta como " INNER2 ".
6. Una declaración que se define con la etiqueta XYZ. Esta etiqueta es aceptable aunque otra declaración en este procedimiento tenga la misma etiqueta, porque las dos etiquetas están en ámbitos diferentes. Ninguna de las dos etiquetas está incluida en el ámbito de la otra.
7. El final de la sentencia compuesta más externa que se etiqueta como " OUTER1 ".

Los siguientes ejemplos muestran usos válidos e inválidos de las etiquetas:

Ejemplo de etiquetas no válido:

```
L1: BEGIN  
L2: SET A = B;  
L1: GOTO L2: --This duplicate use of the label L1 causes an error, because  
      --the same label is already used in the same scope.  
  
END L1;
```

Ejemplo válido de etiquetas:

```
L1: BEGIN
L2: BEGIN
L4: BEGIN --This line contains the first use of the label L4
    DECLARE A CHAR(5);
    SET A = B;
    END L4;
END L2;

L3: BEGIN
L4: BEGIN --This second use of the label L4 is valid, because
          --it is used in a different scope.
    DECLARE A CHAR(5);
    SET A = B;
    END L4;
END L3;
END L1;
```

Declaración de cursos en un procedimiento SQL con sentencias compuestas anidadas

Cuando declara un cursor en un procedimiento SQL que tiene sentencias compuestas anidadas, no puede hacer referencia necesariamente al curso en cualquier punto del procedimiento. El ámbito del cursor se restringe a la sentencia compuesta en la que lo declara.

Procedimiento

Especifique la instrucción DECLARE CURSOR dentro de la instrucción compuesta en la que desea hacer referencia al cursor. Utilice un nombre de cursor que sea único dentro del procedimiento SQL.

Puede hacer referencia al cursor dentro de la sentencia compuesta en la que se declara y dentro de cualquier sentencia anidada. Si el cursor se declara como cursor de conjunto de resultados, incluso si el cursor no se declara en la sentencia compuesta más externa, cualquier aplicación que lo invoque puede hacer referencia a él.

Ejemplo

En el siguiente ejemplo, el cursor X se declara en la sentencia compuesta externa. Se puede hacer referencia a este cursor dentro del bloque exterior en el que se declaró y dentro de cualquier sentencia compuesta anidada.

```
CREATE PROCEDURE SINGLE_CSR
  (INOUT IR1 INT, INOUT JR1 INT, INOUT IR2 INT, INOUT JR2 INT)
  LANGUAGE SQL
  DYNAMIC RESULT SETS 2
BEGIN
  DECLARE I INT;
  DECLARE J INT;
  DECLARE X CURSOR WITH RETURN FOR --outer declaration for X
    SELECT * FROM CSRT1;

  SUB: BEGIN
    OPEN X;                      --references X in outer block
    FETCH X INTO I,J;            --references X in outer block
    SET IR1 = I;
    SET JR1 = J;
  END;

  FETCH X INTO I,J;            --references X in outer block
  SET IR2 = I;
  SET JR2 = j;
  CLOSE X;
END
```

Referencia relacionada

[Instrucción CREATE PROCEDURE \(SQL - procedimiento nativo\) \(Db2 SQL\)](#)

[DECLARE CURSOR declaración \(Db2 SQL\)](#)

Gestión de condiciones SQL en un procedimiento SQL

En un procedimiento SQL, puede especificar cómo gestiona el programa algunos errores y avisos de SQL.

Acerca de esta tarea

Si no incluye un controlador o una instrucción RETURN en el procedimiento SQL, Db2 devuelve automáticamente cualquier condición SQL al llamador en el SQLCA.

Procedimiento

Para manejar condiciones SQL, utilice una de las siguientes técnicas:

- Incluya instrucciones denominadas *controladores* para indicar el procedimiento para realizar alguna otra acción cuando se produzca un error o una advertencia.
- Incluir una instrucción RETURN en un procedimiento SQL para devolver un valor de estado entero al llamador.
- Incluya una sentencia SIGNAL o una sentencia RESIGNAL para generar un SQLSTATE específico y definir el texto del mensaje para ese SQLSTATE.
- Forzar la devolución de un SQLCODE negativo por parte de un procedimiento si un desencadenador llama al procedimiento.

Manejadores en un procedimiento SQL

Si se produce un error cuando se ejecuta el procedimiento SQL, el procedimiento finaliza a menos que incluya sentencias para indicar al procedimiento que realice alguna otra acción. Estas sentencias se denominan manejadores.

Los manipuladores son similares a las sentencias WHENEVER en programas de aplicaciones SQL externas. Los controladores indican al procedimiento SQL qué hacer cuando se produce un error o una advertencia, o cuando no se devuelven más filas de una consulta. Además, puede declarar controladores para SQLSTATEs específicos. Puede hacer referencia a un SQLSTATE por su número en un controlador, o puede declarar un nombre para el SQLSTATE y luego usar ese nombre en el controlador.

La forma general de una declaración del manipulador es:

```
DECLARE handler-type HANDLER FOR condition SQL-procedure-statement;
```

En general, la forma en que trabaja un controlador es que cuando se produce un error que coincide con la condición, se ejecuta la instrucción de procedimiento SQL. Cuando se completa la instrucción de procedimiento SQL, el controlador de tipo (Db2) realiza la acción indicada por el controlador de tipo.

Tipos de manipuladores

El tipo de controlador determina lo que ocurre después de completar la instrucción de procedimiento SQL. Puede declarar el tipo de controlador como CONTINUAR o SALIR:

CONTINUE

Especifica que, una vez completada la instrucción de procedimiento SQL, la ejecución continúa con la instrucción que sigue a la que causó el error.

EXIT

Especifica que, una vez completada la instrucción de procedimiento SQL, la ejecución continúa al final de la instrucción compuesta que contiene el controlador.

Ejemplo: controlador CONTINUE : este controlador establece el indicador at_end cuando no hay más filas que satisfagan una consulta. El controlador hace que la ejecución continúe después de la sentencia que no devolvió ninguna fila.

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET at_end=1;
```

Ejemplo: controlador EXIT : este controlador coloca la cadena «La tabla no existe» en el parámetro de salida OUT_BUFFER cuando se produce la condición NO_TABLE. NO_TABLE se declara previamente como

SQLSTATE 42704 (*el nombre* es un nombre no definido). El controlador hace que el procedimiento SQL salga de la sentencia compuesta en la que se declara el controlador.

```
DECLARE NO_TABLE CONDITION FOR '42704';
:
DECLARE EXIT HANDLER FOR NO_TABLE
    SET OUT_BUFFER='Table does not exist';
```

Definición de los manejadores de condiciones que ejecutan más de una sentencia

Un controlador de condiciones define la acción que un procedimiento SQL realiza cuando se produce una condición concreta. Debe especificar la acción como una sentencia de procedimiento SQL única.

Procedimiento

Para definir un controlador de condiciones que ejecute más de una instrucción cuando se produzca la condición especificada, especifique una instrucción compuesta dentro de la declaración de ese controlador.

ejemplos

Ejemplo

El siguiente ejemplo muestra un controlador de condiciones que captura el valor SQLSTATE y establece un indicador local en TRUE.

```
BEGIN
    DECLARE SQLSTATE CHAR(5);
    DECLARE PrvSQLState CHAR(5) DEFAULT '00000';
    DECLARE ExceptState INT;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
        BEGIN
            SET PrvSQLState = SQLSTATE;
            SET ExceptState = TRUE;
        END;
    ...
END
```

Ejemplo

El siguiente ejemplo declara un controlador de condiciones para SQLSTATE 72822. La siguiente sentencia SIGNAL está dentro del ámbito de este controlador de condiciones y, por lo tanto, activa este controlador. El controlador de condiciones comprueba el valor de la variable SQL VAR con una instrucción IF. Dependiendo del valor de VAR, se cambia el SQLSTATE y se establece el texto del mensaje.

```
DECLARE EXIT HANDLER FOR SQLSTATE '72822'
    IF ( VAR = 'OK' ) THEN
        RESIGNAL SQLSTATE '72623'
            SET MESSAGE_TEXT = 'Got SQLSTATE 72822';
    ELSE
        RESIGNAL SQLSTATE '72319'
            SET MESSAGE_TEXT = VAR;
    END IF;

SIGNAL SQLSTATE '72822';
```

Referencia relacionada

[sentencia compuesta \(Db2 SQL\)](#)

Controlar cómo se manejan los errores dentro de diferentes ámbitos en un procedimiento SQL

Puede utilizar sentencias compuestas anidadas en un procedimiento SQL para especificar que los errores se gestionen de forma diferente dentro de diferentes ámbitos. También puede asegurarse de que los controladores de condición se comprueben solo con una instrucción compuesta concreta.

Procedimiento

Para controlar cómo se manejan los errores dentro de diferentes ámbitos en un procedimiento SQL:

1. Opcional: Declare una condición especificando una instrucción DECLARE CONDITION dentro de la instrucción compuesta en la que desea hacer referencia a ella. Puede hacer referencia a una condición en la declaración de un controlador de condiciones, una declaración SIGNAL o una declaración RESIGNAL.

Restricción: Si existen varias condiciones con ese nombre dentro del mismo ámbito, no se puede hacer referencia explícita a una condición que no sea la más local en su ámbito. Db2 usa la condición en la sentencia compuesta más interna.

2. Declare un controlador de condición especificando una sentencia DECLARE HANDLER dentro de la sentencia compuesta a la que desea que se aplique el controlador de condición. Dentro de la declaración del gestor de condiciones, puede especificar una condición definida previamente.

Restricción: Los controladores de condiciones que se declaran en la misma sentencia compuesta no pueden controlar condiciones encontradas en sí mismos o en otros.

ejemplos

Ejemplo

En el siguiente ejemplo, una condición con el nombre ABC se declara dos veces, y una condición llamada XYZ se declara una vez.

```
CREATE PROCEDURE...
    DECLARE ABC CONDITION...

    DECLARE XYZ CONDITION...
    BEGIN
        DECLARE ABC CONDITION...
        SIGNAL ABC; 1
    END;

    SIGNAL ABC; 2
```

Las siguientes notas se refieren al ejemplo anterior:

1. ABC se refiere a la condición que se declara en el bloque más interno. Si esta declaración se cambiara a SEÑAL XYZ, XYZ se referiría a la condición XYZ que se declara en el bloque más externo.
2. ABC se refiere a la condición que se declara en el bloque más externo.

Ejemplo

El siguiente ejemplo contiene varias declaraciones de una condición con el nombre FOO y una sola declaración de una condición con el nombre GORP.

```
CREATE PROCEDURE MYTEST (INOUT A CHAR(1), INOUT B CHAR(1))
L1: BEGIN
    DECLARE GORP CONDITION
    FOR SQLSTATE '33333'; -- defines a condition with the name GORP for SQLSTATE 33333

    DECLARE EXIT HANDLER FOR GORP --defines a condition handler for SQLSTATE 33333
L2: BEGIN
    DECLARE FOO CONDITION
    FOR SQLSTATE '12345'; --defines a condition with the name FOO for SQLSTATE 12345
    DECLARE CONTINUE HANDLER FOR FOO --defines a condition handler for SQLSTATE 12345
    L3: BEGIN
        SET A = 'A';
        ...more statements...
    END L3;
    SET B = 'B';

    IF...
        SIGNAL FOO; --raises SQLSTATE 12345
    ELSEIF
        SIGNAL GORP; --raises SQLSTATE 33333
    END IF;

    END L2;

L4: BEGIN
    DECLARE FOO CONDITION
    FOR SQLSTATE '54321' --defines a condition with the name FOO for SQLSTATE 54321
    DECLARE EXIT HANDLER FOR FOO...; --defines a condition handler for SQLSTATE 54321
```

```

SIGNAL FOO SET MESSAGE_TEXT = '...'; --raises SQLSTATE 54321

L5: BEGIN
    DECLARE FOO CONDITION
        FOR SQLSTATE '99999'; --defines a condition with the name FOO for SQLSTATE 99999
        ...more statements...
    END L5;

END L4;

--At this point, the procedure cannot reference FOO, because this condition is not defined
--in this outer scope

END L1

```

Ejemplo

En el siguiente ejemplo, la sentencia compuesta con la etiqueta OUTER contiene otras dos sentencias compuestas: "INNER1A" y "INNER1B". La declaración de compuesto INNER1A contiene otra declaración de compuesto, que tiene la etiqueta INNER1A2, y la declaración para un controlador de condiciones HINNER1A. El cuerpo del controlador de condiciones HINNER1A contiene otra instrucción compuesta, que define otro controlador de condiciones, HINNER1A_HANDLER.

```

OUTER:
BEGIN
    -- Handler for OUTER
    DECLARE ... HANDLER -- HOUTER
        BEGIN
            :
        END; -- End of handler
        :
        :

-- Level 1 - first compound statement
INNER1A:
BEGIN
    -- Handler for INNER1A
    DECLARE ... HANDLER -- HINNER1A
        BEGIN
            -- Handler for handler HINNER1A
            DECLARE...HANDLER --HINNER1A_HANDLER
                BEGIN
                    :
                END; -- End of handler
                :
                :
                -- stmt that gets condition
                :
                : -- more statements in handler
            END; -- End of HINNER1A handler<-----.

INNER1A2:
BEGIN
    -- Handler for INNER1A2
    DECLARE ... HANDLER.... HINNER1A2
        BEGIN;
        :
        END; -- End of handler
        :
        :
        -- statement that gets condition
        :
        : -- statement after statement
        -- that encountered condition
    END INNER1A2;
    :
    : -- statements in INNER1A
END INNER1A; <-----.

-- Level 1 - second compound statement
INNER1B:
BEGIN
    -- Handler for handler INNER1B
    DECLARE ...HANDLER -- HINNER1B
        BEGIN
            -- Handler for HINNER1B --
            DECLARE ...HANDLER --HINNER1B_HANDLER
                BEGIN
                    :
                END; -- End of handler
                :
                : -- statements in handler

```

The diagram illustrates the nesting of code blocks using dashed boxes.
 - The outermost block is labeled '1' at the bottom right.
 - Inside it, the 'INNER1A' block is labeled '2' at the bottom right.
 - Within 'INNER1A', the 'INNER1A2' block is also labeled '2' at the bottom right.
 - The 'HINNER1A_HANDLER' block within 'INNER1A2' is labeled '1' at the bottom right.
 - The 'HINNER1B_HANDLER' block within 'INNER1B' is labeled '1' at the bottom right.

```

        END; -- End of HINNER1B handler<-----.
:
: -- statements in INNER1B
END INNER1B;                                <-----
:
: -- statements in OUTER
END OUTER;                                     <=====

```

Las siguientes notas se aplican al ejemplo anterior:

- Si se produce una excepción, advertencia o condición NOT FOUND dentro de la sentencia compuesta " INNER1A2 ", se activa el controlador más apropiado dentro de esa sentencia compuesta para manejar la condición. A continuación, se produce una de las siguientes acciones en función del tipo de controlador de condiciones:

- Si el controlador de condición (HINNER1A2) es un controlador de salida, el control vuelve al final de la sentencia compuesta que contenía el controlador de condición.
- Si el controlador de condición (HINNER1A2) es un controlador continuo, el procesamiento continúa con la sentencia después de la sentencia que encontró la condición.

Si no existe ningún controlador apropiado en la declaración compuesta INNER1A2, Db2 considera los siguientes controladores en el orden especificado:

- El controlador más apropiado dentro de la sentencia compuesta " INNER1A ".
- El controlador más apropiado dentro de la sentencia OUTER compound.

Si no existe un controlador apropiado en la sentencia OUTER compound, la condición es una condición no controlada. Si la condición es una condición de excepción, el procedimiento finaliza y devuelve una condición no gestionada a la aplicación que la invoca. Si la condición es una advertencia o una condición NOT FOUND, el procedimiento devuelve la condición de advertencia no gestionada a la aplicación que la invoca.

- Si se produce una excepción, advertencia o condición NOT FOUND dentro del cuerpo del controlador de condiciones HINNER1A y el controlador de condiciones HINNER1A_HANDLER es el más apropiado para la excepción, se activa ese controlador. De lo contrario, el control más apropiado dentro de la sentencia compuesta OUTER controla la condición. Si no existe un controlador apropiado dentro de la sentencia OUTER compound, la condición se trata como una condición no controlada.

Ejemplo

En el siguiente ejemplo, cuando *statement2* se produce una condición NOT FOUND, se activa el controlador de condiciones adecuado para gestionar la condición. Cuando el controlador de condición finaliza, la sentencia compuesta que contiene ese controlador de condición termina, porque el controlador de condición es un controlador EXIT. El procesamiento continúa entonces con *statement4*.

```

BEGIN
  DECLARE EXIT HANDLER FOR NOT FOUND
    SET OUT_OF_DATA_FLAG = ON;
  statement1...
  statement2... --assume that this statement results in a NOT FOUND condition
  statement3...
END;

statement4
...

```

Ejemplo

En el siguiente ejemplo, Db2 comprueba SQLSTATE 22H11 solo para las sentencias dentro de la sentencia compuesta INNER. Db2 comprueba SQLEXCEPTION para todas las sentencias en los bloques EXTERIOR e INTERIOR.

```

OUTER: BEGIN
  DECLARE var1 INT;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
    RETURN -3;

  INNER: BEGIN

```

```

        DECLARE EXIT HANDLER FOR SQLSTATE '22H11'
            RETURN -1;
        DECLARE C1 CURSOR FOR SELECT col1 FROM table1;
        OPEN C1;
        CLOSE C1;
        :
        : -- more statements
    END INNER;
    :
    : -- more statements

```

Ejemplo

En el siguiente ejemplo, Db2 comprueba SQLSTATE 42704 solo para sentencias dentro de la sentencia compuesta A.

```

CREATE PROCEDURE EXIT_TEST ()
LANGUAGE SQL
BEGIN
    DECLARE OUT_BUFFER VARCHAR(80);
    DECLARE NO_TABLE CONDITION FOR SQLSTATE '42704';

    A: BEGIN
        DECLARE EXIT HANDLER FOR NO_TABLE
        BEGIN
            SET OUT_BUFFER = 'Table does not exist';
        END;

        -- Drop potentially nonexistent table:
        DROP TABLE JAVELIN;
    
```

1
3
4
2

```

        B: SET OUT_BUFFER = 'Table dropped successfully';
    END;
    -- Copy OUT_BUFFER to some message table:
    C: INSERT INTO MESSAGES VALUES (OUT_BUFFER);

```

5

Las siguientes notas describen un posible flujo para el ejemplo anterior:

1. Una sentencia compuesta anidada con etiqueta A limita el alcance del controlador de salida NO_TABLE a las sentencias que se especifican en la sentencia compuesta A.
2. Si la tabla JAVELIN no existe, la sentencia DROP activa la condición NO_TABLE.
3. Se activa el controlador de salida para NO_TABLE.
4. La variable OUT_BUFFER se establece en la cadena «La tabla no existe»
5. La ejecución continúa con la sentencia INSERT. No se procesan más extractos en el extracto compuesto A.

Ejemplo

El siguiente ejemplo ilustra el alcance de los diferentes controladores de condiciones.

```

CREATE PROCEDURE ERROR_HANDLERS(IN PARAM INTEGER)
LANGUAGE SQL
OUTER: BEGIN
    DECLARE I INTEGER;
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';

    DECLARE EXIT HANDLER FOR
        SQLSTATE VALUE '38H02',
        SQLSTATE VALUE '38H04',
        SQLSTATE VALUE '38H14',
        SQLSTATE VALUE '38H06'
    OUTER_HANDLER: BEGIN
        DECLARE TEXT VARCHAR(70);
        SET TEXT = SQLSTATE || 
            ' RECEIVED AND MANAGED BY OUTER ERROR HANDLER' ;
        RESIGNAL SQLSTATE VALUE '38HE0'
            SET MESSAGE_TEXT = TEXT;
    END OUTER_HANDLER;

    INNER: BEGIN
        DECLARE EXIT HANDLER FOR SQLSTATE VALUE '38H03'
        RESIGNAL SQLSTATE VALUE '38HI3'
            SET MESSAGE_TEXT = '38H03 MANAGED BY INNER ERROR HANDLER';
    END INNER;

```

```

DECLARE EXIT HANDLER FOR SQLSTATE VALUE '38H04'
    RESIGNAL SQLSTATE VALUE '38HI4'
        SET MESSAGE_TEXT = '38H04 MANAGED BY INNER ERROR HANDLER';

DECLARE EXIT HANDLER FOR SQLSTATE VALUE '38H05'
    RESIGNAL SQLSTATE VALUE '38HI5'
        SET MESSAGE_TEXT = '38H05 MANAGED BY INNER ERROR HANDLER';

CASE PARAM
    WHEN 1 THEN                                -- (1)
        SIGNAL SQLSTATE VALUE '38H01'
        SET MESSAGE_TEXT =
            'EXAMPLE 1: ERROR SIGNALLED FROM INNER COMPOUND STMT';

    WHEN 2 THEN                                -- (2)
        SIGNAL SQLSTATE VALUE '38H02'
        SET MESSAGE_TEXT =
            'EXAMPLE 2: ERROR SIGNALLED FROM INNER COMPOUND STMT';

    WHEN 3 THEN                                -- (3)
        SIGNAL SQLSTATE VALUE '38H03'
        SET MESSAGE_TEXT =
            'EXAMPLE 3: ERROR SIGNALLED FROM INNER COMPOUND STMT';

    WHEN 4 THEN                                -- (4)
        SIGNAL SQLSTATE VALUE '38H04'
        SET MESSAGE_TEXT =
            'EXAMPLE 4: ERROR SIGNALLED FROM INNER COMPOUND STMT';

    ELSE
        SET I = 1; /*Do not do anything */
    END CASE;
END INNER;

CASE PARAM
    WHEN 5 THEN                                -- (5)
        SIGNAL SQLSTATE VALUE '38H05'
        SET MESSAGE_TEXT =
            'EXAMPLE 5: ERROR SIGNALLED FROM OUTER COMPOUND STMT';

    WHEN 6 THEN                                -- (6)
        SIGNAL SQLSTATE VALUE '38H06'
        SET MESSAGE_TEXT =
            'EXAMPLE 6: ERROR SIGNALLED FROM OUTER COMPOUND STMT';

    ELSE
        SET I = 1; /*Do not do anything */
    END CASE;
END OUTER;

```

La siguiente tabla resume el comportamiento del ejemplo anterior:

Valor de entrada para PARM	Comportamiento esperado
1	Se señala SQLSTATE 38H01 desde la sentencia compuesta INNER. Debido a que no existe un controlador apropiado, el procedimiento finaliza y devuelve la condición de excepción no controlada, 38H01 con SQLCODE -438, a la aplicación de llamada.
2	Se señala SQLSTATE 38H02 desde la sentencia compuesta INNER. Se activa el controlador de condición en la sentencia compuesta OUTER. Se emite una declaración RESIGNAL, con SQLSTATE 38HE0, desde el cuerpo del controlador de condiciones. Esta excepción hace que el control vuelva al final de la sentencia compuesta OUTER con la condición de excepción 38HE0 y SQLCODE -438. El procedimiento finaliza y devuelve la condición no gestionada a la aplicación de llamada.
3	Se señala SQLSTATE 38H03 desde la sentencia compuesta INNER. Se activa un controlador de condiciones dentro de la sentencia compuesta INNER. Se emite una declaración RESIGNAL, con SQLSTATE 38HI3, desde el cuerpo del controlador de condiciones. Debido a que no existe un controlador apropiado, el procedimiento finaliza y devuelve la condición de excepción no controlada, 38HI3 con SQLCODE -438, a la aplicación de llamada.

Valor de entrada para PARM	Comportamiento esperado
4	Se señala SQLSTATE 38H04 desde la sentencia compuesta INNER. Se activa un controlador de condiciones dentro de la sentencia compuesta INNER. Se emite una declaración RESIGNAL, con SQLSTATE 38HI4, desde el cuerpo del controlador de condiciones. Se activa un controlador de condiciones en la sentencia compuesta OUTER. Se emite una declaración RESIGNAL, con SQLSTATE 38HE0, desde el cuerpo del controlador de condiciones. Esta excepción hace que el control vuelva al final de la sentencia compuesta OUTER con la condición de excepción 38HE0 y SQLCODE -438. El procedimiento finaliza y devuelve la condición no gestionada a la aplicación de llamada.
5	Se señala SQLSTATE '38H05' desde la sentencia compuesta OUTER. Debido a que no existe un controlador apropiado, el procedimiento finaliza y devuelve la condición de excepción no controlada, 38H05 con SQLCODE -438, a la aplicación de llamada.
6	Se señala SQLSTATE 38H06 desde la sentencia OUTER compound. Se activa un controlador de condiciones en la sentencia compuesta OUTER. Se emite una declaración RESIGNAL, con SQLSTATE 38HE0, desde el cuerpo del controlador de condiciones. Esta excepción hace que el control vuelva al final de la sentencia compuesta OUTER con la condición de excepción 38HE0 y SQLCODE -438. El procedimiento finaliza y devuelve la condición no gestionada a la aplicación de llamada.
7	La cláusula ELSE de la sentencia "CASE" ejecuta y procesa la sentencia "SET". Se devuelve un código de finalización correcta a la aplicación de llamada.

Ejemplo

En el siguiente ejemplo de procedimiento SQL, el controlador de condiciones para exception1 no está dentro del ámbito del controlador de condiciones para exception0. Si se produce la condición de excepción exception1 en el cuerpo del controlador de condiciones para exception0, no existe ningún controlador apropiado y el procedimiento finaliza con una excepción no controlada.

```

CREATE PROCEDURE divide ( . . . . . )
LANGUAGE SQL CONTAINS SQL
BEGIN
    DECLARE dn_too_long CHAR(5) DEFAULT 'abcde';

    -- Declare condition names -----
    DECLARE exception0 CONDITION FOR SQLSTATE '22001';
    DECLARE exception1 CONDITION FOR SQLSTATE 'xxxxx';

    -- Declare cursors -----
    DECLARE cursor1 CURSOR WITH RETURN FOR
        SELECT * FROM dept;

    -- Declare handlers -----
    DECLARE CONTINUE HANDLER FOR exception0
    BEGIN
        some SQL statement that causes an error 'xxxxx'
    END

    DECLARE CONTINUE HANDLER FOR exception1
    BEGIN
        ...
    END

    -- Mainline of procedure -----
    INSERT INTO DEPT (DEPTNO) VALUES (dn_too_long);
        -- Assume that this statement results in SQLSTATE '22001'

    OPEN CURSOR1;
END

```

Recuperar información de diagnóstico utilizando GET DIAGNOSTICS en un controlador

Los controladores especifican la acción que realiza un procedimiento SQL cuando se produce un error o una condición en particular. En algunos casos, es posible que desee recuperar información de diagnóstico adicional sobre el error o la condición de advertencia.

Acerca de esta tarea

Procedimiento

Puede incluir una instrucción GET DIAGNOSTICS en un controlador para recuperar información de error o advertencia.

Si incluye GET DIAGNOSTICS, debe ser la primera declaración que se especifique en el controlador.

Ejemplo: Uso de GET DIAGNOSTICS para recuperar el texto de un mensaje

Supongamos que crea un procedimiento SQL, llamado "divide1", que calcula el resultado de la división de dos números enteros. Incluye GET DIAGNOSTICS para devolver el texto del mensaje de error de división como parámetro de salida:

```
CREATE PROCEDURE divide1
  (IN numerator INTEGER, IN denominator INTEGER,
   OUT divide_result INTEGER, OUT divide_error VARCHAR(1000))
LANGUAGE SQL
BEGIN
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    GET DIAGNOSTICS CONDITION 1 divide_error = MESSAGE_TEXT;
  SET divide_result = numerator / denominator;
END
```

Ignorar una condición en un procedimiento SQL

Puede especificar que desea ignorar errores o advertencias dentro de un ámbito particular de sentencias en un procedimiento SQL. Sin embargo, hágalo con precaución.

Procedimiento

Declare un controlador de condición que contenga una sentencia compuesta vacía.

Ejemplo

El siguiente ejemplo muestra un controlador de condición que se declara como una forma de ignorar una condición. Supongamos que su procedimiento SQL inserta filas en una tabla que tiene una columna única. Si el valor que se va a insertar para esa columna ya existe en la tabla, la fila no se inserta. Sin embargo, en este caso, no desea que Db2 notifique la aplicación sobre esta condición, que se indica mediante SQLSTATE 23505.

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '23505'
  BEGIN  -- ignore error for duplicate value
  END;
```

Conceptos relacionados

Manejadores en un procedimiento SQL

Si se produce un error cuando se ejecuta el procedimiento SQL, el procedimiento finaliza a menos que incluya sentencias para indicar al procedimiento que realice alguna otra acción. Estas sentencias se denominan manejadores.

Referencia relacionada

Valores de SQLSTATE y códigos de error comunes (Db2 Codes)

Generación de una condición dentro de un procedimiento SQL utilizando las sentencias SIGNAL o RESIGNAL

Dentro de un procedimiento SQL, puede forzar una condición particular para que se produzca con un texto de mensaje y SQLSTATE específicos.

Acerca de esta tarea

Puede utilizar una instrucción SIGNAL o RESIGNAL para plantear una condición con un SQLSTATE y un texto de mensaje específicos dentro de un procedimiento SQL. Las sentencias SIGNAL y RESIGNAL difieren en los siguientes aspectos:

- Puede utilizar la instrucción SIGNAL en cualquier lugar de un procedimiento SQL. Debe especificar el valor SQLSTATE. Además, puede utilizar la instrucción SIGNAL en un cuerpo de activación. Para obtener información sobre el uso de la instrucción SIGNAL en un disparador, consulte [“Creación de un desencadenante”](#) en la página 160.
- Puede utilizar la sentencia RESIGNAL solo dentro de un controlador de un procedimiento SQL. Si no especifica el valor SQLSTATE, Db2 utiliza el mismo valor SQLSTATE que activó el controlador.

Puede utilizar cualquier valor SQLSTATE válido en una instrucción SIGNAL o RESIGNAL, excepto una clase SQLSTATE con «00» como los dos primeros caracteres.

La siguiente tabla resume las diferencias entre emitir una declaración RESIGNAL o SIGNAL dentro del cuerpo de un controlador de condiciones. Para cada fila de la tabla, suponga que el área de diagnóstico contiene la siguiente información cuando se emite la instrucción RESIGNAL o SIGNAL:

```
RETURNED_SQLSTATE      xxxx
MESSAGE_TEXT 'this is my message'
```

Tabla 46. Ejemplo de sentencias RESIGNAL y SIGNAL

¿Especificar una nueva condición?	¿Especificar texto del mensaje?	Ejemplo de declaración de RESIGNAL...	Ejemplo de declaración SIGNAL...	Resultado
Nee	Nee	RESIGNAL 1	No es posible	RETURNED_SQLSTATE xxxx MESSAGE_TEXT «Este es mi mensaje»
Sí	Nee	RESIGNAL '98765' 2	SIGNAL '98765'	RETURNED_SQLSTATE 98765 MESSAGE_TEXT 'APLICACIÓN CON ERROR CON TEXTO DE DIAGNÓSTICO: este es mi mensaje'
Nee	Sí	No es posible	No es posible	N/D
Sí	Sí	RESIGNAL '98765' SET MESSAGE_TEXT = 'xyz' 3	SIGNAL '98765' SET MESSAGE_TEXT = 'xyz' 3	RETURNED_SQLSTATE 98765 MESSAGE_TEXT 'APLICACIÓN ERROR CON TEXTO DE DIAGNÓSTICO: xyz'

Nota:

1. Esta declaración plantea la condición actual con el SQLSTATE, SQLCODE, texto del mensaje y tokens existentes.
2. Esta declaración plantea una nueva condición (SQLSTATE '98765'). El texto y los tokens del mensaje existentes se restablecen. El SQLCODE se establece en -438 para un error o en 438 para una advertencia.

3. Esta declaración genera una nueva condición (SQLSTATE '98765') con un nuevo texto de mensaje ('xyz'). El SQLCODE se establece en -438 para un error o en 438 para una advertencia.

Ejemplo de la sentencia SIGNAL en un procedimiento SQL

Puede utilizar la instrucción SIGNAL en cualquier lugar de un procedimiento SQL para plantear una condición concreta.

El siguiente ejemplo utiliza una tabla ORDERS y una tabla CUSTOMERS que se definen de la siguiente manera:

```
CREATE TABLE ORDERS
(ORDERNO      INTEGER NOT NULL,
 PARTNO       INTEGER NOT NULL,
 ORDER_DATE   DATE DEFAULT,
 CUSTNO       INTEGER NOT NULL,
 QUANTITY     SMALLINT NOT NULL,
 CONSTRAINT REF CUSTNO FOREIGN KEY (CUSTNO)
    REFERENCES CUSTOMERS (CUSTNO) ON DELETE RESTRICT,
 PRIMARY KEY (ORDERNO,PARTNO));

CREATE TABLE CUSTOMERS
(CUSTNO       INTEGER NOT NULL,
 CUSTNAME    VARCHAR(30),
 CUSTADDR    VARCHAR(80),
 PRIMARY KEY (CUSTNO));
```

Ejemplo: Uso de SIGNAL para configurar el texto del mensaje

Supongamos que tiene un procedimiento SQL para un sistema de pedidos que señala un error de aplicación cuando un número de cliente no es conocido por la aplicación. La tabla ORDERS tiene una clave externa a la tabla CUSTOMERS, que requiere que CUSTNO exista en la tabla CUSTOMERS antes de que se pueda insertar un pedido:

```
CREATE PROCEDURE submit_order
(IN ONUM INTEGER, IN PNUM INTEGER,
 IN CNUM INTEGER, IN QNUM INTEGER)
LANGUAGE SQL
MODIFIES SQL DATA
BEGIN
    DECLARE EXIT HANDLER FOR SQLSTATE VALUE '23503'
        SIGNAL SQLSTATE '75002'
        SET MESSAGE_TEXT = 'Customer number is not known';
    INSERT INTO ORDERS (ORDERNO, PARTNO, CUSTNO, QUANTITY)
        VALUES (ONUM, PNUM, CNUM, QNUM);
END
```

En este ejemplo, la instrucción SIGNAL está en el controlador. Sin embargo, puede utilizar la instrucción SIGNAL para invocar un controlador cuando se produzca una condición que dé lugar a un error.

Conceptos relacionados

[Ejemplo de la sentencia RESIGNAL en un descriptor](#)

Puede utilizar la sentencia RESIGNAL en un procedimiento SQL para asignar un valor distinto a la condición que ha activado el manejador. T

Ejemplo de la sentencia RESIGNAL en un descriptor

Puede utilizar la sentencia RESIGNAL en un procedimiento SQL para asignar un valor distinto a la condición que ha activado el manejador. T

Ejemplo: Uso de RESIGNAL para establecer un valor SQLSTATE

Supongamos que crea un procedimiento SQL, llamado divide2, que calcula el resultado de la división de dos números enteros. Incluye SIGNAL para invocar el controlador con una condición de desbordamiento

causada por un divisor cero, e incluye RESIGNAL para establecer un valor SQLSTATE diferente para esa condición de desbordamiento:

```
CREATE PROCEDURE divide2
  (IN numerator INTEGER, IN denominator INTEGER,
   OUT divide_result INTEGER)
LANGUAGE SQL
BEGIN
  DECLARE overflow CONDITION FOR SQLSTATE '22003';
  DECLARE CONTINUE HANDLER FOR overflow
    RESIGNAL SQLSTATE '22375';
  IF denominator = 0 THEN
    SIGNAL overflow;
  ELSE
    SET divide_result = numerator / denominator;
  END IF;
END
```

Ejemplo: RESIGNAL en una sentencia compuesta anidada

Si el siguiente procedimiento SQL se invoca con los valores de argumento 1, 0 y 0, el procedimiento devuelve un valor de 2 para RC y establece el parámetro oparm1 en 650.

```
CREATE PROCEDURE resig4
  (IN iparm1 INTEGER, INOUT oparm1 INTEGER, INOUT rc INTEGER)
LANGUAGE SQL
A1: BEGIN
  DECLARE c1 INT DEFAULT 1;
  DECLARE CONTINUE HANDLER FOR SQLSTATE VALUE '01ABX'
    BEGIN
      .... some other statements
      SET RC = 3; 6
    END;

  A2: SET oparm1 = 5; 1

  A3: BEGIN
    DECLARE c1 INT DEFAULT 1;
    DECLARE CONTINUE HANDLER
      FOR SQLSTATE VALUE '01ABC'
    BEGIN
      SET RC = 1;
      RESIGNAL SQLSTATE VALUE '01ABX' 4
        SET MESSAGE_TEXT = 'get out of here'; 5
      SET RC = 2; 7
    END;

    A7: SET oparm1 = oparm1 + 110; 2
    SIGNAL SQLSTATE VALUE '01ABC' 3
      SET MESSAGE_TEXT = 'yikes';
      SET oparm1 = oparm1 + 215; 8
    END;
    SET oparm1 = oparm1 + 320; 9

  END
END
```

Las siguientes notas se refieren al ejemplo anterior:

1. oparm1 está inicialmente establecido en 5.
2. oparm1 se incrementa en 110. El valor de oparm1 es ahora 115.
3. La sentencia SIGNAL hace que se active el controlador de condiciones que está contenido en la sentencia compuesta "A3".
4. En este controlador de condición, RC se establece en 1.
5. La sentencia RESIGNAL cambia el SQLSTATE a 01ABX. Este valor hace que se active el controlador de continuación en la instrucción compuesta "A1".
6. RC se establece en 3 en este controlador de condición. Dado que este controlador de condición es un controlador continuo, cuando finaliza la acción del controlador, el control vuelve a la instrucción SET después de la instrucción RESIGNAL.

7. RC se establece en 2 en este controlador de condición. Debido a que este controlador de condiciones es un controlador continuo, el control vuelve a la instrucción SET que sigue a la instrucción SIGNAL que provocó la activación del controlador de condiciones.
8. oparm1 se incrementa en 215. El valor de oparm es ahora 330.
9. oparm1 se incrementa en 320. El valor de oparm es ahora 650.

Cómo afectan las sentencias SIGNAL y RESIGNAL al área de diagnóstico

Cuando emite una declaración de SEÑAL, se crea una nueva área de diagnóstico lógico. Cuando emite una declaración de RENUNCIA, se actualiza el área de diagnóstico actual.

Cuando emite una declaración de SEÑAL, se crea lógicamente una nueva área de diagnóstico. En esa área de diagnóstico, RETURNED_SQLSTATE se establece en el SQLSTATE o nombre de condición especificado. Si especificó el texto del mensaje como parte de la declaración SIGNAL, MESSAGE_TEXT en el área de diagnóstico también se establece en el valor especificado.

Cuando emite una declaración RESIGNAL con un valor SQLSTATE, nombre de condición o texto de mensaje, el área de diagnóstico actual se actualiza con la información especificada.

Hacer copias de un paquete para un procedimiento SQL nativo

Cuando se crea un procedimiento SQL nativo, se vincula implícitamente un paquete con las opciones especificadas en la instrucción CREATE PROCEDURE. Si el procedimiento SQL nativo realiza ciertas acciones, debe hacer copias explícitas de ese paquete.

Acerca de esta tarea

Si el procedimiento SQL nativo realiza una o más de las siguientes acciones, debe crear copias del paquete para ese procedimiento:

- Utiliza una instrucción CONNECT para conectarse a un servidor de base de datos.
- Se refiere a una tabla con un nombre de tres partes que incluye una ubicación distinta del servidor actual o se refiere a un alias que se resuelve en dicho nombre.
- Establece el registro especial CURRENT PACKAGESET para controlar qué paquete se invoca para esa versión del procedimiento.
- Establece el registro especial CURRENT PACKAGE PATH para controlar qué paquete se invoca para esa versión del procedimiento.

El paquete de una versión de un procedimiento tiene el siguiente nombre: *location.collection-id.package-id.version-id* donde estas variables tienen los siguientes valores:

ubicación

Valor del registro especial del SERVIDOR ACTUAL

id-colección

Calificador de esquema del procedimiento

id-paquete

Nombre de procedimiento

version-id

Identificador de versión

Para hacer copias de un paquete para un procedimiento SQL nativo, especifique el comando BIND PACKAGE con la opción COPY. Para copias creadas en el servidor actual, especifique un calificador de esquema diferente, que es el ID de colección. Para la primera copia que se crea en un servidor remoto, puede especificar el mismo calificador de esquema. Para otras copias en ese servidor remoto, especifique un calificador de esquema diferente.

Si más adelante cambia el procedimiento SQL nativo, es posible que tenga que volver a vincular explícitamente cualquier copia local o remota del paquete que exista para esa versión del procedimiento.

ejemplos

Ejemplo

Debido a que el siguiente procedimiento nativo de SQL contiene una instrucción CONNECT, debe crear una copia del paquete en el servidor de destino, que en este caso se encuentra en la ubicación SAN_JOSE. El comando BIND posterior crea una copia del paquete para la versión ABC del procedimiento TEST.MYPROC. Este paquete se crea en la ubicación SAN_JOSE y lo utiliza Db2 cuando se ejecuta este procedimiento.

```
CREATE PROCEDURE TEST.MYPROC VERSION ABC LANGUAGE SQL ...
BEGIN
  ...
  CONNECT TO SAN_JOSE
  ...
END

BIND PACKAGE (SAN_JOSE.TEST) COPY(TEST.MYPROC) COPYVER(ABC) ACTION(ADD)
```

Ejemplo

El siguiente procedimiento nativo de SQL establece el registro especial CURRENT PACKAGESET para garantizar que Db2 utilice el paquete con el ID de colección COLL2 para esta versión del procedimiento. Por consiguiente, debe crear dicho paquete. El comando BIND posterior crea este paquete con el ID de colección COLL2. Este paquete es una copia del paquete para la versión ABC del procedimiento TEST.MYPROC. Db2 utiliza este paquete para procesar las sentencias SQL en este procedimiento.

```
CREATE PROCEDURE TEST.MYPROC VERSION ABC LANGUAGE SQL ...
BEGIN
  ...
  SET CURRENT PACKAGESET = 'COLL2'
  ...
END

BIND PACKAGE(COLL2) COPY(TEST.MYPROC) COPYVER(ABC)
ACTION(ADD) QUALIFIER(XYZ)
```

Tareas relacionadas

Regeneración de una versión existente de un procedimiento de SQL nativo

Cuando aplica el mantenimiento de Db2 que cambia la forma en la que se generan los procedimientos de SQL nativo, debe volver a generar cualquier procedimiento afectado. Cuando vuelve a generar una versión de un procedimiento de SQL nativo, Db2 vuelve a enlazar el paquete asociado para esa versión del procedimiento.

Sustitución de copias de un paquete para una versión de un procedimiento de SQL nativo

Cuando cambia una versión de un procedimiento de SQL nativo y la sentencia ALTER PROCEDURE REPLACE contiene ciertas opciones, debe sustituir cualquier copia local o remota de paquetes que existen para esa versión del procedimiento.

Referencia relacionada

[Declaración ALTER PROCEDURE \(SQL - procedimiento nativo\) \(Db2 SQL\)](#)

Sustitución de copias de un paquete para una versión de un procedimiento de SQL nativo

Cuando cambia una versión de un procedimiento de SQL nativo y la sentencia ALTER PROCEDURE REPLACE contiene ciertas opciones, debe sustituir cualquier copia local o remota de paquetes que existen para esa versión del procedimiento.

Acerca de esta tarea

Si especifica cualquiera de las siguientes opciones de ALTER PROCEDURE, debe reemplazar las copias del paquete:

- SUSTITUIR VERSIÓN
- REGENERATE
- DISABLE DEBUG MODE

- QUALIFIER
- Propietario del paquete
- APLAZAR PREPARAR
- NODEFER PREPARE
- datos actuales
- DEGREE
- DYNAMICRULES
- Esquema de codificación de la aplicación
- CON EXPLICACIÓN
- SIN EXPLICAR
- POR ESCRITO INMEDIATO
- SIN ESCRITURA INMEDIATA
- ISOLATION LEVEL
- CON MANTENER DINÁMICO
- SIN MANTENER DINÁMICO
- OPTHINT
- Vía de acceso de SQL
- LIBERACIÓN EN COMPROMISO
- LIBERACIÓN EN DESASIGNACIÓN
- REOPT
- VALIDAR EJECUCIÓN
- VALIDAR VINCULACIÓN
- ROUNDING
- DATE FORMAT
- DECIMAL
- PARA ACTUALIZAR LA CLÁUSULA OPCIONAL
- PARA ACTUALIZAR SE REQUIERE CLÁUSULA
- Formato de hora

Para reemplazar copias de un paquete por una versión de un procedimiento SQL nativo, especifique el comando BIND COPY ACTION(REPLACE) con el nombre de paquete y el ID de versión adecuados.

Creación de nuevas versiones de procedimientos SQL nativos

Una versión nueva de un procedimiento de SQL nativo puede tener diferentes nombres de parámetros, opciones de procedimiento o cuerpo del procedimiento.

Acerca de esta tarea

Todas las versiones de un procedimiento deben tener la misma firma de procedimiento. Por lo tanto, cada versión del procedimiento debe tener los mismos de los siguientes elementos:

- Nombre de esquema
- Nombre de procedimiento
- Número de parámetros
- Tipos de datos para los parámetros correspondientes

Cuando una sola versión de un procedimiento se define como autónoma, todas las versiones deben definirse como autónomas.

Importante: No cree versiones adicionales de los procedimientos que se suministran con Db2 especificando la palabra clave VERSION. Solo se admiten las versiones que se suministran con Db2 . Las versiones adicionales de tales rutinas provocan que la instalación y configuración de las rutinas suministradas fallen.

Procedimiento

Para crear una nueva versión de un procedimiento, emita uno de los siguientes:

- FL 507 La instrucción CREATE PROCEDURE con los siguientes elementos:
 - La cláusula O REEMPLAZAR.
 - La cláusula VERSION con un nuevo identificador de versión.
- La instrucción ALTER PROCEDURE con los siguientes elementos:
 - La cláusula ADD VERSION con un nombre para la nueva versión.

Para cualquiera de las declaraciones, debe incluir lo siguiente:

- El nombre del procedimiento nativo de SQL para el que desea crear una nueva versión.
- La lista de parámetros del procedimiento que desea cambiar. Para ALTER PROCEDURE ADD VERSION, esta lista de parámetros debe ser la misma que la del procedimiento original.
- Cualquier opción de procedimiento. Estas opciones pueden ser diferentes de las opciones para otras versiones de este procedimiento. Si no especifica un valor para una opción concreta, se utilizará el valor predeterminado, independientemente del valor que utilice la versión activa actual de este procedimiento.
- Un cuerpo de procedimiento. Este cuerpo puede ser diferente al cuerpo del procedimiento para otras versiones de este procedimiento.

ejemplos

Ejemplo 1

Por ejemplo, la siguiente instrucción CREATE PROCEDURE define un nuevo procedimiento nativo de SQL llamado UPDATE_BALANCE. La versión del procedimiento es V1, y es la versión activa.

```
CREATE PROCEDURE
UPDATE_BALANCE
(IN CUSTOMER_NO INTEGER,
IN AMOUNT DECIMAL(9,2))
VERSION V1
LANGUAGE SQL
READS SQL DATA
BEGIN
DECLARE CUSTOMER_NAME CHAR(20);
SELECT CUSTNAME
INTO CUSTOMER_NAME
FROM ACCOUNTS
WHERE CUSTNO = CUSTOMER_NO;
END
```

Ejemplo 2

La siguiente instrucción ALTER PROCEDURE crea una nueva versión del procedimiento UPDATE_BALANCE. El nombre de la nueva versión es V2. Esta nueva versión tiene un cuerpo de procedimiento diferente.

```
ALTER PROCEDURE
UPDATE_BALANCE
ADD VERSION V2
(IN CUSTOMER_NO INTEGER,
IN AMOUNT DECIMAL (9,2) )
MODIFIES SQL DATA
BEGIN
UPDATE ACCOUNTS
SET BAL = BAL + AMOUNT
```

```
WHERE CUSTNO = CUSTOMER_NO;
END
```

Ejemplo 3:

FL 507

La siguiente instrucción CREATE PROCEDURE con la cláusula OR REPLACE crea una nueva versión del procedimiento UPDATE_BALANCE, suponiendo que la versión V3 no existe ya (si V3 ya existe, esta instrucción reemplazaría la definición existente). Esta versión cambia el cuerpo del procedimiento de la misma manera que en el Ejemplo 2:

```
CREATE OR REPLACE PROCEDURE
UPDATE_BALANCE
(IN CUSTOMER_NO INTEGER,
IN AMOUNT DECIMAL(9,2))
VERSION V3
LANGUAGE SQL
MODIFIES SQL DATA
BEGIN
UPDATE ACCOUNTS
SET BAL = BAL + AMOUNT
WHERE CUSTNO = CUSTOMER_NO;
END
```

Qué hacer a continuación

Después de crear una nueva versión, si desea que esa versión sea invocada por todas las llamadas posteriores a este procedimiento, debe hacer que esa versión sea la versión activa. Puede utilizar la cláusula ACTIVATE VERSION en una instrucción ALTER PROCEDURE o en una instrucción CREATE PROCEDURE con la cláusula OR REPLACE.

Referencia relacionada

[Declaración ALTER PROCEDURE \(SQL - procedimiento nativo\) \(Db2 SQL\)](#)

[Instrucción CREATE PROCEDURE \(SQL - procedimiento nativo\) \(Db2 SQL\)](#)

Múltiples versiones de procedimientos SQL nativos

Puede definir varias versiones de un procedimiento SQL nativo. Db2 mantiene esta información de versión para usted.

Una o más versiones de un procedimiento pueden existir en cualquier momento en el servidor actual, pero solo una versión de un procedimiento se considera la versión activa. Cuando se crea un procedimiento por primera vez, esa versión inicial se considera la versión activa del procedimiento.

El uso de varias versiones de un procedimiento SQL nativo tiene las siguientes ventajas:

- Puede mantener activa la versión existente de un procedimiento mientras crea otra versión. Cuando la otra versión esté lista, puede convertirla en la activa.
- Cuando activa otra versión de un procedimiento, no es necesario que cambie ninguna llamada existente a ese procedimiento.
- Puede volver fácilmente a una versión anterior de un procedimiento si la versión a la que ha cambiado no funciona según lo previsto.
- Puede eliminar una versión innecesaria de un procedimiento.

Una nueva versión de un procedimiento SQL nativo puede tener valores diferentes para los siguientes elementos:

- Nombres de parámetro
- Opciones de procedimiento (, excepto la opción AUTÓNOMA, que debe especificarse para todas las versiones o para ninguna)
- Cuerpo del procedimiento

Restricciones:

- Una nueva versión de un procedimiento SQL nativo no puede tener valores diferentes para los siguientes elementos:

- Número de parámetros
- Tipos de datos de parámetros
- Atributos de parámetros para datos de caracteres
- Parámetro CCSID
- Si un parámetro es un parámetro de entrada o de salida, según se define en las opciones IN, OUT e INOUT

Si necesita especificar valores diferentes para cualquiera de los elementos anteriores, cree un nuevo procedimiento nativo de SQL, en lugar de una nueva versión.

- Cuando se especifica la opción AUTÓNOMA para una versión de un procedimiento, debe especificarse para todas las versiones de ese procedimiento.

Despliegue de un procedimiento de SQL nativo en otro servidor Db2 for z/OS

Cuando despliegue un procedimiento de SQL nativo en otro servidor Db2 for z/OS, puede cambiar las opciones de enlace para que coincidan mejor con el entorno de despliegue. La lógica de procedimiento permanece igual.

Antes de empezar

Función en desuso: La opción de enlace DEPLOY está obsoleta. Para obtener los mejores resultados, implemente funciones SQL compiladas y procedimientos SQL nativos en múltiples entornos emitiendo las mismas sentencias CREATE o ALTER por separado en cada entorno de e Db2 .

Requisitos:

- El servidor remoto debe estar correctamente definido en la base de datos de comunicaciones del subsistema de Db2 , desde el cual se implementa el procedimiento SQL nativo.
- El subsistema de destino (Db2) debe estar funcionando a un nivel de PTF que sea compatible con el nivel de PTF del subsistema local (Db2).

Procedimiento

Emitir el comando BIND PACKAGE con las siguientes opciones:

DESPLEGAR

Especifique el nombre del procedimiento cuya lógica desea utilizar en el servidor de destino.

Consejo: Al especificar los parámetros para la opción DEPLOY, tenga en cuenta las siguientes reglas de nomenclatura para los procedimientos nativos de SQL:

- El ID de la colección es el mismo que el nombre del esquema en la instrucción CREATE PROCEDURE original.
- El ID del paquete es el mismo que el nombre del procedimiento en la instrucción CREATE PROCEDURE original.

COPYVER

Especifique la versión del procedimiento cuya lógica desea utilizar en el servidor de destino.

ACCIÓN (AÑADIR) o ACCIÓN (SUSTITUIR)

Especifique si desea que Db2 cree una nueva versión del procedimiento SQL nativo y su paquete asociado o que reemplace la versión especificada.

Opcionalmente, también puede especificar las opciones de enlace CALIFICADOR o PROPIETARIO si desea cambiarlas.

ejemplos

Implementar la misma versión de un procedimiento en otra ubicación

El siguiente comando BIND crea un procedimiento SQL nativo con el nombre PRODUCTION.MYPROC en la ubicación CHICAGO. Este procedimiento se crea a partir del procedimiento TEST.MYPROC

en el sitio actual. Los dos procedimientos SQL nativos tienen el mismo contenido y versión, ABC. Sin embargo, el paquete para el procedimiento CHICAGO.PRODUCTION.MYPROC tiene XYZ como calificador.

```
CREATE PROCEDURE TEST.MYPROC VERSION ABC LANGUAGE SQL ...
BEGIN
  ...
END

BIND PACKAGE(CHICAGO.PRODUCTION) DEPLOY(TEST.MYPROC) COPYVER(ABC)
ACTION(ADD) QUALIFIER(XYZ)
```

Sustitución de una versión de un procedimiento en otra ubicación

El siguiente comando BIND reemplaza la versión ABC del procedimiento PRODUCTION.MYPROC en la ubicación CHICAGO con la versión ABC del procedimiento TEST.MYPROC en el sitio actual.

```
BIND PACKAGE(CHICAGO.PRODUCTION) DEPLOY(TEST.MYPROC) COPYVER(ABC)
ACTION(REPLACE) REPLVER(ABC)
```

Conceptos relacionados

[Base de datos de comunicaciones para el servidor \(Managing Security\)](#)

Referencia relacionada

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2\)](#)

[BIND PACKAGE subcomando \(DSN\) \(Comandos de Db2\)](#)

Información relacionada

[Db2 para procedimientos almacenados de SQL Server \(z/OS Stored Procedures: Through the CALL and Beyond \(IBM Redbooks \)](#)

Eliminar una versión existente de un procedimiento SQL nativo

Puede eliminar una versión concreta de un procedimiento SQL nativo sin eliminar las demás versiones del procedimiento.

Antes de empezar

Antes de eliminar una versión existente de un procedimiento SQL nativo, asegúrese de que la versión no esté activa. Si la versión es la versión activa, designe una versión activa diferente antes de continuar.

Procedimiento

Emitir la sentencia ALTER PROCEDURE con la cláusula DROP VERSION y el nombre de la versión que desea eliminar. Si, por el contrario, desea eliminar todas las versiones del procedimiento, utilice la instrucción DROP.

ejemplos

Ejemplo de eliminación de una versión que no está activa

La siguiente declaración elimina la versión OLD_PRODUCTION del procedimiento P1.

```
ALTER PROCEDURE P1 DROP VERSION OLD_PRODUCTION
```

Ejemplo de eliminación de una versión activa

Asume que la versión OLD_PRODUCTION del procedimiento P1 es la versión activa. En el siguiente ejemplo, primero se cambia la versión activa a NUEVA_PRODUCCIÓN y, a continuación, se elimina la versión VIEJA_PRODUCCIÓN.

```
ALTER PROCEDURE P1 ACTIVATE VERSION NEW_PRODUCTION;
ALTER PROCEDURE P1 DROP VERSION OLD_PRODUCTION;
```

Tareas relacionadas

[Designación de la versión activa de un procedimiento de SQL nativo](#)

Cuando se invoca un procedimiento de SQL nativo, Db2 utiliza la versión que se designa como versión activa.

Regeneración de una versión existente de un procedimiento de SQL nativo

Cuando aplica el mantenimiento de Db2 que cambia la forma en la que se generan los procedimientos de SQL nativo, debe volver a generar cualquier procedimiento afectado. Cuando vuelve a generar una versión de un procedimiento de SQL nativo, Db2 vuelve a enlazar el paquete asociado para esa versión del procedimiento.

Acerca de esta tarea

El procedimiento ALTER REGENERATE es diferente al comando REBIND PACKAGE. Cuando especifique REBIND PACKAGE, Db2 vuelve a vincular solo las sentencias SQL que no son de control. Utilice este comando cuando considere que la reasignación mejorará la ruta de acceso. Cuando especifica ALTER PROCEDURE REGENERATE, Db2 vuelve a vincular las instrucciones de control SQL, así como las instrucciones que no son de control.

Procedimiento

Para regenerar una versión existente de un procedimiento SQL nativo:

1. Emitir la instrucción ALTER PROCEDURE con la cláusula REGENERATE y especificar la versión que se va a regenerar.
2. Si existen copias del paquete para la versión especificada del procedimiento en sitios remotos, reemplace esos paquetes. Emitir el comando BIND PACKAGE con la opción COPY y la ubicación adecuada para cada paquete remoto.
3. Si existen localmente copias del paquete para la versión especificada del procedimiento con diferentes nombres de esquema, reemplace esos paquetes. Emitir el comando BIND PACKAGE con la opción COPY y el esquema apropiado para cada paquete local.

Ejemplo

La siguiente instrucción ALTER PROCEDURE regenera la versión activa del procedimiento UPDATE_SALARY_1.

```
ALTER PROCEDURE UPDATE_SALARY_1  
REGENERATE ACTIVE VERSION
```

Cambio de una versión existente de un procedimiento SQL nativo

Puede cambiar una opción o el cuerpo del procedimiento para una versión determinada de un procedimiento SQL nativo. Si desea conservar una copia de ese procedimiento almacenado, considere la posibilidad de crear una nueva versión en lugar de cambiar la versión existente.

Procedimiento

Para cambiar una versión existente de un procedimiento SQL nativo, emita una de las siguientes declaraciones:

- [FL 507](#) con la cláusula OR REPLACE y la cláusula VERSION que identifica la versión que se va a reemplazar.
- La instrucción ALTER PROCEDURE con la cláusula REPLACE VERSION.

Cualquier opción que no especifique explícitamente hereda los valores predeterminados del sistema. Esta herencia se produce incluso si esas opciones se especificaron explícitamente para una versión anterior mediante una instrucción CREATE PROCEDURE, una instrucción ALTER PROCEDURE o un comando REBIND.

ejemplos

Ejemplo 1

La siguiente instrucción ALTER PROCEDURE actualiza la versión V2 del procedimiento UPDATE_BALANCE.

```
ALTER PROCEDURE
TEST.UPDATE_BALANCE
REPLACE VERSION V2
(IN CUSTOMER_NO INTEGER,
IN AMOUNT DECIMAL(9,2))
MODIFIES SQL DATA
ASUTIME LIMIT 100
BEGIN
UPDATE ACCOUNTS
SET BAL = BAL + AMOUNT
WHERE CUSTNO = CUSTOMER_NO
AND CUSTSTAT = 'A';
END
```

Ejemplo 2

FL 507

La siguiente instrucción CREATE PROCEDURE sustituirá la versión V2 del procedimiento UPDATE_BALANCE si ya existe la versión V2 o la creará si aún no se ha definido la versión V2 :

```
CREATE OR REPLACE PROCEDURE
TEST.UPDATE_BALANCE
(IN CUSTOMER_NO INTEGER,
IN AMOUNT DECIMAL(9,2))
VERSION V2
MODIFIES SQL DATA
ASUTIME LIMIT 100
BEGIN
UPDATE ACCOUNTS
SET BAL = BAL + AMOUNT
WHERE CUSTNO = CUSTOMER_NO
AND CUSTSTAT = 'A';
END
```

Tareas relacionadas

[Creación de nuevas versiones de procedimientos SQL nativos](#)

Una versión nueva de un procedimiento de SQL nativo puede tener diferentes nombres de parámetros, opciones de procedimiento o cuerpo del procedimiento.

Referencia relacionada

[REBIND PACKAGE subcomando \(DSN\) \(Comandos de Db2 \)](#)

[Declaración ALTER PROCEDURE \(SQL - procedimiento nativo\) \(Db2 SQL\)](#)

[Instrucción CREATE PROCEDURE \(SQL - procedimiento nativo\) \(Db2 SQL\)](#)

Creación de procedimientos almacenados externos

Un *procedimiento almacenado externo* es un procedimiento que se escribe en un lenguaje de host y que puede contener sentencias SQL. El código fuente de los procedimientos externos es distinto de la definición.

Antes de empezar

Antes de crear un procedimiento externo, [configure Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario durante la instalación](#) o [configure Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario durante la migración](#).

Acerca de esta tarea

Restricción: Estas instrucciones no se aplican a los procedimientos almacenados de Java. El proceso para crear [un procedimiento almacenado de Java](#) es diferente. El proceso de preparación varía en función de lo que contenga el procedimiento.

Procedimiento

Para crear un procedimiento almacenado externo:

1. Escribir el cuerpo del procedimiento almacenado externo en ensamblador, C, C++, COBOL, REXX o PL/I.

Asegúrese de que el cuerpo del procedimiento que escribe sigue las directrices para procedimientos almacenados externos que se describen en la siguiente información:

- “[Acceso a otros sitios en un procedimiento externo](#)” en la página 290
- “[Acceso de recursos no Db2 en el procedimiento almacenado](#)” en la página 291
- “[Escribir un procedimiento externo para acceder a IMS bases de datos](#)” en la página 292
- “[Escribir un procedimiento externo para devolver conjuntos de resultados a un cliente distribuido](#)” en la página 293
- “[Restricciones al invocar otros programas desde un procedimiento almacenado externo](#)” en la página 294
- “[Procedimientos almacenados externos como programas y subprogramas principales](#)” en la página 296
- “[Tipos de datos en procedimientos almacenados](#)” en la página 298
- “[Sentencias COMMIT y ROLLBACK en un procedimiento almacenado](#)” en la página 243

Restricciones:

- No incluya llamadas con función de datos adjuntos explícitas. Los procedimientos almacenados externos que se ejecutan en un espacio de direcciones establecido por WLM utilizan llamadas RRSAF (Resource Recovery Services attachment facility) de forma implícita. Si un procedimiento almacenado externo realiza una llamada explícita a la función de adjuntar, Db2 rechaza la llamada.
- No incluya las llamadas de servicio SRRCMIT o SRRBACK. Si un procedimiento almacenado externo invoca SRRCMIT o SRRBACK, Db2 pone la transacción en un estado en el que se requiere una operación de reversión y la instrucción CALL falla.

Para los procedimientos REXX, continúe con el paso “3” en la página 272.

2. Para los procedimientos almacenados en ensamblador, C, C++, COBOL o PL/I, prepare el procedimiento externo completando las siguientes tareas:

a) Precompilar, compilar y editar enlaces de la aplicación mediante una de las siguientes técnicas:

- El precompilador Db2 y las instrucciones JCL para compilar y editar el enlace del programa
- El coprocesador de instrucciones SQL

Recomendación: Compilar y editar el código como reentrante.

Enlace-edite la aplicación utilizando DSNRLI, el módulo de interfaz de idioma para el Resource Recovery Services attachment facility, o DSNULI, el módulo de interfaz de idioma universal. Debe especificar el parámetro AMODE(31) cuando edite el enlace de la aplicación con cualquiera de estos módulos. (No se admiten aplicaciones de 24 bits)

Si desea que el procedimiento almacenado sea reentrante, consulte “[Creación de un procedimiento almacenado externo como reentrant](#)” en la página 295

Si desea ejecutar su procedimiento como un z/OS -programa autorizado, también debe realizar las siguientes tareas al vincular y editar la aplicación:

- Indique que el módulo de carga puede utilizar servicios de sistema restringidos especificando el valor del parámetro AC=1.
- Coloque el módulo de carga para el procedimiento almacenado en una biblioteca autorizada por APF.

Puede compilar procedimientos almacenados COBOL con las opciones de compilador COBOL DYNAM o NODYNAM. Si utiliza DYNAM, asegúrese de que el módulo de interfaz de idioma de Db2 correcto se carga dinámicamente realizando una de las siguientes acciones:

- Especifique la opción de procesamiento SQL ATTACH(RRSAF).

- Copie el módulo DSNRLI en una biblioteca de carga que se concatene delante de las bibliotecas de Db2 . Utilice el nombre de miembro DSNHLI.
- b) Enlazar el DBRM en un paquete de configuración de red (Db2) emitiendo el comando BIND PACKAGE.

Si desea controlar el acceso a un paquete de procedimientos almacenados, especifique la opción de enlace ENABLE con el tipo de conexión del sistema de la aplicación de llamada.

Los procedimientos almacenados solo requieren un paquete. No es necesario vincular un plan para el procedimiento almacenado ni vincular el paquete de procedimientos almacenados al plan para la aplicación de llamada. Para los escenarios de acceso remoto, se necesita un paquete tanto en el sitio del solicitante como en el del servidor.

Para obtener más información sobre los paquetes de procedimientos almacenados, consulte “[Paquetes para procedimientos almacenados externos](#)” en la página 290.

El siguiente comando BIND PACKAGE de ejemplo vincula el DBRM EMPDTL1P a la colección DEVL7083.

```
BIND PACKAGE(DEVL7083) -
 MEMBER(EMPDTL1P) ACT(REP) ISO(UR) ENCODING(EBCDIC) -
 OWNER(DEVL7083) LIBRARY('SG247083.DEVL.DBRM')
```

3. Defina el procedimiento almacenado en Db2 emitiendo la instrucción CREATE PROCEDURE con la opción EXTERNAL. Utilice la cláusula EXTERNAL NAME para especificar el nombre del módulo de carga para el programa que se ejecuta cuando se llama a este procedimiento.

Si desea ejecutar su procedimiento como un z/OS -programa autorizado, especifique un entorno adecuado con la opción ENTORNO WLM. El procedimiento almacenado debe ejecutarse en un espacio de direcciones con un procedimiento de inicio en el que todas las bibliotecas de la concatenación STEPLIB estén autorizadas por APF.

Si desea que la información del entorno se transmita al procedimiento almacenado cuando se invoque, especifique las opciones DBINFO y PARAMETER STYLE SQL en la instrucción CREATE PROCEDURE. Cuando se invoca el procedimiento, Db2 pasa la estructura DBINFO, que contiene información del entorno, al procedimiento almacenado. Para más información sobre PARAMETER STYLE, consulte “[Definir la convención de enlace para un procedimiento almacenado externo](#)” en la página 274.

Si compiló el procedimiento almacenado como reentrant, especifique la opción STAY RESIDENT YES en la instrucción CREATE PROCEDURE. Esta opción hace que el procedimiento permanezca en almacenamiento.

4. Autorizar a los usuarios adecuados a utilizar el procedimiento almacenado emitiendo la sentencia GRANT EXECUTE.

Por ejemplo, la siguiente declaración permite que una aplicación que se ejecuta bajo el ID de autorización JONES llame al procedimiento almacenado SPSHEMA.STORPRCA:

```
GRANT EXECUTE ON PROCEDURE SPSHEMA.STORPRCA TO JONES;
```

Ejemplo de definición de un procedimiento almacenado C

Supongamos que ha escrito y preparado un procedimiento almacenado que tiene las siguientes características:

- El nombre del procedimiento almacenado es B.
- El procedimiento almacenado tiene los dos parámetros siguientes:
 - Un parámetro de entrada entero que se denomina V1
 - Un parámetro de salida de caracteres de longitud 9 que se denomina V2
- El procedimiento almacenado está escrito en lenguaje C.
- El procedimiento almacenado no contiene instrucciones SQL.
- La misma entrada siempre produce la misma salida.

- El nombre del módulo de carga es SUMMOD.
- El nombre de la colección del paquete es SUMCOLL.
- El procedimiento almacenado debe ejecutarse durante no más de 900 unidades de servicio de CPU.
- Los parámetros pueden tener valores nulos.
- El procedimiento almacenado debe eliminarse de la memoria cuando finalice.
- El procedimiento almacenado necesita las siguientes opciones de tiempo de ejecución de Language Environment :

```
MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)
```

- El procedimiento almacenado forma parte del entorno de aplicación WLM que se denomina PAYROLL.
- El procedimiento almacenado se ejecuta como un programa principal.
- El procedimiento almacenado no accede a recursos no eDb2 es, por lo que no necesita un RACF entorno especial.
- El procedimiento almacenado puede devolver como máximo 10 conjuntos de resultados.
- Cuando el control vuelve al programa cliente, Db2 no realiza las actualizaciones automáticamente.

La siguiente instrucción CREATE PROCEDURE define el procedimiento almacenado para Db2:

```
CREATE PROCEDURE B(IN V1 INTEGER, OUT V2 CHAR(9))
LANGUAGE C
DETERMINISTIC
NO SQL
EXTERNAL NAME SUMMOD
COLLID SUMCOLL
ASUTIME LIMIT 900
PARAMETER STYLE GENERAL WITH NULLS
STAY RESIDENT NO
RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
WLM ENVIRONMENT PAYROLL
PROGRAM TYPE MAIN
SECURITY DB2
DYNAMIC RESULT SETS 10
COMMIT ON RETURN NO;
```

Qué hacer a continuación

Ahora puede invocar el procedimiento almacenado desde [un programa de aplicación](#) o desde [el procesador de línea de comandos de Db2](#).

Conceptos relacionados

[“Interfaz de lenguaje universal \(DSNULI\)” en la página 121](#)

El subcomponente de la interfaz de lenguaje universal (DSNULI) determina el entorno en tiempo de ejecución y carga y se bifurca dinámicamente al modelo de interfaz de lenguaje adecuado.

[Procedimientos almacenados y funciones definidas por el usuario de Java \(Db2 Application Programming for Java\)](#)

Tareas relacionadas

[Implementación de procedimientos almacenados de Db2 \(Db2 Administration Guide\)](#)

Referencia relacionada

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2\)](#)

[Instrucción CREATE PROCEDURE \(procedimiento externo\) \(Db2 SQL\)](#)

[Declaración GRANT \(privilegios de función o procedimiento\) \(Db2 SQL\)](#)

Información relacionada

[Db2 para procedimientos almacenados de SQL Server \(z/OS Stored Procedures: Through the CALL and Beyond \(IBM Redbooks \)](#)

Definir la convención de enlace para un procedimiento almacenado externo

Una convención de vinculación especifica las reglas para la lista de parámetros que pasa el programa que llama al procedimiento almacenado externo. Por ejemplo, la convención puede especificar si el programa de llamada puede pasar valores nulos para los parámetros de entrada.

Procedimiento

Cuando defina el procedimiento almacenado con la instrucción CREATE PROCEDURE, especifique uno de los siguientes valores para la opción PARAMETER STYLE:

- GENERAL
- GENERAL WITH NULLS
- SQL

SQL es el predeterminado.

Convenciones de enlace para procedimientos de almacenamiento externo

La convención de enlace para un procedimiento almacenado puede ser GENERAL, GENERAL WITH NULLS o SQL. Estas convenciones de enlace se aplican únicamente a procedimientos de almacenamiento externos.

GENERAL

Especifique la convención de vinculación GENERAL cuando no desee que el programa de llamada pase valores nulos para los parámetros de entrada (IN o INOUT) al procedimiento almacenado. Si especifica GENERAL, asegúrese de que el procedimiento almacenado contiene una declaración de variable para cada parámetro que se pasa en la instrucción CALL.

La siguiente figura muestra la estructura de la lista de parámetros para PARAMETER STYLE GENERAL.

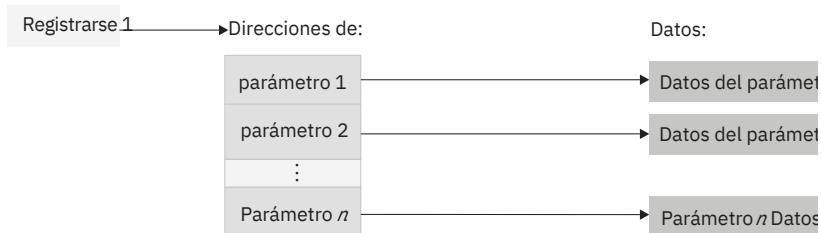


Figura 12. Convención de parámetros GENERAL para un procedimiento almacenado

GENERAL WITH NULLS

Especifique la convención de enlace GENERAL WITH NULLS cuando desee permitir que el programa de llamada proporcione un valor nulo para cualquier parámetro que se pase al procedimiento almacenado. Si especifica GENERAL CON NULLS, asegúrese de que el procedimiento almacenado realice las siguientes tareas:

- Declara una variable para cada parámetro que se pasa en la instrucción CALL.
- Declara una estructura de indicador nulo que contiene una variable de indicador para cada parámetro.
- Al entrar, examina todas las variables indicadoras que están asociadas con los parámetros de entrada para determinar qué parámetros contienen valores nulos.
- Al salir, asigna valores a todas las variables indicadoras que están asociadas con las variables de salida. Si la variable de salida devuelve un valor nulo al llamador, asigne un número negativo a la variable indicadora asociada. De lo contrario, asigne un valor de 0 a la variable indicadora.

En la instrucción CALL de la aplicación de llamada, siga cada parámetro con su variable indicadora. Utilice uno de los siguientes formularios:

- *variable-host :variable-indicador*
- *variable host INDICADOR: variable indicadora*

La siguiente figura muestra la estructura de la lista de parámetros para PARAMETER STYLE GENERAL WITH NULLS.

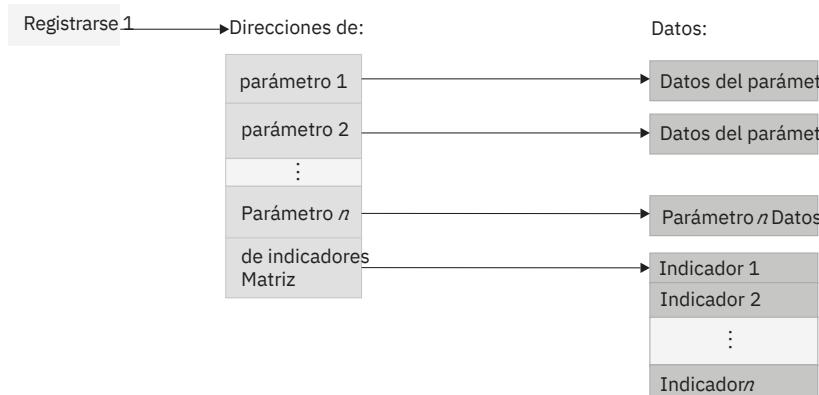


Figura 13. Convención de parámetros GENERAL WITH NULLS para un procedimiento almacenado

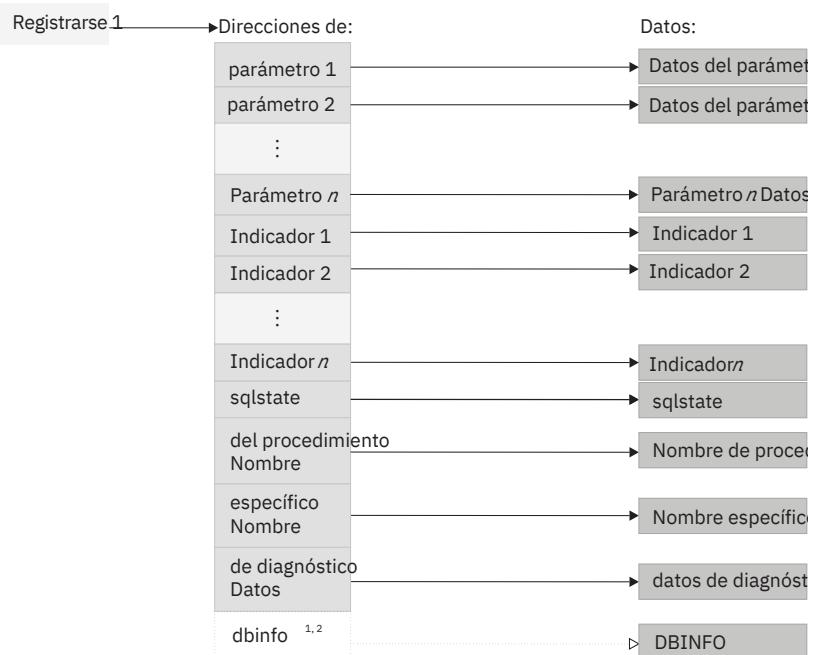
SQL

Especifique la convención de vinculación SQL cuando desee que se cumplan las dos condiciones siguientes:

- El programa de llamada debe poder proporcionar un valor nulo para cualquier parámetro que se pase al procedimiento almacenado.
- Db2 para pasar parámetros de entrada y salida al procedimiento almacenado que contenga la siguiente información:
 - El SQLSTATE que se devolverá a Db2. Este valor es un parámetro CHAR(5) que representa el SQLSTATE que se pasa al programa desde el gestor de la base de datos. El valor inicial se establece en «00000». Aunque el programa no suele establecer el SQLSTATE, se puede establecer como el resultado SQLSTATE que se utiliza para devolver un error o una advertencia. Los valores devueltos que comienzan con cualquier otra cifra que no sea «00», «01» o «02» son condiciones de error.
 - El nombre calificado del procedimiento almacenado. Este es un valor VARCHAR(128).
 - El nombre específico del procedimiento almacenado. El nombre específico es un valor VARCHAR(128) que es el mismo que el nombre sin calificador.
 - La cadena de diagnóstico SQL que se debe devolver a Db2. Este es un valor VARCHAR(1000). Utilice esta área para transmitir información descriptiva sobre un error o advertencia a la persona que llama.

Restricción: No puede utilizar la convención de vinculación SQL para un procedimiento almacenado en lenguaje REXX.

La siguiente figura muestra la estructura de la lista de parámetros para PARAMETER STYLE SQL.



¹ Para PL/I, este valor es la dirección de un puntero a los datos DBINFO.

² Se aprueba si la opción DBINFO se especifica en la definición de la función definida por el

Figura 14. Convención de parámetros SQL para un procedimiento almacenado

Conceptos relacionados

[Programas de ejemplo que llaman a procedimientos almacenados](#)

Los ejemplos se pueden utilizar como modelos cuando escribe aplicaciones que llaman a procedimientos almacenados. Asimismo, *prefijo.SDSNSAMP* contiene trabajos de ejemplo de DSNTEJ6P y DSNTEJ6S y programas DSN8EP1 y DSN8EP2 de ejemplo, que puede ejecutar.

Referencia relacionada

[Instrucción CREATE PROCEDURE \(procedimiento externo\) \(Db2 SQL\)](#)

[Valores de SQLSTATE y códigos de error comunes \(Db2 Codes\)](#)

Ejemplo de convención de vinculación GENERAL

Especifique la convención de vinculación GENERAL cuando no desee que el programa de llamada pase valores nulos para los parámetros de entrada (IN o INOUT) al procedimiento almacenado.

ejemplos

Los siguientes ejemplos demuestran cómo un ensamblador, C, COBOL o procedimiento almacenado PL/I utiliza la convención de vinculación GENERAL para recibir parámetros.

Para estos ejemplos, suponga que una aplicación COBOL tiene las siguientes declaraciones de parámetros y la instrucción CALL:

```
*****
* PARAMETERS FOR THE SQL STATEMENT CALL *
*****
01 V1 PIC S9(9) USAGE COMP.
01 V2 PIC X(9).
:
EXEC SQL CALL A (:V1, :V2) END-EXEC.
```

En la declaración CREATE PROCEDURE, los parámetros se definen de la siguiente manera:

```
IN V1 INT, OUT V2 CHAR(9)
```

Ejemplo de ensamblador

El siguiente ejemplo muestra cómo un procedimiento almacenado que está escrito en lenguaje ensamblador recibe estos parámetros.

```
*****
* CODE FOR AN ASSEMBLER LANGUAGE STORED PROCEDURE THAT USES      *
* THE GENERAL LINKAGE CONVENTION.                                     *
*****
A      CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS
      USING PROGAREA,R13
*****
* BRING UP THE LANGUAGE ENVIRONMENT.                                *
*****
:
*****
* GET THE PASSED PARAMETER VALUES. THE GENERAL LINKAGE CONVENTION*
* FOLLOWS THE STANDARD ASSEMBLER LINKAGE CONVENTION:              *
* ON ENTRY, REGISTER 1 POINTS TO A LIST OF POINTERS TO THE        *
* PARAMETERS.                                                       *
*****
L      R7,0(R1)           GET POINTER TO V1
MVC    LOCV1(4),0(R7)     MOVE VALUE INTO LOCAL COPY OF V1
:
L      R7,4(R1)           GET POINTER TO V2
MVC    0(9,R7),LOCV2      MOVE A VALUE INTO OUTPUT VAR V2
:
CEETERM RC=0
*****
* VARIABLE DECLARATIONS AND EQUATES                               *
*****
R1      EQU   1             REGISTER 1
R7      EQU   7             REGISTER 7
PPA     CEEPPA ,           CONSTANTS DESCRIBING THE CODE BLOCK
      LTORG ,
PROGAREA DSECT
      ORG   *+CEEDSASZ    LEAVE SPACE FOR DSA FIXED PART
LOCV1   DS    F             LOCAL COPY OF PARAMETER V1
LOCV2   DS    CL9           LOCAL COPY OF PARAMETER V2
:
PROGSIZE EQU   *-PROGAREA
      CEEDSA ,           MAPPING OF THE DYNAMIC SAVE AREA
      CEECAA ,           MAPPING OF THE COMMON ANCHOR AREA
      END   A
```

Ejemplo C

La siguiente figura muestra cómo un procedimiento almacenado que está escrito en lenguaje C recibe estos parámetros.

```
#pragma runopts(PLIST(OS))
#pragma options(RENT)
#include <stdlib.h>
#include <stdio.h>
/****************************************************************************
/* Code for a C language stored procedure that uses the          */
/* GENERAL linkage convention.                                    */
/****************************************************************************
main(argc,argv)
    int argc;                      /* Number of parameters passed */
    char *argv[];                  /* Array of strings containing */
                                   /* the parameter values */
{
    long int locv1;                /* Local copy of V1 */
    char locv2[10];                /* Local copy of V2 */
                                   /* (null-terminated) */
:
/****************************************************************************
/* Get the passed parameters. The GENERAL linkage convention */
/* follows the standard C language parameter passing */
/* conventions:                                                 */
/* - argc contains the number of parameters passed            */
/* - argv[0] is a pointer to the stored procedure name       */
/* - argv[1] to argv[n] are pointers to the n parameters     */
/*   in the SQL statement CALL.                                */
/****************************************************************************
if(argc==3)                    /* Should get 3 parameters: */
```

```

{
    locv1 = *(int *) argv[1];          /* procname, V1, V2      */
    :                                /* Get local copy of V1   */
    strcpy(argv[2],locv2);            /* Assign a value to V2    */
}

```

Ejemplo de COBOL

La siguiente figura muestra cómo un procedimiento almacenado que está escrito en el lenguaje COBOL recibe estos parámetros.

```

CBL RENT
IDENTIFICATION DIVISION.
***** 
* CODE FOR A COBOL LANGUAGE STORED PROCEDURE THAT USES THE *
* GENERAL LINKAGE CONVENTION.                                 *
***** 
PROGRAM-ID.      A.
:
DATA DIVISION.
:
LINKAGE SECTION.
***** 
* DECLARE THE PARAMETERS PASSED BY THE SQL STATEMENT      *
* CALL HERE.                                                 *
***** 
01 V1 PIC S9(9) USAGE COMP.
01 V2 PIC X(9).
:
PROCEDURE DIVISION USING V1, V2.
***** 
* THE USING PHRASE INDICATES THAT VARIABLES V1 AND V2      *
* WERE PASSED BY THE CALLING PROGRAM.                      *
***** 
* ASSIGN A VALUE TO OUTPUT VARIABLE V2 *
***** 
MOVE '123456789' TO V2.

```

Ejemplo PL/I

La siguiente figura muestra cómo un procedimiento almacenado que está escrito en el lenguaje PL/I recibe estos parámetros.

```

*PROCESS SYSTEM(MVS);
A: PROC(V1, V2) OPTIONS(MAIN NOEXECOPS REENTRANT);
/******
/* Code for a PL/I language stored procedure that uses the      */
/* GENERAL linkage convention.                                     */
***** 
/* Indicate on the PROCEDURE statement that two parameters       */
/* were passed by the SQL statement CALL. Then declare the      */
/* parameters in the following section.                           */
***** 
DCL V1 BIN FIXED(31),
     V2 CHAR(9);
:
V2 = '123456789';    /* Assign a value to output variable V2 */

```

Ejemplo de convención de vinculación GENERAL CON NULOS

Especifique la convención de enlace GENERAL WITH NULLS cuando desee permitir que el programa de llamada proporcione un valor nulo para cualquier parámetro que se pase al procedimiento almacenado.

ejemplos

Los siguientes ejemplos demuestran cómo un ensamblador, C, COBOL o procedimiento almacenado PL/I utiliza la convención de vinculación GENERAL WITH NULLS para recibir parámetros.

Para estos ejemplos, suponga que una aplicación C tiene las siguientes declaraciones de parámetros y la instrucción CALL:

```
*****  
/* Parameters for the SQL statement CALL */  
*****  
long int v1; /* Allow an extra byte for */  
char v2[10]; /* the null terminator */  
*****  
/* Indicator structure */  
*****  
struct indicators {  
    short int ind1;  
    short int ind2;  
} indstruc;  
...  
indstruc.ind1 = 0; /* Remember to initialize the */  
/* input parameter's indicator*/  
/* variable before executing */  
/* the CALL statement */  
EXEC SQL CALL B (:v1 :indstruc.ind1, :v2 :indstruc.ind2);  
:
```

En la declaración CREATE PROCEDURE, los parámetros se definen de la siguiente manera:

```
IN V1 INT, OUT V2 CHAR(9)
```

Ejemplo de ensamblador

La siguiente figura muestra cómo un procedimiento almacenado que está escrito en lenguaje ensamblador recibe estos parámetros.

```
*****  
* CODE FOR AN ASSEMBLER LANGUAGE STORED PROCEDURE THAT USES *  
* THE GENERAL WITH NULLS LINKAGE CONVENTION. *  
*****  
B      CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS  
      USING PROGAREA,R13  
*****  
* BRING UP THE LANGUAGE ENVIRONMENT. *  
*****  
:  
*****  
* GET THE PASSED PARAMETER VALUES. THE GENERAL WITH NULLS LINKAGE*  
* CONVENTION IS AS FOLLOWS: *  
* ON ENTRY, REGISTER 1 POINTS TO A LIST OF POINTERS. IF N *  
* PARAMETERS ARE PASSED, THERE ARE N+1 POINTERS. THE FIRST *  
* N POINTERS ARE THE ADDRESSES OF THE N PARAMETERS, JUST AS *  
* WITH THE GENERAL LINKAGE CONVENTION. THE N+1ST POINTER IS *  
* THE ADDRESS OF A LIST CONTAINING THE N INDICATOR VARIABLE *  
* VALUES. *  
*****  
L      R7,0(R1)      GET POINTER TO V1  
MVC   L0CV1(4),0(R7) MOVE VALUE INTO LOCAL COPY OF V1  
L      R7,8(R1)      GET POINTER TO INDICATOR ARRAY  
MVC   LOCIND(2*2),0(R7) MOVE VALUES INTO LOCAL STORAGE  
LH    R7,LOCIND    GET INDICATOR VARIABLE FOR V1  
LTR   R7,R7        CHECK IF IT IS NEGATIVE  
BM    NULLIN       IF SO, V1 IS NULL  
:  
L      R7,4(R1)      GET POINTER TO V2  
MVC   0(9,R7),LOCV2 MOVE A VALUE INTO OUTPUT VAR V2  
L      R7,8(R1)      GET POINTER TO INDICATOR ARRAY  
MVC   2(2,R7),=H(0) MOVE ZERO TO V2'S INDICATOR VAR  
:  
CEETERM RC=0  
*****  
* VARIABLE DECLARATIONS AND EQUATES *  
*****  
R1    EQU 1          REGISTER 1
```

```

R7      EQU    7          REGISTER 7
PPA     CEEPPA  ,
LTORG  ,
PROGAREA DSECT
ORG    *+CEEDSASZ   LEAVE SPACE FOR DSA FIXED PART
LOCV1  DS     F          LOCAL COPY OF PARAMETER V1
LOCV2  DS     CL9        LOCAL COPY OF PARAMETER V2
LOCIND DS     2H         LOCAL COPY OF INDICATOR ARRAY
:
PROGSIZE EQU    *-PROGAREA
CEEDSA  ,
CEECAA  ,
END    B          MAPPING OF THE DYNAMIC SAVE AREA
                  MAPPING OF THE COMMON ANCHOR AREA

```

Ejemplo C

La siguiente figura muestra cómo un procedimiento almacenado que está escrito en lenguaje C recibe estos parámetros.

```

#pragma options(RENT)
#pragma runopts(PLIST(OS))
#include <stdlib.h>
#include <stdio.h>
/*********************************************************/
/* Code for a C language stored procedure that uses the */
/* GENERAL WITH NULLS linkage convention. */
/*********************************************************/
main(argc,argv)
int argc;           /* Number of parameters passed */
char *argv[];       /* Array of strings containing */
                   /* the parameter values */
{
long int locv1;     /* Local copy of V1 */
char locv2[10];    /* Local copy of V2 */
                   /* (null-terminated) */
short int locind[2];/* Local copy of indicator */
                   /* variable array */
short int *tempint; /* Used for receiving the */
                   /* indicator variable array */
:
/*********************************************************/
/* Get the passed parameters. The GENERAL WITH NULLS linkage */
/* convention is as follows: */
/* - argc contains the number of parameters passed */
/* - argv[0] is a pointer to the stored procedure name */
/* - argv[1] to argv[n] are pointers to the n parameters */
/* in the SQL statement CALL. */
/* - argv[n+1] is a pointer to the indicator variable array */
/*********************************************************/
if(argc==4)
{
    /* Should get 4 parameters: */
    /* procname, V1, V2, */
    /* indicator variable array */
    locv1 = *(int *) argv[1];
    /* Get local copy of V1 */
    tempint = argv[3];
    /* Get pointer to indicator */
    /* variable array */
    locind[0] = *tempint;
    /* Get 1st indicator variable */
    locind[1] = *(++tempint);
    if(locind[0]<0)
    {
        /* Get 2nd indicator variable */
        /* If 1st indicator variable */
        /* is negative, V1 is null */
        :
    }
    :
    strcpy(argv[2],locv2);
    /* Assign a value to V2 */
    *(++tempint) = 0;
    /* Assign 0 to V2's indicator */
    /* variable */
}
}

```

Ejemplo de COBOL

La siguiente figura muestra cómo un procedimiento almacenado que está escrito en lenguaje COBOL recibe estos parámetros.

```
CBL RENT
IDENTIFICATION DIVISION.
*****
* CODE FOR A COBOL LANGUAGE STORED PROCEDURE THAT USES THE *
* GENERAL WITH NULLS LINKAGE CONVENTION. *
*****
PROGRAM-ID. B.
.
.
.
DATA DIVISION.
.
.
.
LINKAGE SECTION.
*****
* DECLARE THE PARAMETERS AND THE INDICATOR ARRAY THAT      *
* WERE PASSED BY THE SQL STATEMENT CALL HERE.           *
*****
01 V1 PIC S9(9) USAGE COMP.
01 V2 PIC X(9).
*
01 INDARRAY.
  10 INDDVAR PIC S9(4) USAGE COMP OCCURS 2 TIMES.
.
.
.
PROCEDURE DIVISION USING V1, V2, INDARRAY.
*****
* THE USING PHRASE INDICATES THAT VARIABLES V1, V2, AND      *
* INDARRAY WERE PASSED BY THE CALLING PROGRAM.           *
*****
.
.
.
*****
* TEST WHETHER V1 IS NULL *
*****
  IF INDARRAY(1) < 0
    PERFORM NULL-PROCESSING.
.
.
.
*****
* ASSIGN A VALUE TO OUTPUT VARIABLE V2 *
* AND ITS INDICATOR VARIABLE          *
*****
  MOVE '123456789' TO V2.
  MOVE ZERO TO INDARRAY(2).
```

Ejemplo PL/I

La siguiente figura muestra cómo un procedimiento almacenado que está escrito en el lenguaje PL/I recibe estos parámetros.

```
*PROCESS SYSTEM(MVS);
A: PROC(V1, V2, INDSTRUC) OPTIONS(MAIN NOEXECOPS REENTRANT);
/*****************************************/
/* Code for a PL/I language stored procedure that uses the      */
/* GENERAL WITH NULLS linkage convention.                         */
/*****************************************/
/*****************************************/
/* Indicate on the PROCEDURE statement that two parameters       */
/* and an indicator variable structure were passed by the SQL */
/* statement CALL. Then declare them in the following section.*/
/* For PL/I, you must declare an indicator variable structure, */
/* not an array.                                                 */
/*****************************************/
      DCL V1 BIN FIXED(31),
          V2 CHAR(9);
      DCL
        01 INDSTRUC,
          02 IND1 BIN FIXED(15),
          02 IND2 BIN FIXED(15);
.
.
.
IF IND1 < 0 THEN
  CALL NULLVAL;      /* If indicator variable is negative   */
                     /* then V1 is null                  */
.
.
.
V2 = '123456789';  /* Assign a value to output variable V2 */
IND2 = 0;            /* Assign 0 to V2's indicator variable */
```

Ejemplo de convención de vinculación SQL

Especifique la convención de vinculación SQL cuando desee que la información de diagnóstico se transmita en los parámetros y permita valores nulos.

ejemplos

Los siguientes ejemplos demuestran cómo un ensamblador, C, COBOL o procedimiento almacenado PL/I utiliza la convención de vinculación SQL para recibir parámetros. Estos ejemplos también muestran cómo un procedimiento almacenado recibe la estructura DBINFO.

Para estos ejemplos, suponga que una aplicación C tiene las siguientes declaraciones de parámetros y la instrucción CALL:

```
*****  
/* Parameters for the SQL statement CALL */  
*****  
long int v1;                                /* Allow an extra byte for */  
char v2[10];                                 /* the null terminator */  
*****  
/* Indicator variables */  
*****  
short int ind1;                                /* Remember to initialize the */  
short int ind2;                               /* input parameter's indicator*/  
                                         /* variable before executing */  
                                         /* the CALL statement */  
EXEC SQL CALL B (:v1 :ind1, :v2 :ind2);  
*****
```

En la declaración CREATE PROCEDURE, los parámetros se definen de la siguiente manera:

```
IN V1 INT, OUT V2 CHAR(9)
```

Ejemplo de ensamblador

La siguiente figura muestra cómo un procedimiento almacenado que está escrito en lenguaje ensamblador recibe estos parámetros.

```
*****  
* CODE FOR AN ASSEMBLER LANGUAGE STORED PROCEDURE THAT USES  
*  
* THE SQL LINKAGE CONVENTION.                                     *  
*****  
B      CEEENTRY AUTO=PROGSIZE,MAIN=YES,PLIST=OS  
      USING PROGAREA,R13  
*****  
* BRING UP THE LANGUAGE ENVIRONMENT.  
*  
*****  
*  
*****  
* GET THE PASSED PARAMETER VALUES. THE SQL LINKAGE          *  
* CONVENTION IS AS FOLLOWS:  
*  
* ON ENTRY, REGISTER 1 POINTS TO A LIST OF POINTERS. IF N  
*  
* PARAMETERS ARE PASSED, THERE ARE 2N+4 POINTERS. THE FIRST  
*  
* N POINTERS ARE THE ADDRESSES OF THE N PARAMETERS, JUST AS  
*  
* WITH THE GENERAL LINKAGE CONVENTION. THE NEXT N POINTERS ARE  
*  
* THE ADDRESSES OF THE INDICATOR VARIABLE VALUES. THE LAST  
*  
* 4 POINTERS (5, IF DBINFO IS PASSED) ARE THE ADDRESSES OF  
*  
* INFORMATION ABOUT THE STORED PROCEDURE ENVIRONMENT AND  
*  
* EXECUTION RESULTS.  
*
```

```
*****
L    R7,0(R1)      GET POINTER TO V1
MVC LOCV1(4),0(R7) MOVE VALUE INTO LOCAL COPY OF V1
L    R7,8(R1)      GET POINTER TO 1ST INDICATOR VARIABLE
MVC LOCI1(2),0(R7) MOVE VALUE INTO LOCAL STORAGE
L    R7,20(R1)     GET POINTER TO STORED PROCEDURE
NAME
MVC LOCSPNM(20),0(R7) MOVE VALUE INTO LOCAL STORAGE
L    R7,24(R1)     GET POINTER TO DBINFO
MVC LOCDBINF(DBINFLN),0(R7)
*
LH   R7,LOCI1      MOVE VALUE INTO LOCAL STORAGE
LTR  R7,R7        GET INDICATOR VARIABLE FOR V1
BM   NULLIN       CHECK IF IT IS NEGATIVE
:    BM            IF SO, V1 IS NULL
L    R7,4(R1)      GET POINTER TO V2
MVC 0(9,R7),LOCV2 MOVE A VALUE INTO OUTPUT VAR V2
L    R7,12(R1)     GET POINTER TO INDICATOR VAR 2
MVC 0(2,R7),=H'0' MOVE ZERO TO V2'S INDICATOR VAR
L    R7,16(R1)     GET POINTER TO SQLSTATE
MVC 0(5,R7),=CL5'xxxxx' MOVE xxxx TO SQLSTATE
:
CEETERM RC=0
```

```
*****
* VARIABLE DECLARATIONS AND EQUATES
*
*****
R1    EQU 1          REGISTER 1
R7    EQU 7          REGISTER 7
PPA   CEEPPA ,       CONSTANTS DESCRIBING THE CODE BLOCK
                   LTORG ,
PROGAREA DSECT      PLACE LITERAL POOL HERE
ORG   *+CEEDSASZ    LEAVE SPACE FOR DSA FIXED PART
LOCV1 DS   F         LOCAL COPY OF PARAMETER V1
LOCV2 DS   CL9       LOCAL COPY OF PARAMETER V2
LOCI1 DS   H         LOCAL COPY OF INDICATOR 1
LOCI2 DS   H         LOCAL COPY OF INDICATOR 2
LOCSQLST DS  CL5     LOCAL COPY OF SQLSTATE
LOCSPNM DS  H,CL27   LOCAL COPY OF STORED PROC NAME
LOCSPSNM DS  H,CL18   LOCAL COPY OF SPECIFIC NAME
LOCDIAG DS  H,CL1000 LOCAL COPY OF DIAGNOSTIC DATA
LOCDBINF DS  OH      LOCAL COPY OF DBINFO DATA
DBNAMELN DS  H      DATABASE NAME LENGTH
DBNAME  DS  CL128   DATABASE NAME
AUTHIDLN DS  H      APPL AUTH ID LENGTH
AUTHID  DS  CL128   APPL AUTH ID
ASC_SBCS DS  F      ASCII SBCS CCSID
ASC_DBCS DS  F      ASCII DBCS CCSID
ASC_MIXD DS  F      ASCII MIXED CCSID
EBC_SBCS DS  F      EBCDIC SBCS CCSID
EBC_DBCS DS  F      EBCDIC DBCS CCSID
EBC_MIXD DS  F      EBCDIC MIXED CCSID
UNI_SBCS DS  F      UNICODE SBCS CCSID
UNI_DBCS DS  F      UNICODE DBCS CCSID
UNI_MIXD DS  F      UNICODE MIXED CCSID
ENCODE  DS  F      PROCEDURE ENCODING SCHEME
RESERVO DS  CL20    RESERVED
TBQUALLN DS  H      TABLE QUALIFIER LENGTH
TBQUAL   DS  CL128   TABLE QUALIFIER
TBNAMELN DS  H      TABLE NAME LENGTH
TBNAME   DS  CL128   TABLE NAME
CLNAMELN DS  H      COLUMN NAME LENGTH
COLNAME  DS  CL128   COLUMN NAME
RELVER   DS  CL8    DBMS RELEASE AND VERSION
RESERV1  DS  CL2    RESERVED
PLATFORM DS  F      DBMS OPERATING SYSTEM
NUMTFCOL DS  H      NUMBER OF TABLE FUNCTION COLS USED
RESERV2  DS  CL26   RESERVED
TFCOLNUM DS  A      POINTER TO TABLE FUNCTION COL LIST
APPLID   DS  A      POINTER TO APPLICATION ID
RESERV3  DS  CL20   RESERVED
DBINFLN EQU  *-LOCDBINF LENGTH OF DBINFO
:
PROGSIZE EQU  *-PROGAREA
CEEDSA  ,           MAPPING OF THE DYNAMIC SAVE AREA
CEECAA  ,           MAPPING OF THE COMMON ANCHOR AREA
END    B
```

Ejemplo C

La siguiente figura muestra cómo un procedimiento almacenado que está escrito como un programa principal en lenguaje C recibe estos parámetros.

```
#pragma runopts(plist(os))
#include <stdlib.h>
#include <stdio.h>

main(argc,argv)
    int argc;
    char *argv[];
{
    int parm1;
    short int ind1;
    char p_proc[28];
    char p_spec[19];
/*****
/* Assume that the SQL CALL statement included */
/* 3 input/output parameters in the parameter list.*/
/* The argv vector will contain these entries: */
/*      argv[0]      1  contains load module */
/*      argv[1-3]     3  input/output parms */
/*      argv[4-6]     3  null indicators */
/*      argv[7]       1  SQLSTATE variable */
/*      argv[8]       1  qualified proc name */
/*      argv[9]       1  specific proc name */
/*      argv[10]      1  diagnostic string */
/*      argv[11]      + 1 dbinfo */
/*      ----- */
/*          12   for the argc variable */
/*****
if argc<>12 {
:
/* We end up here when invoked with wrong number of parms */
}

/*****
/* Assume the first parameter is an integer. */
/* The following code shows how to copy the integer*/
/* parameter into the application storage. */
/*****
parm1 = *(int *) argv[1];
/*****
/* We can access the null indicator for the first */
/* parameter on the SQL CALL as follows: */
/*****
ind1 = *(short int *) argv[4];
/*****
/* We can use the following expression */
/* to assign 'xxxxx' to the SQLSTATE returned to */
/* caller on the SQL CALL statement. */
/*****
strcpy(argv[7],"xxxxx/0");
/*****
/* We obtain the value of the qualified procedure */
/* name with this expression. */
/*****
strcpy(p_proc,argv[8]);
/*****
/* We obtain the value of the specific procedure */
/* name with this expression. */
/*****
strcpy(p_spec,argv[9]);
/*****
/* We can use the following expression to assign */
/* 'yyyyyyyy' to the diagnostic string returned */
/* in the SQLDA associated with the CALL statement.*/
/*****
strcpy(argv[10],"yyyyyyyy/0");
:
}
```

La siguiente figura muestra cómo un procedimiento almacenado que está escrito como un subprograma en el lenguaje C recibe estos parámetros.

```
#pragma linkage(myproc,fetchable)
#include <stdlib.h>
```

```

#include <stdio.h>
#include <sqludf.h>

void myproc(*parm1 int,          /* assume INT for PARM1
   */
            parm2 char[11],      /* assume CHAR(10) parm2
   */
   :
            *p_ind1 short int,    /* null indicator for parm1
   */
            *p_ind2 short int,    /* null indicator for parm2
   */
   :
            p_sqlstate char[6],    /* SQLSTATE returned to DB2
   */
            p_proc char[28],      /* Qualified stored proc name
   */
            p_spec char[19],      /* Specific stored proc name
   */
            p_diag char[1001],     /* Diagnostic string
   */
            struct sqludf_dbinfo *udf_dbinfo); /* DBINFO
   */
{
    int l_p1;
    char[11] l_p2;
    short int l_ind1;
    short int l_ind2;
    char[6] l_sqlstate;
    char[28] l_proc;
    char[19] l_spec;
    char[71] l_diag;
    sqludf_dbinfo *ludf_dbinfo;
   :
   ****
   /* Copy each of the parameters in the parameter */
   /* list into a local variable, just to demonstrate */
   /* how the parameters can be referenced. */
   ****
   l_p1 = *parm1;

    strcpy(l_p2,parm2);

    l_ind1 = *p_ind1;
    l_ind1 = *p_ind2;

    strcpy(l_sqlstate,p_sqlstate);
    strcpy(l_proc,p_proc);
    strcpy(l_spec,p_spec);

    strcpy(l_diag,p_diag);
    memcpy(&ludf_dbinfo,udf_dbinfo,sizeof(ludf_dbinfo));
   :
}

```

Ejemplo de COBOL

La siguiente figura muestra cómo un procedimiento almacenado que está escrito en el lenguaje COBOL recibe estos parámetros.

```

CBL RENT
IDENTIFICATION DIVISION.
:
DATA DIVISION.
:
LINKAGE SECTION.
* Declare each of the parameters
01 PARM1 ...
01 PARM2 ...
:
* Declare a null indicator for each parameter
01 P-IND1 PIC S9(4) USAGE COMP.
01 P-IND2 PIC S9(4) USAGE COMP.
:
* Declare the SQLSTATE that can be set by stored proc
01 P-SQLSTATE PIC X(5).

```

```

* Declare the qualified procedure name
01 P-PROC.
  49 P-PROC-LEN PIC 9(4) USAGE BINARY.
  49 P-PROC-TEXT PIC X(27).
* Declare the specific procedure name
01 P-SPEC.
  49 P-SPEC-LEN PIC 9(4) USAGE BINARY.
  49 P-SPEC-TEXT PIC X(18).
* Declare SQL diagnostic message token
01 P-DIAG.
  49 P-DIAG-LEN PIC 9(4) USAGE BINARY.
  49 P-DIAG-TEXT PIC X(1000).
*****
* Structure used for DBINFO
*****
01 SQLUDF-DBINFO.
*   Location name length
  05 DBNAMELEN PIC 9(4) USAGE BINARY.
*   Location name
  05 DBNAME PIC X(128).
*   authorization ID length
  05 AUTHIDLEN PIC 9(4) USAGE BINARY.
*   authorization ID
  05 AUTHID PIC X(128).
*   environment CCSID information
  05 CODEPG PIC X(48).
  05 CDPG-DB2 REDEFINES CODEPG.
    10 DB2-CCSID OCCURS 3 TIMES.
      15 DB2-SBCS PIC 9(9) USAGE BINARY.
      15 DB2-DBCS PIC 9(9) USAGE BINARY.
      15 DB2-MIXED PIC 9(9) USAGE BINARY.
    10 ENCODING-SCHEME PIC 9(9) USAGE BINARY.
    10 RESERVED PIC X(20).

```

```

* other platform-specific
deprecated CCSID structures not included here
*   schema name length
  05 TBSCHHEMALEN PIC 9(4) USAGE BINARY.
*   schema name
  05 TBSCHHEMA PIC X(128).
*   table name length
  05 TBNAMELEN PIC 9(4) USAGE BINARY.
*   table name
  05 TBNAME PIC X(128).
*   column name length
  05 COLNAMELEN PIC 9(4) USAGE BINARY.
*   column name
  05 COLNAME PIC X(128).
*   product information
  05 VER-REL PIC X(8).
*   reserved
  05 RESD0 PIC X(2).
*   platform type
  05 PLATFORM PIC 9(9) USAGE BINARY.
*   number of entries in the TF column list array (tfcolumn, below)
  05 NUMTFCOL PIC 9(4) USAGE BINARY.
*   reserved
  05 RESD1 PIC X(26).
*   tfcolumn will be allocated dynamically if it is defined
*   otherwise this will be a null pointer
  05 TFCOLUMN USAGE IS POINTER.
*   application identifier
  05 APPL-ID USAGE IS POINTER.
*   reserved
  05 RESD2 PIC X(20).
* :
: PROCEDURE DIVISION USING PARM1, PARM2,
  P-IND1, P-IND2,
  P-SQLSTATE, P-PROC, P-SPEC, P-DIAG,
  SQLUDF-DBINFO.
:
```

Ejemplo PL/I

La siguiente figura muestra cómo un procedimiento almacenado que está escrito en el lenguaje PL/I recibe estos parámetros.

```
*PROCESS SYSTEM(MVS);
MYMAIN: PROC(PARM1, PARM2, ...,
             P_IND1, P_IND2, ...,
             P_SQLSTATE, P_PROC, P_SPEC, P_DIAG, DBINFO)
OPTIONS(MAIN NOEXECOPS REENTRANT);

DCL PARM1 ...          /* first parameter */
DCL PARM2 ...          /* second parameter */
:
DCL P_IND1 BIN FIXED(15);/* indicator for 1st parm   */
DCL P_IND2 BIN FIXED(15);/* indicator for 2nd parm   */
:
DCL P_SQLSTATE CHAR(5); /* SQLSTATE to return to DB2 */
DCL 01 P_PROC CHAR(27) /* Qualified procedure name */
          VARYING;
DCL 01 P_SPEC CHAR(18) /* Specific stored proc */
          VARYING;
DCL 01 P_DIAG CHAR(1000) /* Diagnostic string */
          VARYING;
DCL DBINFO PTR;

DCL 01 SP_DBINFO BASED(DBINFO),
/* Dbinfo           */
  03 UDF_DBINFO_LLEN BIN FIXED(15),    /* location length   */
  03 UDF_DBINFO_LOC CHAR(128),        /* location name     */
  03 UDF_DBINFO_ALEN BIN FIXED(15),   /* auth ID length   */
  03 UDF_DBINFO_AUTH CHAR(128),       /* authorization ID */
  03 UDF_DBINFO_CCSID,               /* CCSIDs for DB2 for z/OS */
  05 R1      BIN FIXED(15), /* Reserved           */
  05 UDF_DBINFO_ASBCS BIN FIXED(15), /* ASCII SBCS CCSID */
  05 R2      BIN FIXED(15), /* Reserved           */
  05 UDF_DBINFO_ADBCS BIN FIXED(15), /* ASCII DBCS CCSID */
  05 R3      BIN FIXED(15), /* Reserved           */
  05 UDF_DBINFO_AMIXED BIN FIXED(15), /* ASCII MIXED CCSID */
  05 R4      BIN FIXED(15), /* Reserved           */
  05 UDF_DBINFO_ESBCS BIN FIXED(15), /* EBCDIC SBCS CCSID */
  05 R5      BIN FIXED(15), /* Reserved           */
  05 UDF_DBINFO_EDBCSCS BIN FIXED(15), /* EBCDIC DBCS CCSID */
  05 R6      BIN FIXED(15), /* Reserved           */
  05 UDF_DBINFO_EMIXED BIN FIXED(15), /* EBCDIC MIXED CCSID */
  05 R7      BIN FIXED(15), /* Reserved           */
  05 UDF_DBINFO_USBCS BIN FIXED(15), /* Unicode SBCS CCSID */
/*
  05 R8      BIN FIXED(15), /* Reserved           */
  05 UDF_DBINFO_UDBCS BIN FIXED(15), /* Unicode DBCS CCSID */
*/
  05 R9      BIN FIXED(15), /* Reserved           */
  05 UDF_DBINFO_UMIXED BIN FIXED(15), /* Unicode MIXED CCSID */
  05 UDF_DBINFO_ENCODE BIN FIXED(31), /* SP encode scheme */
  05 UDF_DBINFO_RESERVO CHAR(08),    /* reserved           */
/*
  03 UDF_DBINFO_SLEN BIN FIXED(15),   /* schema length     */
  03 UDF_DBINFO_SCHEMA CHAR(128),     /* schema name       */
  03 UDF_DBINFO_TLEN BIN FIXED(15),   /* table length      */
  03 UDF_DBINFO_TABLE CHAR(128),      /* table name        */
  03 UDF_DBINFO_CLEN BIN FIXED(15),   /* column length     */
  03 UDF_DBINFO_COLUMN CHAR(128),     /* column name       */
  03 UDF_DBINFO_RELVER CHAR(8),       /* DB2 release level */
  03 UDF_DBINFO_RESERVO CHAR(2),      /* reserved           */
  03 UDF_DBINFO_PLATFORM BIN FIXED(31), /* database platform */
  03 UDF_DBINFO_NUMTFCOL BIN FIXED(15), /* # of TF cols used */
  03 UDF_DBINFO_RESERV1 CHAR(26),     /* reserved           */
  03 UDF_DBINFO_TFCOLUMN PTR,         /* -> table fun col list */
  03 UDF_DBINFO_APPLID PTR,          /* -> application id */
  03 UDF_DBINFO_RESERV2 CHAR(20);    /* reserved           */
:
```

Estructura DBINFO

Utilice la estructura DBINFO para pasar información de entorno a funciones definidas por el usuario y procedimientos almacenados. Algunos campos de la estructura no se utilizan para procedimientos almacenados.

DBINFO es una estructura que contiene información como el nombre del servidor actual, el ID de autorización de tiempo de ejecución de la aplicación y la identificación de la versión y el lanzamiento del gestor de base de datos que invocó el procedimiento.

La estructura DBINFO incluye la siguiente información:

Nombre de la ubicación longitud

Un campo entero de 2 bytes sin firmar. Contiene la longitud del nombre de la ubicación en el siguiente campo.

Nombre de la ubicación

Un campo de caracteres de 128 bytes. Contiene el nombre de la ubicación a la que el invocador está conectado actualmente.

Longitud del ID de autorización

Un campo entero de 2 bytes sin firmar. Contiene la longitud del ID de autorización en el siguiente campo.

ID de autorización

Un campo de caracteres de 128 bytes. Contiene el ID de autorización de la aplicación desde la que se invoca el procedimiento almacenado, relleno a la derecha con espacios en blanco. Si este procedimiento almacenado está anidado dentro de otras rutinas (funciones definidas por el usuario o procedimientos almacenados), este valor es el ID de autorización de la aplicación que invocó la rutina de nivel más alto.

Página de códigos del subsistema

Una estructura de 48 bytes que consta de 10 campos enteros y un área reservada de ocho bytes. Estos campos proporcionan información sobre los CCSID del subsistema desde el que se invoca el procedimiento almacenado.

Longitud del calificador de tabla

Un campo entero de 2 bytes sin firmar. Este campo contiene 0.

Calificador de tabla

Un campo de caracteres de 128 bytes. Este campo no se utiliza para procedimientos almacenados.

Longitud del nombre de tabla

Un campo entero de 2 bytes sin firmar. Este campo contiene 0.

Nombre de tabla

Un campo de caracteres de 128 bytes. Este campo no se utiliza para procedimientos almacenados.

Longitud del nombre de la columna

Un campo entero de 2 bytes sin firmar. Este campo contiene 0.

Nombre de columna

Un campo de caracteres de 128 bytes. Este campo no se utiliza para procedimientos almacenados.

Información sobre el producto

Un campo de caracteres de 8 bytes que identifica el producto en el que se ejecuta el procedimiento almacenado.

El valor del identificador de producto (PRDID) es un valor de carácter de 8 bytes en formato *ppvvrrm*, donde: *ppp* es un código de producto de 3 letras; *vv* es la versión; *rr* es el release; y *m* es el nivel de modificación. En Db2 12 for z/OS, el nivel de modificación indica un rango de niveles de función:

DSN12015 para V12R1M500 o superior.

DSN12010 para V12R1M100.

Para obtener más información, consulte [Valores del identificador de producto \(PRDID\) en la guía de administración de Db2 for z/OS \(Db2 \)](#).

Área reservada

2 bytes.

Sistema operativo

Un campo entero de 4 bytes. Identifica el sistema operativo en el que se ejecuta el programa que invoca la función definida por el usuario. El valor es uno de estos:

0	Desconocido
1	OS/2
3	Windows
4	AIX
5	Windows NT
6	HP-UX
7	Solaris
8	z/OS
13	Siemens Nixdorf
15	Windows 95
16	SCO UNIX
18	Linux
19	DYNIX/ptx
24	Linux para S/390
25	Linux para IBM zSystems
26	Linux/IA64
27	Linux /PPC
28	Linux/PPC64
29	Linux/AMD64
400	iSeries

Número de entradas en la lista de columnas de la función de tabla

Un campo entero de 2 bytes sin firmar. Este campo contiene 0.

Área reservada

26 bytes.

Tabla de funciones de columna de lista de puntero

Este campo no se utiliza para procedimientos almacenados.

Identificador único de la aplicación

Este campo es un puntero a una cadena que identifica de forma única la conexión de la aplicación a Db2. La cadena se regenera en cada conexión a Db2.

La cadena es el LUWID, que consiste en un nombre de red LU totalmente calificado seguido de un punto y un número de instancia LUW. El nombre de red LU consta de un identificador de red de uno a ocho caracteres, un punto y un nombre de red LU de uno a ocho caracteres. El número de instancia LUW consta de 12 caracteres hexadecimales que identifican de forma única la unidad de trabajo.

Área reservada

20 bytes.

Paquetes para procedimientos almacenados externos

Un procedimiento almacenado externo debe tener un paquete asociado.

Como parte del proceso de creación de un procedimiento almacenado externo, usted prepara el procedimiento, lo que significa que precompila, compila, edita enlaces y vincula la aplicación. El resultado de este proceso es un paquete de Db2 . No es necesario crear un plan de Db2 e para un procedimiento externo. El procedimiento se ejecuta en el subproceso del llamador y utiliza el plan del programa cliente que lo llama.

La aplicación de llamada puede utilizar un paquete o plan de Db2 para ejecutar la instrucción CALL.

Tanto el paquete de procedimientos almacenados como el plan o paquete de la aplicación de llamada deben existir en el servidor antes de ejecutar la aplicación de llamada.

La siguiente figura muestra esta relación entre un programa cliente y un procedimiento almacenado. En la figura, el programa cliente, que estaba vinculado al paquete A, emite una instrucción CALL al programa B. El programa B es un procedimiento almacenado externo en un espacio de direcciones WLM. Este procedimiento almacenado externo se vinculó al paquete B.

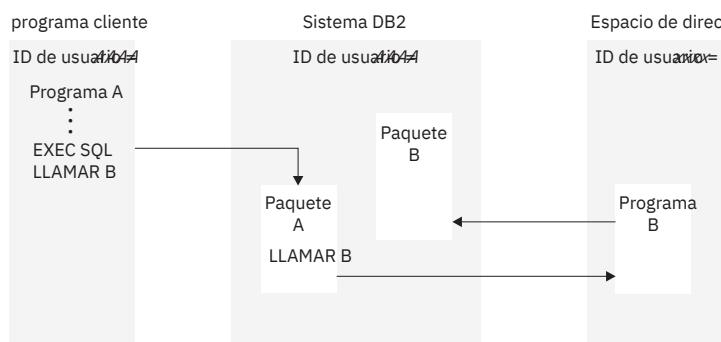


Figura 15. Entorno de tiempo de ejecución de procedimientos almacenados

Puede controlar el acceso al paquete de procedimientos almacenados especificando la opción ENABLE bind cuando vincula el paquete.

En las siguientes situaciones, el procedimiento almacenado podría utilizar más de un paquete:

- Vincula un DBRM varias veces en varias versiones del mismo paquete, todos los cuales tienen el mismo nombre de paquete pero residen en diferentes colecciones de paquetes. Su procedimiento almacenado puede cambiar de una versión a otra utilizando la instrucción SET CURRENT PACKAGESET.
- El procedimiento almacenado llama a otro programa que contiene instrucciones SQL. Este programa tiene un paquete asociado. Este paquete debe existir en la ubicación donde se define el procedimiento almacenado y en la ubicación donde se ejecutan las sentencias SQL.

Referencia relacionada

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2\)](#)

[BIND PACKAGE subcomando \(DSN\) \(Comandos de Db2\)](#)

[SET CURRENT PACKAGESET declaración \(Db2 SQL\)](#)

Acceso a otros sitios en un procedimiento externo

Los procedimientos externos pueden acceder a tablas en otras ubicaciones de Db2.

Acerca de esta tarea

Los procedimientos almacenados pueden acceder a tablas en otras ubicaciones de la base de datos (Db2) utilizando nombres de objeto de tres partes o sentencias CONNECT.

Conceptos relacionados

[Acceso a datos distribuidos utilizando nombres de tablas de tres partes](#)

Puede utilizar nombres de tabla de tres partes para acceder a datos en una ubicación remota a través del acceso DRDA.

Acceso de recursos no Db2 en el procedimiento almacenado

Las aplicaciones que se ejecutan en un espacio de direcciones de procedimientos almacenados pueden acceder a cualquier recurso que esté disponible para z/OS espacios de direcciones. Por ejemplo, pueden acceder a archivos VSAM, archivos planos, conversaciones APPC/MVS y IMS o transacciones CICS transacciones.

Acerca de esta tarea

Acceder a estos recursos desde un procedimiento almacenado puede ser útil si desea actualizar aplicaciones antiguas. Supongamos que tiene aplicaciones existentes que acceden a recursos no-Db2, pero desea utilizar aplicaciones Db2 más nuevas para acceder a los mismos datos. No es necesario que reescriba la aplicación ni que migre los datos a Db2. En su lugar, puede utilizar procedimientos almacenados para ejecutar el programa existente o acceder directamente a los datos no eDb2 es.

Cuando se ejecuta un procedimiento almacenado, este utiliza los Servicios de recuperación de datos (Resource Manager, RRS) para el control de compromisos. Cuando Db2 realiza o revierte un trabajo, Db2 coordina todas las actualizaciones que realizan otros gestores de recursos compatibles con RRS en el z/OS sistema.

Procedimiento

Para acceder a recursos no pertenecientes aDb2 en su procedimiento almacenado:

1. Considere la posibilidad de serializar el acceso a recursos no-Db2 es dentro de su aplicación.
No todos los recursos no-Db2 s pueden tolerar el acceso simultáneo de múltiples TCBs en el mismo espacio de direcciones.
2. Para acceder a CICS, utilice uno de los siguientes métodos:
 - Procedimiento almacenado DSNACICS
 - Interfaz de cola de mensajes (MQI) para la ejecución asíncrona de CICS transacciones
 - Interfaz externa CICS (EXCI) para la ejecución sincrónica de CICS transacciones
 - Comunicación avanzada entre programas (APP), utilizando la interfaz de programación de aplicaciones de comunicaciones de interfaz de programación común (CPI Communications)

Si su sistema ejecuta una versión de CICS que utiliza z/OS RRS, RRS controla el compromiso de todos los recursos z/OS RRS controla el compromiso de todos los recursos.

3. Para acceder a IMS Datos de DL/I, utilice uno de los siguientes métodos
 - Interfaz de acceso a bases de datos abiertas (ODBA)
 - Procedimientos almacenados DSNAIMS y DSNAIMS2
- Si su sistema no ejecuta una versión de IMS que utilice z/OS RRS, realice una de las siguientes acciones:
- Utilice la CICS Interfaz EXCI para ejecutar una CICS transacción de forma sincrónica. Esa CICS transacción puede, a su vez, acceder a datos de DL/I.
 - Invocar IMS transacciones de forma asíncrona utilizando el MQI.
 - Utilizar APPC a través de la interfaz de programación de aplicaciones de comunicaciones de la interfaz de programación común (CPI).

4. Determine cuál de los siguientes ID de autorización desea utilizar para acceder a los recursos noDb2 .

Tabla 47. ID de autorización para acceder a recursos no pertenecientes aDb2 desde un procedimiento almacenado

Identificación que desea utilizar para acceder a los recursos no pertenecientes aDb2	Valor SECURITY que se debe especificar en la instrucción CREATE PROCEDURE
El ID de autorización que está asociado con el espacio de direcciones de los procedimientos almacenados	SeguridadDb2
El ID de autorización bajo el cual se ejecuta la declaración CALL	SECURITY USER
El ID de autorización bajo el cual se ejecuta la declaración CREATE PROCEDURE	DEFINICIÓN DE SEGURIDAD

5. Emitir la instrucción CREATE PROCEDURE con la opción SECURITY adecuada que determinó en el paso anterior.

Resultados

Cuando se ejecuta el procedimiento almacenado, Db2 establece un RACF entorno para acceder a recursos noDb2 y utiliza el ID de autorización especificado para acceder a recursos protegidos z/OS recursos.

Tareas relacionadas

[Invocación de un procedimiento almacenado desde una aplicación](#)

Para ejecutar un procedimiento almacenado, puede llamarlo desde un programa cliente o invocarlo desde el Db2 command line processor.

[Implementación de RSS para procedimientos almacenados durante la instalación \(Db2 Installation and Migration\)](#)

[Control del acceso al procedimiento almacenado a recursos que no son de Db2 utilizando RACE \(Managing Security\)](#)

Referencia relacionada

[Procedimiento almacenado DSNACICS \(Db2 SQL\)](#)

[Procedimiento almacenado DSNAIMS \(Db2 SQL\)](#)

[Procedimiento almacenado DSNAIMS2 \(Db2 SQL\)](#)

[Instrucción CREATE PROCEDURE \(SQL - procedimiento externo\) \(obsoleta\) \(Db2 SQL\)](#)

[Configuración de APPC/MVS \(Multiplatform APPC Configuration Guide\)](#)

Información relacionada

[Db2 para procedimientos almacenados de SQL Server \(z/OS Stored Procedures: Through the CALL and Beyond \(IBM Redbooks \)](#)

[Interfaz de CICS externa \(EXCI\) \(CICS Transaction Server for z/OS\)](#)

Escribir un procedimiento externo para acceder a IMS bases de datos

IMS El soporte de acceso abierto a bases de datos (ODBA) permite que un procedimiento almacenado de Db2 se conecte a un IMS DBCTL o IMS DB/DC y emitir llamadas DL/I para acceder a IMS bases de datos.

Acerca de esta tarea

El soporte de ODBA utiliza RRS para el control de puntos de sincronización de recursos y Db2 IMS recursos. Por lo tanto, los procedimientos almacenados que utilizan ODBA solo pueden ejecutarse en espacios de direcciones de procedimientos almacenados establecidos por WLM.

Cuando escriba un procedimiento almacenado que utilice ODBA, siga las reglas para escribir un IMS programa de aplicación que emite llamadas DL/I.

IMS el trabajo que se realiza en un procedimiento almacenado está en el mismo ámbito de confirmación que el procedimiento almacenado. Al igual que con cualquier otro procedimiento almacenado, la aplicación que llama se compromete a trabajar.

Un procedimiento almacenado que utiliza ODBA debe emitir una llamada DPSB PREP para desasignar un PSB cuando todo el IMS trabajo bajo ese PSB se haya completado. La palabra clave PREP le indica IMS que traslade el trabajo en curso a un estado incierto. Cuando el trabajo está en estado indudable, IMS no requiere la activación del procesamiento de puntos de sincronización cuando se ejecuta la llamada DPSB. IMS confirma o revoca el trabajo como parte de la confirmación en dos fases de RRS cuando el llamador del procedimiento almacenado ejecuta COMMIT o ROLLBACK.

Un ejemplo de procedimiento almacenado COBOL y programa cliente demuestran el acceso a IMS datos mediante la interfaz ODBA. El código fuente del procedimiento almacenado se encuentra en el miembro DSN8EC1 y está preparado por el trabajo DSNTEJ61. El código fuente del programa de llamada se encuentra en el miembro DSN8EC1 y es preparado y ejecutado por el trabajo DSNTEJ62. Todo el código está en el conjunto de datos DSN1210.SDSNSAMP.

El procedimiento de inicio para un espacio de direcciones de procedimientos almacenados en el que se ejecutan procedimientos almacenados que utilizan ODBA debe incluir una instrucción DFSRESLB DD y un conjunto de datos adicional en la concatenación STEPLIB.

Conceptos relacionados

[Paso de instalación 21 : Configurar Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario \(Db2 Instalación y migración\)](#)

[Paso de migración 23 : Configurar Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario \(opcional\) \(Instalación y migración Db2 \)](#)

Información relacionada

[Diseño de programación de aplicaciones](#)

Escribir un procedimiento externo para devolver conjuntos de resultados a un cliente distribuido

Un procedimiento externo puede devolver varios conjuntos de resultados de consulta a un cliente distribuido si el valor de DYNAMIC RESULT SETS en la definición del procedimiento almacenado es mayor que 0.

Procedimiento

- Para cada conjunto de resultados que deseé que se devuelva, su procedimiento almacenado debe completar los siguientes pasos:
 - a) Declare un cursor con la opción CON RETORNO.
 - b) Abre el cursor.
 - c) Si el cursor se puede desplazar, asegúrese de que esté situado antes de la primera fila de la tabla de resultados.
 - d) Deje el cursor abierto.

Por ejemplo, supongamos que desea devolver un conjunto de resultados que contenga entradas para todos los empleados del departamento de D11. En primer lugar, declare un cursor que describa este subconjunto de empleados:

```
EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
  SELECT * FROM DSN8C10.EMP
  WHERE WORKDEPT='D11';
```

A continuación, abra el cursor:

```
EXEC SQL OPEN C1;
```

Db2 devuelve al cliente el conjunto de resultados y el nombre del cursor SQL para el procedimiento almacenado.

Cuando finaliza el procedimiento almacenado, Db2 devuelve las filas del conjunto de resultados de la consulta al cliente.

Db2 no devuelve conjuntos de resultados para cursos que se cierran antes de que finalice el procedimiento almacenado. El procedimiento almacenado debe ejecutar una instrucción CLOSE para cada cursor asociado a un conjunto de resultados que no deba devolverse al cliente DRDA.

- Utilice nombres de cursor significativos para los conjuntos de resultados devueltos.

El nombre del cursor que se utiliza para devolver conjuntos de resultados se pone a disposición de la aplicación cliente a través de extensiones de la sentencia DESCRIBE.

Utilice nombres de cursor que tengan sentido para la aplicación cliente DRDA, especialmente cuando el procedimiento almacenado devuelva varios conjuntos de resultados.

- Puede utilizar cualquiera de estos objetos en la instrucción SELECT que esté asociada al cursor para un conjunto de resultados:

Tablas, sinónimos, vistas, tablas temporales creadas, tablas temporales declaradas y alias definidos en el subsistema local Db2 .

- Devuelve un subconjunto de filas al cliente emitiendo sentencias FETCH con un cursor de conjunto de resultados. no devuelve las filas obtenidas al programa cliente.

Db2 no devuelve las filas obtenidas al programa cliente. Por ejemplo, si declara un cursor WITH RETURN y luego ejecuta las sentencias OPEN, FETCH y FETCH, el cliente recibe datos que comienzan con la tercera fila del conjunto de resultados. Si el cursor del conjunto de resultados se puede desplazar y se utilizan para buscar filas, es necesario colocar el cursor antes de la primera fila de la tabla de resultados después de buscar las filas y antes de que finalice el procedimiento almacenado.

- Puede utilizar una tabla temporal creada o una tabla temporal declarada para devolver conjuntos de resultados de un procedimiento almacenado.

Esta capacidad puede utilizarse para devolver datos no relacionales a un cliente DRDA. Por ejemplo, puede acceder a IMS datos de un procedimiento almacenado mediante el siguiente proceso:

- a) Utilice APPC/MVS para emitir una IMS transacción.
- b) Recibir el IMS mensaje de respuesta, que contiene datos que deben devolverse al cliente.
- c) Inserte los datos del mensaje de respuesta en una tabla temporal.
- d) Abre un cursor contra la tabla temporal. Cuando finaliza el procedimiento almacenado, las filas de la tabla temporal se devuelven al cliente.

Tareas relacionadas

[Escribir un programa para recibir los conjuntos de resultados de un procedimiento almacenado](#)

Puede escribir un programa para recibir los resultados de un procedimiento almacenado para un número fijo de conjuntos de resultados, cuando conoce el contenido, o un número variable de conjuntos de resultados, cuando no conoce el contenido.

Restricciones al invocar otros programas desde un procedimiento almacenado externo

Un procedimiento externo puede constar de más de un programa, cada uno con su propio paquete. El procedimiento almacenado puede llamar a otros programas, procedimientos almacenados o funciones definidas por el usuario. Utilice los recursos del lenguaje de programación para llamar a otros programas.

Si el procedimiento almacenado llama a otros programas que contienen sentencias de SQL, cada uno de los programas llamados debe tener un paquete de Db2. El propietario del paquete o plan que contiene la sentencia CALL debe tener la autorización EXECUTE para todos los paquetes que los otros programas utilizan.

Cuando un procedimiento almacenado llama a otro programa, Db2 determina a qué colección pertenece el paquete del programa llamado de una de las siguientes maneras:

- Si la definición del procedimiento almacenado contiene PACKAGE PATH con una lista específica de ID de colección, Db2 utiliza esos ID de colección. Si también especifica COLLID, Db2 ignora esa cláusula.
- Si la definición del procedimiento almacenado contiene COLLID *collection-id*, Db2 utiliza *collection-id*.

- Si el procedimiento almacenado ejecuta SET CURRENT PACKAGE PATH y contiene la opción NO COLLID, Db2 utiliza el registro especial CURRENT PACKAGE PATH. El paquete del programa llamado proviene de la lista de colecciones en el registro especial RUTA DEL PAQUETE ACTUAL. Por ejemplo, supongamos que la RUTA DEL PAQUETE ACTUAL contiene la lista COLL1, COLL2, COLL3, COLL4. Db2 busca el primer paquete (en el orden de la lista) que existe en estas colecciones.
- Si el procedimiento almacenado no ejecuta SET CURRENT PACKAGE PATH y en su lugar ejecuta SET CURRENT PACKAGESET, Db2 utiliza el registro especial CURRENT PACKAGESET. El paquete del programa llamado proviene de la colección que se especifica en el registro especial CURRENT PACKAGESET.
- Si se cumplen las dos condiciones siguientes, Db2 utiliza el ID de colección del paquete que contiene la instrucción SQL CALL:
 - el procedimiento almacenado no ejecuta SET CURRENT PACKAGE PATH o SET CURRENT PACKAGESET
 - la definición del procedimiento almacenado contiene la opción NO COLLID

Cuando el control vuelve del procedimiento almacenado, el valor del registro especial CURRENT PACKAGESET se restablece. Db2 restaura el valor del registro especial CURRENT PACKAGESET al valor que contenía antes de que el programa cliente ejecutara la instrucción SQL CALL.

Creación de un procedimiento almacenado externo como reentrante

El código reentrant es aquel del que dos o más procesos pueden utilizar una sola copia simultáneamente. Para mejorar el rendimiento, prepare sus procedimientos almacenados para que sean reentrantes siempre que sea posible.

Acerca de esta tarea

Los procedimientos almacenados reentrantes pueden mejorar el rendimiento por las siguientes razones:

- No es necesario cargar en el almacenamiento un procedimiento almacenado reentrant cada vez que se llama.
- Una sola copia del procedimiento almacenado puede ser compartida por múltiples tareas en el espacio de direcciones de los procedimientos almacenados. Este intercambio disminuye la cantidad de almacenamiento virtual que se utiliza para el código en el espacio de direcciones de los procedimientos almacenados.

Procedimiento

Para crear un procedimiento almacenado externo como reentrant:

1. Compile el procedimiento como reentrant y edítelo como reentrant y reutilizable.

Para obtener instrucciones sobre cómo compilar programas para que sean reentrantes, consulte la información del lenguaje de programación que esté utilizando. Para los procedimientos C y C++, puede utilizar el z/OS agrupador para producir módulos de carga reentrantes y reutilizables.

Si su procedimiento almacenado no puede ser reentrant, modifíquelo como no reentrant y no reutilizable. El atributo no reutilizable impide que varias tareas utilicen una sola copia del procedimiento almacenado al mismo tiempo.

2. Especifique STAY RESIDENT YES en la declaración CREATE PROCEDURE o ALTER PROCEDURE para el procedimiento almacenado. Esta opción hace que un procedimiento almacenado reentrant permanezca almacenado.

Un procedimiento almacenado no reentrant no debe permanecer almacenado. Por lo tanto, debe especificar STAY RESIDENT NO en la instrucción CREATE PROCEDURE o ALTER PROCEDURE para un procedimiento almacenado no reentrant. PERMANECER RESIDENTE NO es la opción predeterminada.

Conceptos relacionados

[Cómo hacer que los programas sean reentrantes \(Enterprise COBOL for z/OS Programming Guide\)](#)

Referencia relacionada

- [Opciones de compilador \(COBOL\) \(Enterprise COBOL for z/OS Programming Guide\)](#)
- [Declaración ALTER PROCEDURE \(procedimiento externo\) \(Db2 SQL\)](#)
- [Instrucción CREATE PROCEDURE \(procedimiento externo\) \(Db2 SQL\)](#)
- [Referencia de opciones de vinculador \(MVS Program Management: User's Guide and Reference\)](#)
- [Idioma restringido \(Enterprise PL/I for z/OS Compiler and Runtime Migration Guide\)](#)
- [Descripciones de opciones de tiempo de compilación \(PL/I\) \(Enterprise PL/I for z/OS Programming Guide:\)](#)
- [Reentrada \(XL C/C++ User's Guide\)](#)

Procedimientos almacenados externos como programas y subprogramas principales

Un procedimiento almacenado que se ejecuta en un espacio de direcciones establecido por WLM y utiliza Language Environment Release 1.7 o una versión posterior puede ser un programa principal o un subprograma. Un procedimiento almacenado que se ejecuta como un subprograma puede funcionar mejor porque el Entorno de lenguaje realiza menos procesamiento para él.

En general, un subprograma debe realizar las siguientes tareas adicionales que el Entorno de lenguaje lleva a cabo para un programa principal:

- Procesamiento de inicialización y limpieza
- Asignación y liberación de almacenamiento
- Cerrar todos los archivos abiertos antes de salir

Cuando codifique procedimientos almacenados como subprogramas, siga estas reglas:

- Siga las reglas de idioma para un subprograma. Por ejemplo, no puede realizar operaciones de E/S en un subprograma PL/I.
- Evite utilizar sentencias que terminen el enclave del Entorno de lenguaje cuando finalice el programa. Ejemplos de tales declaraciones son STOP o EXIT en un subprograma PL/I, o STOP RUN en un subprograma COBOL. Si el enclave finaliza cuando termina un procedimiento almacenado y el programa cliente llama a otro procedimiento almacenado que se ejecuta como un subprograma, el Entorno de lenguaje debe crear un nuevo enclave. Como resultado, se pierden las ventajas de codificar un procedimiento almacenado como un subprograma.
- En los procedimientos almacenados COBOL que se definen como PROGRAM TYPE SUB y STAY RESIDENT YES, si utiliza parámetros de procedimiento almacenado como variables de host, establezca la variable SQL-INIT-FLAG en 0. Esta variable es generada por el precompilador de PHP (Db2). Si se establece en 0, se garantiza que el SQLDA se actualiza con las direcciones actuales.

La siguiente tabla resume las características que definen un programa principal y un subprograma.

Tabla 48. Características de los programas y subprogramas principales

Idioma	Programa principal	Subprograma
Assembler	MAIN=YES se especifica en la invocación de la macro CEEENTRY.	MAIN=NO se especifica en la invocación de la macro CEEENTRY.
C	Contiene una función main(). Pásale parámetros a través de argc y argv.	Una función recuperable. Pasarle parámetros explícitamente.
COBOL	Un programa COBOL que termina con GOBACK	Un subprograma cargado dinámicamente que termina con GOBACK
PL/I	Contiene un procedimiento declarado con OPTIONS(MAIN)	Un procedimiento declarado con OPTIONS(FETCHABLE)

El siguiente código muestra un ejemplo de codificación de un procedimiento almacenado C como un subprograma.

```
/****************************************/
/* This C subprogram is a stored procedure that uses linkage */
/* convention GENERAL and receives 3 parameters.           */
/****************************************/
#pragma linkage(cfunc,fetchable)
#include <stdlib.h>
void cfunc(char p1[11],long *p2,short *p3)
{
   /****************************************/
    /* Declare variables used for SQL operations. These variables */
    /* are local to the subprogram and must be copied to and from */
    /* the parameter list for the stored procedure call.          */
   /****************************************/
EXEC SQL BEGIN DECLARE SECTION;
    char parm1[11];
    long int parm2;
    short int parm3;
EXEC SQL END DECLARE SECTION;

/****************************************/
/* Receive input parameter values into local variables.      */
/****************************************/
strcpy(parm1,p1);
parm2 = *p2;
parm3 = *p3;
/****************************************/
/* Perform operations on local variables.                    */
/****************************************/
:
/****************************************/
/* Set values to be passed back to the caller.            */
/****************************************/
strcpy(parm1,"SETBYSP");
parm2 = 100;
parm3 = 200;
/****************************************/
/* Copy values to output parameters.                      */
/****************************************/
strcpy(p1,parm1);
*p2 = parm2;
*p3 = parm3;
}
```

El siguiente código muestra un ejemplo de codificación de un procedimiento almacenado en C++ como subprograma.

```
/****************************************/
/* This C++ subprogram is a stored procedure that uses linkage */
/* convention GENERAL and receives 3 parameters.           */
/* The extern statement is required.                     */
/****************************************/
extern "C" void cppfunc(char p1[11],long *p2,short *p3);
#pragma linkage(cppfunc,fetchable)
#include <stdlib.h>
EXEC SQL INCLUDE SQLCA;
void cppfunc(char p1[11],long *p2,short *p3)
{
   /****************************************/
    /* Declare variables used for SQL operations. These variables */
    /* are local to the subprogram and must be copied to and from */
    /* the parameter list for the stored procedure call.          */
   /****************************************/
EXEC SQL BEGIN DECLARE SECTION;
    char parm1[11];
    long int parm2;
    short int parm3;
EXEC SQL END DECLARE SECTION;

/****************************************/
/* Receive input parameter values into local variables.      */
/****************************************/
strcpy(parm1,p1);
parm2 = *p2;
parm3 = *p3;
```

```

/*****+
/* Perform operations on local variables.          */
/*****+
:
/*****+
/* Set values to be passed back to the caller.    */
/*****+
strcpy(parm1,"SETBYSP");
parm2 = 100;
parm3 = 200;
/*****+
/* Copy values to output parameters.              */
/*****+
strcpy(p1,parm1);
*p2 = parm2;
*p3 = parm3;
}

```

Tipos de datos en procedimientos almacenados

Un procedimiento almacenado que esté escrito en cualquier lenguaje excepto REXX debe declarar cada parámetro que se le pase. La definición para ese procedimiento almacenado también debe contener una declaración de tipo de datos SQL compatible para cada parámetro.

Para idiomas distintos del REXX

Para todos los tipos de datos excepto LOB, ROWID, localizadores y VARCHAR (para lenguaje C), consulte las tablas enumeradas en la siguiente tabla para los tipos de datos de host que son compatibles con los tipos de datos en la definición del procedimiento almacenado. No puede tener parámetros XML en un procedimiento externo.

Para LOB, ROWID, VARCHAR y localizadores, la siguiente tabla muestra las declaraciones compatibles para el lenguaje ensamblador.

Tabla 49. Declaraciones de lenguaje ensamblador compatibles para LOB, ROWID y localizadores

Tipo de datos SQL en la definición	Declaración del ensamblador
localizador de tablas	DS FL4
localizador de BLOB	
localizador de CLOB	
localizador de DBCLOB	
BL OB (n)	<pre> If n <= 65535: var DS OFL4 var_length DS FL4 var_data DS CLn If n > 65535: var DS OFL4 var_length DS FL4 var_data DS CL65535 ORG var_data+(n-65535) </pre>
CLOB(n)	<pre> If n <= 65535: var DS OFL4 var_length DS FL4 var_data DS CLn If n > 65535: var DS OFL4 var_length DS FL4 var_data DS CL65535 ORG var_data+(n-65535) </pre>

Tabla 49. Declaraciones de lenguaje ensamblador compatibles para LOB, ROWID y localizadores (continuación)

Tipo de datos SQL en la definición	Declaración del ensamblador
DBCL OB (n)	<pre>If m (=2*n) <= 65534: var DS OFL4 var_length DS FL4 var_data DS CLm If m > 65534: var DS OFL4 var_length DS FL4 var_data DS CL65534 ORG var_data+(m-65534)</pre>
ROWID	DS HL2,CL40
VARCHAR (n)	<p>Si se especifica o implica PARAMETER VARCHAR NULTERM:</p> <pre>char data[n+1];</pre> <p>Si se especifica PARAMETER VARCHAR STRUCTURE:</p> <pre>struct {short len; char data[n]; } var;</pre>

Nota:

1. Esta fila no se aplica a VARCHAR(n) PARA DATOS DE BIT. Los DATOS BIT siempre se transmiten en una representación estructurada.

Para LOB, ROWID y localizadores, la siguiente tabla muestra declaraciones compatibles para el lenguaje C.

Tabla 50. Declaraciones compatibles con el lenguaje C para LOB, ROWID y localizadores

Tipo de datos SQL en la definición	Declaración C
localizador de tablas	unsigned long
localizador de BLOB	
localizador de CLOB	
localizador de DBCLOB	
BL OB (n)	<pre>struct {unsigned long length; char data[n]; } var;</pre>
CLOB(n)	<pre>struct {unsigned long length; char var_data[n]; } var;</pre>
DBCL OB (n)	<pre>struct {unsigned long length; sqldbchar data[n]; } var;</pre>

Tabla 50. Declaraciones compatibles con el lenguaje C para LOB, ROWID y localizadores (continuación)

Tipo de datos SQL en la definición	Declaración C
ROWID	<pre>struct {short int length; char data[40]; } var;</pre>

Para LOB, ROWID y localizadores, la siguiente tabla muestra las declaraciones compatibles para COBOL.

Tabla 51. Declaraciones COBOL compatibles para LOB, ROWID y localizadores

Tipo de datos SQL en la definición	Declaración COBOL
localizador de tablas	01 var PIC S9(9) COMP-5.
localizador de BLOB	
localizador de CLOB	
localizador de DBCLOB	
BL OB (n)	<pre>01 var. 49 var-LENGTH PIC S9(9) COMP-5. 49 var-DATA PIC X(n).</pre>
CLOB(n)	<pre>01 var. 49 var-LENGTH PIC S9(9) COMP-5. 49 var-DATA PIC X(n).</pre>
DBCL OB (n)	<pre>01 var. 49 var-LENGTH PIC S9(9) COMP-5. 49 var-DATA PIC G(n) DISPLAY-1.</pre>
ROWID	<pre>01 var. 49 var-LEN PIC S9(4) COMP-5. 49 var-DATA PIC X(40).</pre>

Para LOB, ROWID y localizadores, la siguiente tabla muestra declaraciones compatibles para PL/I.

Tabla 52. Declaraciones PL/I compatibles para LOB, ROWID y localizadores

Tipo de datos SQL en la definición	PL/I
localizador de tablas	BIN FIXED(31)
localizador de BLOB	
localizador de CLOB	
localizador de DBCLOB	

Tabla 52. Declaraciones PL/I compatibles para LOB, ROWID y localizadores (continuación)

Tipo de datos SQL en la definición	PL/I
BL OB (n)	<p>Si $n \leq 32767$:</p> <pre>01 var, 03 var_LENGTH BIN FIXED(31), 03 var_DATA CHAR(n);</pre> <p>Si $n > 32767$:</p> <pre>01 var, 02 var_LENGTH BIN FIXED(31), 02 var_DATA, 03 var_DATA1(n) CHAR(32767), 03 var_DATA2 CHAR(mod(n, 32767));</pre>
CLOB(n)	<p>Si $n \leq 32767$:</p> <pre>01 var, 03 var_LENGTH BIN FIXED(31), 03 var_DATA CHAR(n);</pre> <p>Si $n > 32767$:</p> <pre>01 var, 02 var_LENGTH BIN FIXED(31), 02 var_DATA, 03 var_DATA1(n) CHAR(32767), 03 var_DATA2 CHAR(mod(n, 32767));</pre>
DBCL OB (n)	<p>Si $n \leq 16383$:</p> <pre>01 var, 03 var_LENGTH BIN FIXED(31), 03 var_DATA GRAPHIC(n);</pre> <p>Si $n > 16383$:</p> <pre>01 var, 02 var_LENGTH BIN FIXED(31), 02 var_DATA, 03 var_DATA1(n) GRAPHIC(16383), 03 var_DATA2 GRAPHIC(mod(n, 16383));</pre>
ROWID	CHAR(40) VAR

Tablas de resultados : Cada definición de lenguaje de alto nivel para parámetros de procedimientos almacenados admite solo una instancia (un valor escalar) del parámetro. No hay soporte para parámetros de estructura, matriz o vector. Debido a esto, la instrucción SQL CALL limita la capacidad de una aplicación para devolver algunos tipos de tablas. Por ejemplo, una aplicación puede necesitar devolver una tabla que represente múltiples ocurrencias de uno o más de los parámetros pasados al

procedimiento almacenado. Dado que la instrucción SQL CALL no puede devolver más de un conjunto de parámetros, utilice una de las siguientes técnicas para devolver dicha tabla:

- Ponga los datos que devuelve la aplicación en una tabla de resultados (Db2). El programa de llamada puede recibir los datos de una de estas formas:
 - El programa de llamada puede recuperar las filas de la tabla directamente. Especifique FOR FETCH ONLY o FOR READ ONLY en la instrucción SELECT que recupera datos de la tabla. Una captura de bloqueo puede recuperar los datos necesarios de manera eficiente.
 - El procedimiento almacenado puede devolver el contenido de la tabla como un conjunto de resultados. Consulte “[Escribir un procedimiento externo para devolver conjuntos de resultados a un cliente distribuido](#)” en la página 293 y “[Escribir un programa para recibir los conjuntos de resultados de un procedimiento almacenado](#)” en la página 811 para obtener más información.
- Convertir datos tabulares a formato de cadena y devolverlos como un parámetro de cadena de caracteres al programa de llamada. El programa de llamada y el procedimiento almacenado pueden establecer una convención para interpretar el contenido de la cadena de caracteres. Por ejemplo, la instrucción SQL CALL puede pasar un parámetro de cadena de caracteres de 1920 bytes a un procedimiento almacenado, lo que permite que el procedimiento almacenado devuelva una imagen de pantalla de tipo 24x80 al programa de llamada.

Conceptos relacionados

[Compatibilidad de SQL y tipos de datos de lenguaje](#)

Los tipos de datos de variable host que se utilizan en sentencias SQL deben ser compatibles con los tipos de datos de las columnas con las que tiene intención de utilizarlos.

[Paso de instalación 21 : Configurar Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario \(Db2 Instalación y migración \)](#)

[Paso de migración 23 : Configurar Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario \(opcional\) \(Instalación y migración Db2 \)](#)

Procedimientos almacenados en REXX

Un procedimiento almacenado REXX es similar a cualquier otro procedimiento REXX y sigue las mismas reglas que los procedimientos almacenados en otros lenguajes. Un procedimiento almacenado REXX recibe parámetros de entrada, ejecuta comandos REXX, ejecuta opcionalmente sentencias SQL y devuelve como máximo un parámetro de salida. Sin embargo, hay algunas diferencias.

Un procedimiento almacenado REXX es diferente de otros procedimientos REXX en los siguientes aspectos:

- Un procedimiento almacenado REXX no debe ejecutar ninguno de los siguientes comandos DSNREXX que se utilizan para la conexión de subprocessos del subsistema de Db2 :

DIRECCIÓN DSNREXX CONECTAR
DIRECCIÓN DSNREXX DESCONECTAR
LLAMAR A SQLDBS ADJUNTAR A
LLAMAR A SQLDBS DETACH

Cuando ejecutas instrucciones SQL en tu procedimiento almacenado, Db2 establece la conexión por ti.

- Un procedimiento almacenado REXX debe ejecutarse en un espacio de direcciones de procedimientos almacenados establecido por WLM.
- Un procedimiento almacenado en lenguaje REXX se ejecuta en un entorno TSO/E REXX en segundo plano proporcionado por el servicio de entorno TSO/E IKJTSOEV.

A diferencia de otros procedimientos almacenados, no se preparan procedimientos almacenados REXX para su ejecución. Los procedimientos almacenados de REXX se ejecutan utilizando uno de los cuatro paquetes que se vinculan durante la instalación de Db2 REXX Language Support. El nivel de aislamiento actual en el que se ejecuta el procedimiento almacenado depende del paquete que utiliza Db2 cuando se ejecuta el procedimiento almacenado:

Nombre de paquete

Nivel de aislamiento

DSNREXRR

Lectura repetible (RR)

DSNREXRS

Estabilidad de lectura (RS)

DSNREXCS

estabilidad del cursor (CS)

DSNREXUR

lectura no confirmada (UR)

Este tema muestra un ejemplo de un procedimiento almacenado REXX que ejecuta comandos de la interfaz de línea de comandos (Db2). El procedimiento almacenado realiza las siguientes acciones:

- Recibe un parámetro de entrada que contiene un comando "Db2".
- Llama a la función IFI COMMAND para ejecutar el comando.
- Extrae los mensajes de resultado del comando del área de retorno IFI y coloca los mensajes en una tabla temporal creada. Cada fila de la tabla temporal contiene un número de secuencia y el texto de un mensaje.
- Abre un cursor para devolver un conjunto de resultados que contiene los mensajes de resultado del comando.
- Devuelve el contenido sin formato del área de devolución IFI en un parámetro de salida.

El siguiente ejemplo muestra la definición del procedimiento almacenado.

```
CREATE PROCEDURE COMMAND(IN CMDTEXT VARCHAR(254), OUT CMDRESULT VARCHAR(32704))
LANGUAGE REXX
EXTERNAL NAME COMMAND
NO COLLID
ASUTIME NO LIMIT
PARAMETER STYLE GENERAL
STAY RESIDENT NO
RUN OPTIONS 'TRAP(ON)'
WLM ENVIRONMENT WLMENV1
SECURITY DB2
DYNAMIC RESULT SETS 1
COMMIT ON RETURN NO;
```

El siguiente ejemplo muestra el procedimiento almacenado COMMAND que ejecuta comandos de Db2 .

```
/* REXX */
PARSE UPPER ARG CMD          /* Get the DB2 command text */
                                /* Remove enclosing quotation marks */
IF LEFT(CMD,1) = "" & RIGHT(CMD,1) = "" THEN
CMD = SUBSTR(CMD,2,LENGTH(CMD)-2)
ELSE
IF LEFT(CMD,1) = '''' & RIGHT(CMD,1) = '''' THEN
CMD = SUBSTR(CMD,2,LENGTH(CMD)-2)
COMMAND = SUBSTR("COMMAND",1,18," ")
/*****************************************/
/* Set up the IFCA, return area, and output area for the */
/* IFI COMMAND call. */
/*****************************************/
IFCA = SUBSTR('00'X,1,180,'00'X)
IFCA = OVERLAY(D2C(LENGTH(IFCA),2),IFCA,1+0)
IFCA = OVERLAY("IFCA",IFCA,4+1)
RTRNAREASIZE = 262144 /*1048572*/
RTRNAREA = D2C(RTRNAREASIZE+4,4)LEFT(' ',RTRNAREASIZE,' ')
OUTPUT = D2C(LENGTH(CMD)+4,2)||'0000'X||CMD
BUFFER = SUBSTR(" ",1,16," ")
/*****************************************/
/* Make the IFI COMMAND call. */
/*****************************************/
ADDRESS LINKPGM "DSNWLR COMMAND IFCA RTRNAREA OUTPUT"
WRC = RC
RTRN= SUBSTR(IFCA,12+1,4)
REAS= SUBSTR(IFCA,16+1,4)
TOTLEN = C2D(SUBSTR(IFCA,20+1,4))
/*****************************************/
/* Set up the host command environment for SQL calls. */
/*****************************************/
"SUBCOM DSNREXX"           /* Host cmd env available? */
```

```

IF RC THEN                                /* No--add host cmd env      */
  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')

/*****************************************************************/
/* Set up SQL statements to insert command output messages    */
/* into a temporary table.                                     */
/*****************************************************************/
SQLSTMT='INSERT INTO SYSIBM.SYSPRINT(SEQNO,TEXT) VALUES(?:,?)'
ADDRESS DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
IF SQLCODE ~= 0 THEN CALL SQLCA
ADDRESS DSNREXX "EXECSQL PREPARE S1 FROM :SQLSTMT"
IF SQLCODE ~= 0 THEN CALL SQLCA
  /*****************************************************************/
  /* Extract messages from the return area and insert them into */
  /* the temporary table.                                         */
  /*****************************************************************/
SEQNO = 0
OFFSET = 4+1
DO WHILE ( OFFSET < TOTLEN )
  LEN = C2D(SUBSTR(RTRNAREA,OFFSET,2))
  SEQNO = SEQNO + 1
  TEXT = SUBSTR(RTRNAREA,OFFSET+4,LEN-4-1)
  ADDRESS DSNREXX "EXECSQL EXECUTE S1 USING :SEQNO,:TEXT"
  IF SQLCODE ~= 0 THEN CALL SQLCA
  OFFSET = OFFSET + LEN
END
  /*****************************************************************/
  /* Set up a cursor for a result set that contains the command */
  /* output messages from the temporary table.                  */
  /*****************************************************************/
SQLSTMT='SELECT SEQNO,TEXT FROM SYSIBM.SYSPRINT ORDER BY SEQNO'
ADDRESS DSNREXX "EXECSQL DECLARE C2 CURSOR FOR S2"
IF SQLCODE ~= 0 THEN CALL SQLCA
ADDRESS DSNREXX "EXECSQL PREPARE S2 FROM :SQLSTMT"
IF SQLCODE ~= 0 THEN CALL SQLCA
  /*****************************************************************/
  /* Open the cursor to return the message output result set to */
  /* the caller.                                                 */
  /*****************************************************************/
ADDRESS DSNREXX "EXECSQL OPEN C2"
IF SQLCODE ~= 0 THEN CALL SQLCA
S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX') /* REMOVE CMD ENV */
EXIT SUBSTR(RTRNAREA,1,TOTLEN+4)

/*****************************************************************/
/* Routine to display the SQLCA                           */
/*****************************************************************/
SQLCA:
SAY 'SQLCODE ='SQLCODE
SAY 'SQLERRMC ='SQLERRMC
SAY 'SQLERRP ='SQLERRP
SAY 'SQLERRD ='SQLERRD.1',',
  || SQLERRD.2',',
  || SQLERRD.3',',
  || SQLERRD.4',',
  || SQLERRD.5',',
  || SQLERRD.6
SAY 'SQLWARN ='SQLWARN.0',',
  || SQLWARN.1',',
  || SQLWARN.2',',
  || SQLWARN.3',',
  || SQLWARN.4',',
  || SQLWARN.5',',
  || SQLWARN.6',',
  || SQLWARN.7',',
  || SQLWARN.8',',
  || SQLWARN.9',',
  || SQLWARN.10
SAY 'SQLSTATE='SQLSTATE
SAY 'SQLCODE ='SQLCODE
EXIT 'SQLERRMC ='SQLERRMC',',
  || 'SQLERRP ='SQLERRP',',
  || 'SQLERRD ='SQLERRD.1',',
    || SQLERRD.2',',
    || SQLERRD.3',',
    || SQLERRD.4',',
    || SQLERRD.5',',
    || SQLERRD.6',',
  || 'SQLWARN ='SQLWARN.0',',
    || SQLWARN.1','

```

```

|| SQLWARN.2' , ,
|| SQLWARN.3' , ,
|| SQLWARN.4' , ,
|| SQLWARN.5' , ,
|| SQLWARN.6' , ,
|| SQLWARN.7' , ,
|| SQLWARN.8' , ,
|| SQLWARN.9' , ,
|| SQLWARN.10' , ,
|| 'SQLSTATE='SQLSTATE';

```

Referencia relacionada

[Invocación de un procedimiento almacenado desde un procedimiento REXX](#)

El formato de los parámetros que se pasan en la instrucción CALL en un procedimiento REXX debe ser compatible con los tipos de datos de los parámetros en la instrucción CREATE PROCEDURE.

[Servicios TSO/E disponibles en IKJTSOEV \(TSO/E Programming Services\)](#)

Modificación de una definición de procedimiento almacenado externo

Puede modificar la definición de un procedimiento almacenado externo o el código fuente del procedimiento almacenado. En cualquier caso, debe preparar de nuevo el procedimiento almacenado.

Procedimiento

Para modificar una definición de procedimiento almacenado externo:

1. Emita uno de los siguientes:

- [FL 507](#) con la cláusula OR REPLACE y la cláusula SPECIFIC en los siguientes casos:
 - Cuando la lista de parámetros del procedimiento existente incluye un parámetro de tabla.
 - Cuando la instrucción CREATE especifica cambios en la lista de parámetros distintos de los nombres de los parámetros.
- La instrucción ALTER PROCEDURE con las opciones adecuadas.

Esta nueva definición sustituye a la definición existente.

2. Vuelva a preparar el procedimiento almacenado externo, tal como lo hizo cuando lo creó originalmente.

Ejemplo

Supongamos que un procedimiento almacenado C existente se definió con la siguiente instrucción:

```

CREATE PROCEDURE B(IN V1 INTEGER, OUT V2 CHAR(9))
LANGUAGE C
DETERMINISTIC
NO SQL
EXTERNAL NAME SUMMOD
COLLID SUMCOLL
ASUTIME LIMIT 900
PARAMETER STYLE GENERAL WITH NULLS
STAY RESIDENT NO
RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
WLM ENVIRONMENT PAYROLL
PROGRAM TYPE MAIN
SECURITY DB2
DYNAMIC RESULT SETS 10
COMMIT ON RETURN NO;

```

Suponga que necesita realizar los siguientes cambios en la definición del procedimiento almacenado:

- El procedimiento almacenado selecciona datos de tablas de datos de usuario (Db2), pero no modifica datos de datos de usuario (Db2).
- Los parámetros pueden tener valores nulos, y el procedimiento almacenado puede devolver una cadena de diagnóstico.
- El tiempo que se ejecuta el procedimiento almacenado es ilimitado.

- Si el procedimiento almacenado es llamado por otro procedimiento almacenado o una función definida por el usuario, el procedimiento almacenado utiliza el entorno WLM del llamante.

Cualquiera de las siguientes declaraciones puede realizar estos cambios:

```
CREATE OR REPLACE PROCEDURE B(IN V1 INTEGER, OUT V2 CHAR(9))
LANGUAGE C
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME SUMMOD
COLLID SUMCOLL
ASUTIME NO LIMIT
PARAMETER STYLE SQL
STAY RESIDENT NO
RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
WLM ENVIRONMENT (PAYROLL,*)
PROGRAM TYPE MAIN
SECURITY DB2
DYNAMIC RESULT SETS 10
COMMIT ON RETURN NO;
```

```
ALTER PROCEDURE B
READS SQL DATA
ASUTIME NO LIMIT
PARAMETER STYLE SQL
WLM ENVIRONMENT (PAYROLL,*) ;
```

Tareas relacionadas

[Creación de procedimientos almacenados externos](#)

Un *procedimiento almacenado externo* es un procedimiento que se escribe en un lenguaje de host y que puede contener sentencias SQL. El código fuente de los procedimientos externos es distinto de la definición.

[Referencia relacionada](#)

[Declaración ALTER PROCEDURE \(procedimiento externo\) \(Db2 SQL\)](#)

Creación de procedimientos SQL externos (en desuso)

Un *procedimiento SQL externo* es un procedimiento cuyo cuerpo se ha escrito en SQL en su totalidad. El cuerpo está escrito en el lenguaje de procedimientos de SQL (SQL PL). Sin embargo, los procedimientos SQL externo se crean, implementan y ejecutan como cualquier otro procedimiento almacenado externo. Todos los procedimientos SQL creados antes de DB2 9 son procedimientos SQL externos.

Antes de empezar

Función en desuso: Los procedimientos de SQL externo están en desuso y no están totalmente soportados como procedimientos SQL nativos. Para obtener mejores resultados, cree procedimientos de SQL nativo en su lugar. Para obtener más información, consulte [“Creación de procedimientos SQL nativos” en la página 244](#) y [“Migración de un procedimiento de SQL externo a un procedimiento de SQL nativo” en la página 307.](#)

Antes de crear un procedimiento SQL externo, [configure Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario durante la instalación](#) o [configure Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario durante la migración](#).

Si planea utilizar el depurador de procedimientos almacenados Db2 o el Unified Debugger, no utilice JCL. Utilice DSNTPSMP en su lugar.

Si planea utilizar DSNTPSMP, debe [configurar el soporte para procedimientos SQL externos](#).

Procedimiento

Para crear un procedimiento SQL externo:

1. Utilice uno de los siguientes métodos para crear el procedimiento SQL externo:

- IBM Data Studio. Consulte Desarrollo de rutinas de base de datos (IBM Data Studio, IBM Optim Database Administrator, IBM infoSphere Data Architect, IBM Optim Development Studio).
- Usar JCL
- Utilizar el procesador de procedimientos SQL de Microsoft (Db2 for z/OS , DSNTPSMP)

Los métodos anteriores que utiliza para crear un procedimiento SQL externo realizan las siguientes acciones:

- Convertir las instrucciones de origen del procedimiento SQL externo en un programa en lenguaje C utilizando el precompilador Db2
- Cree un módulo de carga ejecutable y un paquete de instalación (Db2) a partir del programa en lenguaje C.
- Defina el procedimiento SQL externo a Db2 emitiendo una instrucción CREATE PROCEDURE de forma estática o dinámica.

2. Autorizar a los usuarios adecuados a utilizar el procedimiento almacenado emitiendo la sentencia GRANT EXECUTE.

Ejemplo

Para ver ejemplos de cómo preparar y ejecutar procedimientos SQL externos, consulte “[Programas de ejemplo para ayudar a preparar y ejecutar procedimientos SQL externos](#)” en la página 323.

Conceptos relacionados

[Procedimientos de SQL](#)

Un procedimiento de SQL es un procedimiento almacenado que solo contiene sentencias SQL.

Tareas relacionadas

[Implementación de procedimientos almacenados de Db2 \(Db2 Administration Guide\)](#)

Referencia relacionada

[Instrucción CREATE PROCEDURE \(SQL - procedimiento externo\) \(obsoleta\) \(Db2 SQL\)](#)

[Declaración GRANT \(privilegios de función o procedimiento\) \(Db2 SQL\)](#)

Migración de un procedimiento de SQL externo a un procedimiento de SQL nativo

Puede migrar un procedimiento SQL externo existente, que está en desuso, a un procedimiento SQL nativo descartando el procedimiento existente y creándolo de nuevo como un procedimiento SQL nativo. Los procedimientos de SQL nativo son más compatibles y más fáciles de mantener y normalmente funcionan mejor que los procedimientos SQL externos, que están en desuso.

Antes de empezar

Si creó el procedimiento SQL externo en una versión anterior de Db2, tenga en cuenta las incompatibilidades de la versión para las aplicaciones que utilizan procedimientos almacenados, examine el código fuente de su procedimiento SQL externo y realice los ajustes necesarios. Consulte [Incompatibilidades de SQL y aplicaciones entre releases \(Novedades de DB2 para z/OS\)](#).

Acerca de esta tarea

Un *procedimiento de SQL nativo* es un procedimiento cuyo cuerpo se ha escrito en SQL en su totalidad. El cuerpo está escrito en el lenguaje de procedimientos de SQL (SQL PL). Un procedimiento de SQL nativo se crea emitiendo una única sentencia SQL, CREATE PROCEDURE. Los procedimientos de SQL nativos no necesitan ninguna otra preparación de programa, como la precompilación, la compilación o la edición de enlaces del código fuente. Los procedimientos de SQL nativos se ejecutan como sentencias de SQL que se han vinculado en un paquete de Db2. Los procedimientos de SQL nativo no tienen un programa de aplicación externo asociado. Los procedimientos de SQL nativo son más compatibles y más fáciles de mantener y normalmente funcionan mejor que los procedimientos SQL externos, que están en desuso.

Un *procedimiento SQL externo* es un procedimiento cuyo cuerpo se ha escrito en SQL en su totalidad. El cuerpo está escrito en el lenguaje de procedimientos de SQL (SQL PL). Sin embargo, los procedimientos SQL externo se crean, implementan y ejecutan como cualquier otro procedimiento almacenado externo.

Procedimiento

Para migrar un procedimiento SQL externo a un procedimiento SQL nativo, siga estos pasos:

1. Busque y guarde las instrucciones CREATE PROCEDURE y GRANT EXECUTE existentes para el procedimiento SQL externo existente.
2. Elimine el procedimiento SQL externo existente mediante la instrucción DROP PROCEDURE.
3. Vuelva a crear el procedimiento como un procedimiento SQL nativo utilizando la misma instrucción CREATE PROCEDURE que utilizó para crear originalmente el procedimiento, con los dos cambios siguientes:
 - Si el procedimiento se definió con las opciones CERCADO o EXTERNO, elimine estas palabras clave.
 - Elimine la palabra clave WLM ENVIRONMENT o añada la cláusula FOR DEBUG MODE.
 - Si el cuerpo del procedimiento contiene sentencias con nombres no calificados que podrían referirse a una columna o a una variable o parámetro SQL, califique estos nombres. De lo contrario, es posible que tengas que cambiar el extracto.

Db2 resuelve estos nombres de forma diferente dependiendo de si el procedimiento es un procedimiento SQL externo o un procedimiento SQL nativo. Para los procedimientos SQL externos, Db2 trata primero el nombre como una variable o parámetro si existe uno con ese nombre. Para los procedimientos SQL nativos, Db2 trata primero el nombre como una columna si existe una columna con ese nombre. Por ejemplo, considere la sentencia siguiente:

```
CREATE PROCEDURE P1 (INOUT C1 INT) ... SELECT C1 INTO xx FROM T1
```

En el ejemplo anterior, si P1 es un procedimiento SQL externo, C1 es un parámetro. Para los procedimientos SQL nativos, C1 es una columna en la tabla T1. Si no existe dicha columna, C1 es un parámetro.

4. Emite las mismas sentencias GRANT EXECUTE que utilizó originalmente para conceder privilegios para este procedimiento almacenado.
5. Aumentar el valor del parámetro TIME en la declaración de trabajo para aplicaciones que llaman a procedimientos almacenados.

Importante: Este cambio es necesario porque el tiempo para los procedimientos almacenados externos de SQL se carga en el espacio de direcciones WLM, mientras que el tiempo para los procedimientos almacenados nativos de SQL se carga en el espacio de direcciones de la tarea.

6. Pruebe su nuevo procedimiento SQL nativo.

Tareas relacionadas

[Uso de la herramienta Db2 precompiler para ayudarle a convertir un procedimiento SQL externo en un procedimiento SQL nativo](#)

El Db2 precompiler puede ser útil cuando se considera la conversión de un procedimiento SQL externo a un procedimiento SQL nativo.

[Creación de procedimientos SQL nativos](#)

Un *procedimiento de SQL nativo* es un procedimiento cuyo cuerpo se ha escrito en SQL en su totalidad y que se crea emitiendo una única sentencia SQL, CREATE PROCEDURE.

Referencia relacionada

[Instrucción CREATE PROCEDURE \(SQL - procedimiento nativo\) \(Db2 SQL\)](#)

[Declaración GRANT \(privilegios de función o procedimiento\) \(Db2 SQL\)](#)

[DROP declaración \(Db2 SQL\)](#)

Uso de la herramienta Db2 precompiler para ayudarle a convertir un procedimiento SQL externo en un procedimiento SQL nativo

El Db2 precompiler puede ser útil cuando se considera la conversión de un procedimiento SQL externo a un procedimiento SQL nativo.

Acerca de esta tarea

Utilice el Db2 precompilador para inspeccionar el origen del procedimiento SQL desde una perspectiva nativa de SQL PL. Se genera una lista que ayuda a aislar problemas e incompatibilidades entre la codificación de procedimientos SQL externos y nativos. Los cambios de origen se pueden hacer antes de hacer cualquier cambio en Db2.

Procedimiento

Para inspeccionar la calidad de la codificación de origen nativa de SQL PL utilizando el precompilador de Db2 :

1. Copie el código fuente SQL PL original en un conjunto de datos de FB80. Cambie el formato del código fuente según sea necesario para que quepa dentro de los márgenes del precompilador.
2. Precompilar la fuente SQL PL ejecutando el programa DSNHPSM con la opción HOST(SQLPL).
3. Inspeccionar el listado producido (SYSPRINT). Preste atención a los mensajes de error y advertencia.
4. Modifique la fuente SQL PL para solucionar los problemas de codificación que se identifican mediante mensajes en el listado.
5. Repita los pasos “1” en la página 309 - “4” en la página 309 hasta que se resuelvan todos los mensajes de error y advertencia. A continuación, dirija los mensajes informativos según sea necesario.
6. Copie el archivo fuente SQL PL modificado de nuevo a su formato fuente original, reformateándolo según sea necesario.

Resultados

El ejemplo JCL DSNTEJ67 muestra este proceso para un procedimiento SQL externo que se produjo utilizando el procesador de procedimientos SQL de Db2 , DSNTPSMP.

Tareas relacionadas

[Migración de un procedimiento de SQL externo a un procedimiento de SQL nativo](#)

Puede migrar un procedimiento SQL externo existente, que está en desuso, a un procedimiento SQL nativo descartando el procedimiento existente y creándolo de nuevo como un procedimiento SQL nativo. Los procedimientos de SQL nativo son más compatibles y más fáciles de mantener y normalmente funcionan mejor que los procedimientos SQL externos, que están en desuso.

Referencia relacionada

[Programas de ejemplo para ayudar a preparar y ejecutar procedimientos SQL externos](#)

Db2 proporciona trabajos de ejemplo para ayudarle a preparar y ejecutar procedimientos SQL externos. Todos los ejemplos están en el conjunto de datos de DSN1210.SDSNSAMP. Antes de poder ejecutar los ejemplos, debe personalizarlos para la instalación.

Creación de un procedimiento SQL externo utilizando DSNTPSMP

El procesador de procedimiento SQL, DSNTPSMP, es uno de los distintos métodos que puede utilizar para crear y preparar un procedimiento SQL externo. DSNTPSMP es un procedimiento almacenado REXX que puede invocar desde el programa de aplicación.

Antes de empezar

Función en desuso: Los procedimientos de SQL externo están en desuso y no están totalmente soportados como procedimientos SQL nativos. Para obtener mejores resultados, cree procedimientos de SQL nativo en su lugar. Para obtener más información, consulte “[Creación de procedimientos SQL nativos](#)” en la página 244 y “[Migración de un procedimiento de SQL externo a un procedimiento de SQL nativo](#)” en la página 307.

Configurar soporte para procedimientos SQL externos. Para obtener más información, consulte [Configuración del soporte para procedimientos de SQL externo \(Db2 Installation and Migration\)](#).

Asegúrese también de que dispone de las autorizaciones necesarias, tal y como se indica en la siguiente tabla, para invocar DSNTPSMP.

Tabla 53. Autorizaciones necesarias para invocar DSNTPSMP

Autorización requerida	Sintaxis asociada para la autorización
Procedimiento privilegiado para ejecutar programas de aplicación que invocan el procedimiento almacenado.	EXECUTE ON PROCEDURE SYSPROC.DSNTPSMP
Privilegio de colección para usar BIND para crear paquetes en la colección especificada. Puede utilizar un asterisco (*) como identificador de una colección.	CREAR EN LA COLECCIÓN <i>id-de-la-colección</i>
Privilegio de paquete para usar BIND o REBIND para encuadrinar paquetes en la colección especificada.	ENVÍO EN PAQUETE <i>collection -id.*</i>
Privilegio del sistema para utilizar BIND con la opción ADD para crear paquetes y planes.	BINDADD
Privilegio de esquema para crear, modificar o eliminar procedimientos almacenados en el esquema especificado. El ID de autorización de BUILDDOWNER debe tener el privilegio CREATEIN en el esquema. Puede utilizar un asterisco (*) como identificador de un esquema.	CREATEIN, ALTERIN, DROPIN ON SCHEMA <i>nombre-esquema</i>
Privilegios de tabla para seleccionar o eliminar de, insertar en o actualizar las tablas de catálogo especificadas.	SELECCIONAR EN LA TABLA SYSIBM.SYSROUTINES SELECCIONAR EN LA TABLA SYSIBM.SYSPARMS SELECCIONAR, INSERTAR, ACTUALIZAR, ELIMINAR EN LA TABLA SYSIBM.SYSROUTINES_SRC SELECCIONAR, INSERTAR, ACTUALIZAR, ELIMINAR EN LA TABLA SYSIBM.SYSROUTINES_OPTS TODO EN LA MESA SYSIBM.SYSPSMOUT
Cualquier privilegio que se requiera para las sentencias SQL y que esté contenido en el cuerpo del procedimiento SQL. Estos privilegios deben estar asociados con el <i>ID de autorización</i> del PROPIETARIO que se especifica en sus opciones de enlace. El propietario predeterminado es el usuario que invoca DSNTPSMP.	La sintaxis varía en función del cuerpo del procedimiento SQL

Procedimiento

Para crear un procedimiento SQL externo mediante DSNTPSMP:

- Escribir un programa de aplicación que llame a DSNTPSMP. Incluya los siguientes elementos en su programa:

- Una variable de host CLOB que contiene una instrucción CREATE PROCEDURE para el procedimiento SQL externo. Esta declaración debe incluir la palabra clave FENCED o la palabra clave EXTERNAL, y el cuerpo del procedimiento, que está escrito en SQL.

De forma alternativa, en lugar de definir una variable de host para la sentencia CREATE PROCEDURE, puede almacenar la sentencia en un miembro del conjunto de datos.

- Una instrucción SQL CALL con la función BUILD. La sentencia CALL debe utilizar la sintaxis adecuada para invocar DSNTPSMP.

Pase el origen del procedimiento SQL a DSNTPSMP como uno de los siguientes parámetros de entrada:

Fuente de procedimiento SQL

Utilice este parámetro si ha definido una variable de host en su aplicación para contener la instrucción CREATE PROCEDURE.

nombre-del-conjunto-de-datos-de-origen

Utilice este parámetro si ha almacenado la instrucción CREATE PROCEDURE en un conjunto de datos.

- En función del valor de retorno de la instrucción CALL, emita una instrucción SQL COMMIT o una instrucción ROLLBACK. Si el valor devuelto es 0 o 4, emite una sentencia COMMIT. De lo contrario, emita una declaración de ROLLBACK.

Debe procesar el conjunto de resultados antes de emitir la instrucción COMMIT o ROLLBACK.

Una solicitud QUERYLEVEL debe ir seguida de la instrucción COMMIT.

2. Precompilar, compilar y editar el enlace del programa de aplicación.
3. Envuelva un paquete para el programa de aplicación.
4. Ejecute el programa de aplicación.

Conceptos relacionados

Cuerpo de un procedimiento SQL

El cuerpo de un procedimiento SQL contiene una o más sentencias SQL. En el cuerpo del procedimiento SQL, también puede declarar y utilizar variables, condiciones, códigos de retorno, sentencias, cursor y manejadores.

Referencia relacionada

[Instrucción CREATE PROCEDURE \(SQL - procedimiento externo\) \(obsoleta\) \(Db2 SQL\)](#)

Procesador de procedimiento SQL de Db2 for z/OS (DSNTPSMP)

El procesador de procedimientos SQL, DSNTPSMP, es un procedimiento almacenado REXX que puede utilizar para preparar un procedimiento SQL externo para su ejecución.

También puede utilizar DSNTPSMP para realizar pasos seleccionados en el proceso de preparación o eliminar un procedimiento SQL externo existente. DSNTPSMP es el único método de preparación que permite depurar procedimientos SQL externos con el depurador SQL o el depurador de procedimientos SQL (Unified Debugger).

DSNTPSMP requiere que el CCSID EBCDIC de su sistema también sea compatible con el compilador C. El uso de un CCSID incompatible da lugar a errores en tiempo de compilación. Algunos ejemplos de CCSID incompatibles son 290, 930, 1026 y 1155. Si el CCSID EBCDIC de su sistema no es compatible, no lo cambie sin más. Póngase en contacto con IBM Support para obtener ayuda.

Ejemplo de procedimiento de inicio para un espacio de direcciones WLM para DSNTPSMP

Debe ejecutar DSNTPSMP en un espacio de direcciones de procedimientos almacenados establecido por WLM. Solo debe ejecutar DSNTPSMP en ese espacio de direcciones, y debe limitar el espacio de direcciones para ejecutar solo una tarea a la vez.

Este ejemplo muestra cómo configurar un espacio de direcciones WLM para DSNTPSMP.

Recomendación: Utilice el entorno principal de WLM DSNWLM_REXX. Job DSNTIJMV crea un procedimiento de espacio de direcciones llamado DSNWLMR para este entorno.

El siguiente ejemplo muestra un JCL de muestra para un procedimiento de inicio para el espacio de direcciones en el que se ejecuta DSNTPSMP.

```
//DSNWLMR  PROC DB2SSN=DSN,NUMTCB=1,APPLENV=DSNWLM_REXX
//*
//WLMTSPMP EXEC PGM=DSNX9WLM,TIME=1440,
//                      PARM='&DB2SSN,&NUMTCB,&APPLENV',
```

1

2

```

//          REGION=0M,DYNAMNBR=10
//STEPLIB  DD DISP=SHR,DSN=DSN1010.SDSNEXIT          3
//          DD DISP=SHR,DSN=DSN1010.SDSNLOAD
//          DD DISP=SHR,DSN=CBC.SCCNCMP
//          DD DISP=SHR,DSN=CEE.SCEERUN
//          DD DISP=SHR,DSN=DSN1010.DBRLMLIB.DATA      3
//SYSEXEC  DD DISP=SHR,DSN=DSN1010.SDSNCLST        4
//SYSTSPRT DD SYSOUT=A
//CEEDUMP   DD SYSOUT=A
//SYSABEND  DD DUMMY
///*
//SQLDBRM  DD DISP=SHR,DSN=DSN1010.DBRLMLIB.DATA    5
//SQLCSRC  DD DISP=SHR,DSN=DSN1010.SRCLIB.DATA      6
//SQLMOD   DD DISP=SHR,DSN=DSN1010.RUNLIB.LOAD       7
//SQLIBC   DD DISP=SHR,DSN=CEE.SCEEH.H             8
//          DD DISP=SHR,DSN=CEE.SCEEH.SYS.H
//SQLLIBL  DD DISP=SHR,DSN=CEE.SCEELKED            9
//          DD DISP=SHR,DSN=DSN1010.SDSNLOAD
//SYSMSGS  DD DISP=SHR,DSN=CEE.SCEEMSGP(EDCPMSG)    10
///*
//** DSNTPSMP Configuration File - CFGTPSMP (optional) 11
//**           A site-provided sequential data set or member, used to
//**           define customized operation of DSNTPSMP in this APPLENV
//**
//** CFGTPSMP DD  DISP=SHR,DSN=
//*
//SQLSRC   DD UNIT=SYSALLDA,SPACE=(23440,(20,20)),      12
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLPRINT  DD UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLTERM   DD UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLOUT    DD UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLCPRT   DD UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLUT1    DD UNIT=SYSALLDA,SPACE=(23440,(20,20)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLUT2    DD UNIT=SYSALLDA,SPACE=(23440,(20,20)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLCIN    DD UNIT=SYSALLDA,SPACE=(32000,(20,20))
//SQLLIN    DD UNIT=SYSALLDA,SPACE=(3200,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSMOD   DD UNIT=SYSALLDA,SPACE=(23440,(20,20)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLDUMMY  DD DUMMY

```

Notas:

1

APPLENV especifica el entorno de aplicación en el que se ejecuta DSNTPSMP. Para garantizar que DSNTPSMP siempre utilice los conjuntos de datos y parámetros correctos para preparar cada procedimiento SQL externo, puede configurar diferentes entornos de aplicación para preparar procedimientos almacenados con diferentes requisitos de preparación de programas. Por ejemplo, si todas las aplicaciones de nóminas utilizan el mismo conjunto de datos durante la preparación del programa, podría configurar un entorno de aplicación llamado NÓMINA para preparar únicamente aplicaciones de nóminas. El procedimiento de inicio de PAYROLL indicaría los conjuntos de datos que se utilizan para las aplicaciones de nómina.

DB2SSN especifica el nombre del subsistema de Db2 .

NUMTCB especifica el número de programas que pueden ejecutarse simultáneamente en el espacio de direcciones. Siempre debe establecer NUMTCB en 1 para garantizar que las ejecuciones de DSNTPSMP se produzcan en serie.

2

WLMTPSMP especifica el espacio de direcciones en el que se ejecuta DSNTPSMP.

DYNAMNBR reserva espacio para la asignación dinámica de conjuntos de datos durante el proceso de preparación del procedimiento SQL.

- 3** STEPLIB especifica las bibliotecas de carga de e Db2 , la z/OS Biblioteca del compilador C/C++ y la biblioteca de tiempo de ejecución del entorno de lenguaje que DSNTPSMP utiliza cuando se ejecuta. Al menos una biblioteca no debe estar autorizada por APF.
- 4** SYSEXEC especifica la biblioteca que contiene el REXX exec DSNTPSMP.
- 5** SQLDBRM es un conjunto de datos de salida que especifica la biblioteca en la que DSNTPSMP coloca el DBRM que genera cuando precompila su procedimiento SQL externo.
- 6** SQLCSRC es un conjunto de datos de salida que especifica la biblioteca en la que DSNTPSMP coloca el código fuente C que genera a partir del código fuente del procedimiento SQL externo. Este conjunto de datos debe tener una longitud de registro lógica de 80.
- 7** SQLMOD es un conjunto de datos de salida que especifica la biblioteca en la que DSNTPSMP coloca el módulo de carga que genera cuando compila y edita mediante enlace su procedimiento SQL externo.
- 8** SQLLIBC especifica la biblioteca que contiene los archivos de cabecera estándar de C. Esta biblioteca se utiliza durante la compilación del programa C generado.
- 9** SQLLIBL especifica las siguientes bibliotecas, que DSNTPSMP utiliza cuando edita el enlace del procedimiento SQL externo :
- Idioma Entorno Enlace - Editar biblioteca
 - Db2 biblioteca de carga
- 10** SYSMSGs especifica la biblioteca que contiene mensajes que son utilizados por la utilidad C prelink-edit.
- 11** CFGTPSMP especifica un conjunto de datos opcional que puede utilizar para personalizar DSNTPSMP, incluida la especificación del nivel del compilador. Para obtener detalles sobre todas las opciones que puede configurar en este archivo y cómo configurarlas, consulte los comentarios de DSNTPSMP CLIST.
- 12** Las siguientes declaraciones DD describen conjuntos de datos de archivos de trabajo que utiliza DSNTPSMP.

Tareas relacionadas

[Conversión de funciones de MQ basadas en AMI a funciones de MQ basadas en MQI \(Db2 Installation and Migration\)](#)

Sintaxis de la instrucción CALL para invocar DSNTPSMP

Puede invocar el procesador de procedimientos SQL, DSNTPSMP, desde un programa de aplicación mediante una instrucción SQL CALL. DSNTPSMP prepara un procedimiento SQL externo.

Los siguientes diagramas muestran la sintaxis de invocar DSNTPSMP a través de la instrucción SQL CALL:

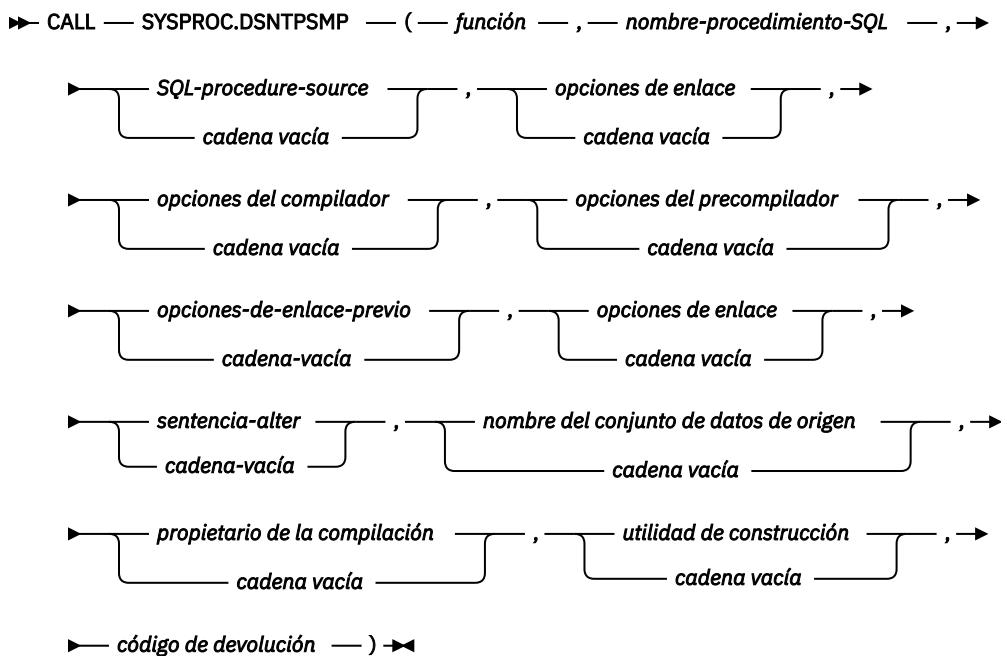


Figura 16. Sintaxis DSNTPSMP

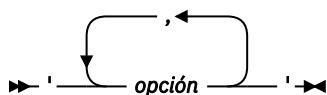


Figura 17. LLAMAR DSNTPSMP opciones de enlace, opciones de compilador, opciones de precompilador, opciones de preenlace, opciones de enlace

Nota: Se debe especificar:

- Los parámetros DSNTPSMP en el orden indicado
- La cadena vacía si un parámetro opcional no es necesario para la función
- Las opciones en el orden: enlazar, compilador, precompilador, preenlazar y enlazar

Los parámetros DSNTPSMP son:

función

Un parámetro de entrada VARCHAR(20) que identifica la tarea que desea que DSNTPSMP realice. Las tareas son:

BUILD

Crea los siguientes objetos para un procedimiento SQL externo :

- Un DBRM, en el conjunto de datos al que apunta el nombre DD SQLDBRM
- Un módulo de carga, en el conjunto de datos al que apunta el nombre DD SQLLMOD
- El código fuente en lenguaje C para el procedimiento SQL externo, en el conjunto de datos al que apunta el nombre DD SQLCSRC
- El paquete de procedimientos almacenados
- La definición del procedimiento almacenado

Los siguientes parámetros de entrada son necesarios para la función BUILD:

Nombre del procedimiento SQL
Procedimiento-SQL-fuente o nombre-conjunto-datos-fuente

Si elige la función BUILD y ya existe un procedimiento SQL externo con el nombre *nombre-del-procedimiento-SQL*, DSNTPSMP emite un mensaje de error y se cierra.

BUILD_DEBUG

Crea los siguientes objetos para un procedimiento SQL externo e incluye la preparación necesaria para depurar el procedimiento SQL externo con el depurador SQL y el depurador de procedimientos (Unified Debugger):

- Un DBRM, en el conjunto de datos al que apunta el nombre DD SQLDBRM
- Un módulo de carga, en el conjunto de datos al que apunta el nombre DD SQLLMOD
- El código fuente en lenguaje C para el procedimiento SQL externo, en el conjunto de datos al que apunta el nombre DD SQLCSRC
- El paquete de procedimientos almacenados
- La definición del procedimiento almacenado

Los siguientes parámetros de entrada son necesarios para la función BUILD_DEBUG:

Nombre del procedimiento SQL
Procedimiento-SQL-fuente o nombre-conjunto-datos-fuente

Si elige la función BUILD_DEBUG y ya existe un procedimiento SQL externo con el nombre *nombre-del-procedimiento-SQL*, DSNTPSMP emite un mensaje de error y se cierra.

REBUILD

Reemplaza todos los objetos que fueron creados por la función BUILD para un procedimiento SQL externo, si existe, de lo contrario crea esos objetos.

Los siguientes parámetros de entrada son necesarios para la función REBUILD:

Nombre del procedimiento SQL
Procedimiento-SQL-fuente o nombre-conjunto-datos-fuente

REBUILD_DEBUG

Reemplaza todos los objetos que fueron creados por la función BUILD_DEBUG para un procedimiento SQL externo, si existe, de lo contrario crea esos objetos, e incluye la preparación necesaria para depurar el procedimiento SQL externo con el depurador SQL y el generador de código (Unified Debugger).

Los siguientes parámetros de entrada son necesarios para la función REBUILD_DEBUG:

Nombre del procedimiento SQL
Procedimiento-SQL-fuente o nombre-conjunto-datos-fuente

REBIND

Enlaza el paquete de procedimientos SQL externos para un procedimiento SQL externo existente.

El siguiente parámetro de entrada es necesario para la función REBIND:

Nombre del procedimiento SQL

DESTRUIR

Elimina los siguientes objetos para un procedimiento SQL externo existente:

- El DBRM, del conjunto de datos al que apunta el nombre DD SQLDBRM
- El módulo de carga, del conjunto de datos al que apunta el nombre DD SQLLMOD
- El código fuente en lenguaje C para el procedimiento SQL externo, desde el conjunto de datos al que apunta el nombre DD SQLCSRC
- El paquete de procedimientos almacenados
- La definición del procedimiento almacenado

El siguiente parámetro de entrada es necesario para la función DESTROY:

Nombre del procedimiento SQL

ALTER

Actualiza el registro de un procedimiento SQL externo existente.

Los siguientes parámetros de entrada son necesarios para la función ALTER:

Nombre del procedimiento SQL
declaración de alteración

ALTER_REBUILD

Actualiza un procedimiento SQL externo existente.

Los siguientes parámetros de entrada son necesarios para la función ALTER_REBUILD:

Nombre del procedimiento SQL
Procedimiento-SQL-fuente o nombre-conjunto-datos-fuente

ALTER_REBUILD_DEBUG

Actualiza un procedimiento SQL externo existente e incluye la preparación necesaria para depurar el procedimiento SQL externo con el depurador SQL y el depurador de procedimientos (Unified Debugger).

Los siguientes parámetros de entrada son necesarios para la función ALTER_REBUILD_DEBUG:

Nombre del procedimiento SQL
Procedimiento-SQL-fuente o nombre-conjunto-datos-fuente

ALTER_REBIND

Actualiza el registro y vincula el paquete SQL para un procedimiento SQL externo existente.

Los siguientes parámetros de entrada son necesarios para la función ALTER_REBIND:

Nombre del procedimiento SQL
declaración de alteración

NIVEL DE CONSULTA

Obtiene el nivel de interfaz de la utilidad de compilación invocada. No se requiere ninguna otra información.

Nombre del procedimiento SQL

Un parámetro de entrada VARCHAR(261) que especifica el nombre del procedimiento SQL externo.

El nombre puede estar cualificado o no cualificado. El nombre debe coincidir con el nombre del procedimiento que se especifica en la instrucción CREATE PROCEDURE que se proporciona en *SQL-procedure-source* o que se obtiene de *source-data-set-name*. Además, el nombre debe coincidir con el nombre del procedimiento que se especifica en la instrucción ALTER PROCEDURE que se proporciona en *alter-statement*. No mezcle referencias cualificadas y no cualificadas.

Fuente de procedimiento SQL

Un parámetro de entrada CLOB (2M) que contiene la instrucción CREATE PROCEDURE para el procedimiento SQL externo. Si especifica una cadena vacía para este parámetro, debe especificar el nombre *source-data-set-name* de un conjunto de datos que contenga el código fuente del procedimiento SQL externo.

opciones de enlace

Un parámetro de entrada VARCHAR(1024) que contiene las opciones que desea especificar para vincular el paquete de procedimientos SQL externo. No especifique la opción MEMBER o LIBRARY para el comando BIND PACKAGE de Db2 .

opciones de compilador

Un parámetro de entrada VARCHAR(255) que contiene las opciones que desea especificar para compilar el programa en lenguaje C que genera Db2 para el procedimiento SQL externo.

opciones-del-precompilador

Un parámetro de entrada VARCHAR(255) que contiene las opciones que desea especificar para precompilar el programa en lenguaje C que genera Db2 para el procedimiento SQL externo. No especifique la opción HOST.

opciones de enlace previo

Un parámetro de entrada VARCHAR(255) que contiene las opciones que desea especificar para preenlazar el programa en lenguaje C que genera Db2 para el procedimiento SQL externo.

opciones de enlace

Un parámetro de entrada VARCHAR(255) que contiene las opciones que desea especificar para vincular el programa en lenguaje C que genera Db2 para el procedimiento SQL externo.

declaración de alteración

Un parámetro de entrada VARCHAR(32672) que contiene la instrucción SQL ALTER PROCEDURE para procesar con la función ALTER o ALTER_REBIND.

nombre-del-conjunto-de-datos-de-origen

Un parámetro de entrada VARCHAR(80) que contiene el nombre de un z/OS conjunto de datos secuenciales o miembro de conjunto de datos particionado que contiene el código fuente para el procedimiento SQL externo. Si especifica una cadena vacía para este parámetro, debe proporcionar el código fuente del procedimiento SQL externo en *SQL-procedure-source*.

propietario del edificio

Un parámetro de entrada VARCHAR(130) que contiene el identificador SQL para servir como propietario de la compilación de los procedimientos almacenados SQL recién creados.

Cuando este parámetro no se especifica, el valor predeterminado es el valor del registro especial CURRENT_SQLID cuando se invoca la utilidad de compilación.

construcción-utilidad

Un parámetro de entrada VARCHAR(255) que contiene el nombre de la utilidad de compilación que se invoca. Se sugiere la forma calificada del nombre, por ejemplo, SYSPROC.DSNTPSMP.

código-retorno

Un parámetro de salida VARCHAR(255) en el que Db2 pone el código de retorno de la invocación DSNTPSMP. Los valores son:

0

Invocación exitosa. La aplicación de llamada puede recuperar opcionalmente el conjunto de resultados y, a continuación, emitir la instrucción SQL COMMIT necesaria.

4

Invocación exitosa, pero se produjeron advertencias. La aplicación de llamada debe recuperar los mensajes de advertencia en el conjunto de resultados y, a continuación, emitir la instrucción SQL COMMIT necesaria.

8

Invocación fallida. La aplicación de llamada debe recuperar los mensajes de error en el conjunto de resultados y, a continuación, emitir la instrucción SQL ROLLBACK necesaria.

99 x

Donde x es un dígito en el rango de 0 a 9. Invocación fallida con errores graves. La aplicación de llamada debe recuperar los mensajes de error en el conjunto de resultados y, a continuación, emitir la instrucción SQL ROLLBACK necesaria. Para ver los mensajes de error que no están en el conjunto de resultados, consulte el registro de trabajo del espacio de direcciones para la ejecución de DSNTPSMP.

999

Error interno grave desconocido

998

Error de configuración del entorno APF

997

Error de configuración de DSNREXX

996

Error global de configuración temporal de la mesa

995

Error de programación interno de REXX

1.2 x

Donde x es un dígito en el rango de 0 a 9. Nivel de DSNTPSMP cuando la solicitud es QUERYLEVEL. La aplicación de llamada puede recuperar el conjunto de resultados para obtener información adicional sobre el nivel de liberación y servicio y, a continuación, emitir la instrucción SQL COMMIT necesaria.

Referencia relacionada

[Descripciones de opciones de proceso de SQL](#)

Puede especificar cualquier opción de proceso de SQL tanto si utiliza el precompilador de Db2 como si utiliza el coprocesador de Db2. Sin embargo, es probable que el coprocesador de Db2 omita ciertas opciones ya que existen opciones del compilador del lenguaje principal que proporcionan la misma información.

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2\)](#)

[Opciones de compilador \(C/C++\) \(XL C/C++ User's Guide\)](#)

[Referencia de opciones de vinculador \(MVS Program Management: User's Guide and Reference\)](#)

Ejemplos de invocación del procesador de procedimientos SQL (DSNTPSMP)

Puede invocar las funciones BUILD, DESTROY, REBUILD y REBIND de DSNTPSMP.

Función BUILD de DSNTPSMP : Llama a DSNTPSMP para crear un procedimiento SQL externo. La información que necesita DSNTPSMP se enumera en la siguiente tabla:

Tabla 54. Las funciones que DSNTPSMP necesita para CONSTRUIR un procedimiento SQL

Función	BUILD
Nombre del procedimiento SQL externo	MYSHEMA.SQLPROC
Ubicación de origen	Cadena en la variable de host CLOB procsrc
Opciones de vinculación	VALIDATE(BIND)
Opciones de compilador	FUENTE, LISTA, NOMBRE COMPLETO, ALQUILER
Opciones de precompilador	FUENTE, XREF, STDSQL(NO)
Opciones de preenlace	Ninguno especificado
Opciones de enlace	AMODE=31, RMODE=CUALQUIERA, MAPA, ALQUILAR
Programa de utilidad de compilación	SYSPROC.DSNTPSMP
Valor de retorno	Cadena devuelta en variable de host de longitud variable returnval

La instrucción CALL es:

```
EXEC SQL CALL SYSPROC.DSNTPSMP('BUILD','MYSHEMA.SQLPROC',:procsrc,  
'VALIDATE(BIND)',  
'SOURCE,LIST,LONGNAME,RENT',  
'SOURCE,XREF,STDSQL(NO)',  
'',  
'AMODE=31,RMODE=ANY,MAP,RENT',  
'',',',',','SYSPROC.DSNTPSMP',  
:returnval);
```

Función DSNTPSMP DESTROY : Llame a DSNTPSMP para eliminar una definición de procedimiento SQL externo y el módulo de carga asociado. La información que necesita DSNTPSMP se enumera en la siguiente tabla:

Tabla 55. Las funciones que DSNTPSMP necesita para DESTRUIR un procedimiento SQL

Función	DESTRUIR
Nombre del procedimiento SQL externo	MYSHEMA.OLDPROC
Valor de retorno	Cadena devuelta en variable de host de longitud variable returnval

La instrucción CALL es:

```
EXEC SQL CALL SYSPROC.DSNTPSMP('DESTROY','MYSHEMA.OLDPROC','','',
                                '',
                                ':returnval');
```

Función DSNTPSMP REBUILD : Llama a DSNTPSMP para volver a crear un procedimiento SQL externo existente. La información que necesita DSNTPMSP se enumera en la siguiente tabla:

Tabla 56. Las funciones que DSNTPSMP necesita para RECONSTRUIR un procedimiento SQL

Función	REBUILD
Nombre del procedimiento SQL externo	MYSHEMA.SQLPROC
Opciones de vinculación	VALIDATE(BIND)
Opciones de compilador	FUENTE, LISTA, NOMBRE COMPLETO, ALQUILER
Opciones de precompilador	FUENTE, XREF, STDSQL(NO)
Opciones de preenlace	Ninguno especificado
Opciones de enlace	AMODE=31, RMODE=CUALQUIERA, MAPA, ALQUILAR
Nombre del conjunto de datos de origen	Miembro PROCSRC del conjunto de datos particionado DSN1210.SDSNSAMP
Valor de retorno	Cadena devuelta en variable de host de longitud variable returnval

La instrucción CALL es:

```
EXEC SQL CALL SYSPROC.DSNTPSMP('REBUILD','MYSHEMA.SQLPROC','','',
                                'VALIDATE(BIND)',
                                'SOURCE,LIST,LONGNAME,RENT',
                                'SOURCE,XREF,STDSQL(NO)',
                                '',
                                'AMODE=31,RMODE=ANY,MAP,RENT',
                                '',
                                'DSN1210.SDSNSAMP(PROCSRC)', '',
                                ':returnval');
```

Si desea recrear un procedimiento SQL externo existente para depurarlo con el depurador SQL y el generador de consultas SQL (Unified Debugger), utilice la siguiente instrucción CALL, que incluye la función REBUILD_DEBUG:

```
EXEC SQL CALL SYSPROC.DSNTPSMP('REBUILD_DEBUG','MYSHEMA.SQLPROC','','',
                                'VALIDATE(BIND)',
                                'SOURCE,LIST,LONGNAME,RENT',
                                'SOURCE,XREF,STDSQL(NO)',
                                '',
                                'AMODE=31,RMODE=ANY,MAP,RENT',
                                '',
                                'DSN1210.SDSNSAMP(PROCSRC)', '',
                                ':returnval');
```

Función DSNTPSMP REBIND : Llame a DSNTPSMP para volver a vincular el paquete para un procedimiento SQL externo existente. La información que necesita DSNTPMSP se enumera en la siguiente tabla:

Tabla 57. Las funciones que DSNTPSMP necesita para REBIND un procedimiento SQL

Función	REBIND
Nombre del procedimiento SQL externo	MYSHEMA.SQLPROC
Opciones de vinculación	VALIDAR (RUN), AISLAMIENTO (RR)
Valor de retorno	Cadena devuelta en variable de host de longitud variable returnval

La instrucción CALL es:

```
EXEC SQL CALL SYSPROC.DSNTPSMP('REBIND','MYSHEMA.SQLPROC','',
      'VALIDATE(RUN),ISOLATION(RR)', '',
      ':returnval');
```

Conjunto de resultados que devuelve el procesador de procedimientos SQL (DSNTPSMP)

DSNTPSMP devuelve un conjunto de resultados que contiene mensajes y listados. Puede escribir su programa de cliente para recuperar información de este conjunto de resultados. Debido a que DSNTPSMP es un procedimiento almacenado, utilice la misma técnica que utilizaría para escribir un programa para recibir conjuntos de resultados de cualquier procedimiento almacenado.

Cada fila del conjunto de resultados contiene la siguiente información:

Etapa de procesamiento

El paso en *el proceso de la función DSNTPSMP* al que se aplica el mensaje.

Nombre DD

La declaración DD que identifica el conjunto de datos que contiene el mensaje.

Número de secuencia

El número de secuencia de una línea de texto de un mensaje.

Mensaje

Una línea de texto del mensaje.

Las filas del conjunto de resultados del mensaje están ordenadas por paso de procesamiento, nombre de DD y número de secuencia.

Para ver un ejemplo de cómo procesar un conjunto de resultados de DSNTPSMP, consulte el programa de ejemplo Db2 DSNTEJ65.

Conceptos relacionados

[Db2 for z/OS Procesador de procedimientos SQL \(DSNTPSMP\)](#)

El procesador de procedimientos SQL, DSNTPSMP, es un procedimiento almacenado REXX que puede utilizar para preparar un procedimiento SQL externo para su ejecución.

[Trabajo DSNTEJ65 \(Db2 Installation and Migration\)](#)

Tareas relacionadas

[Escribir un programa para recibir los conjuntos de resultados de un procedimiento almacenado](#)

Puede escribir un programa para recibir los resultados de un procedimiento almacenado para un número fijo de conjuntos de resultados, cuando conoce el contenido, o un número variable de conjuntos de resultados, cuando no conoce el contenido.

Creación de un procedimiento SQL externo utilizando JCL

Utilizar JCL es una de las muchas formas que puede utilizar para crear y preparar un procedimiento SQL externo.

Antes de empezar

Función en desuso: Los procedimientos de SQL externo están en desuso y no están totalmente soportados como procedimientos SQL nativos. Para obtener mejores resultados, cree procedimientos de SQL nativo en su lugar. Para obtener más información, consulte “[Creación de procedimientos SQL nativos](#)” en la página 244 y “[Migración de un procedimiento de SQL externo a un procedimiento de SQL nativo](#)” en la página 307.

Acerca de esta tarea

Restricción: No puede utilizar JCL para preparar un procedimiento SQL externo para depuración con el depurador de procedimientos almacenados Db2 o el depurador de procedimientos almacenados Unified Debugger. Si planea utilizar alguna de estas herramientas de depuración, utilice DSNTPSMP o IBM Data Studio para crear el procedimiento SQL externo.

Procedimiento

Para crear un procedimiento SQL externo mediante JCL, incluya los siguientes pasos de trabajo en su trabajo JCL:

1. Emitir una instrucción CREATE PROCEDURE que incluya la palabra clave FENCED o la palabra clave EXTERNAL y el cuerpo del procedimiento, que está escrito en SQL.

De forma alternativa, puede emitir la instrucción CREATE PROCEDURE de forma dinámica utilizando una aplicación como SPUFI (DSNTEP2), DSNTIAD (Db2 command line processor) o.

Consejo: Si el cuerpo de rutina de la instrucción CREATE PROCEDURE contiene puntos y comas incrustados, cambie el carácter terminador predeterminado de SQL de un punto y coma a algún otro carácter especial, como el signo de porcentaje (%).

Esta declaración define el procedimiento almacenado para Db2. Db2 almacena la definición en el catálogo de Db2 .

2. Ejecute el programa DSNHPC con la opción HOST(SQL).

Este programa convierte las instrucciones de origen de procedimiento SQL externas en un programa en lenguaje C. DSNHPC también escribe una nueva instrucción CREATE PROCEDURE en el conjunto de datos que se especifica en la instrucción DD de la instrucción CREATE PROCEDURE (SYSUT1).

3. Precompilar, compilar y editar el enlace del programa C generado utilizando una de las siguientes técnicas:

- El precompilador Db2 y las instrucciones JCL para compilar y editar el enlace del programa
- El coprocesador de instrucciones SQL

Cuando realice este paso, especifique los siguientes ajustes:

- Asigne al DBRM el mismo nombre que el nombre del módulo de carga para el procedimiento SQL externo.
- Especifique MARGINS(1,80) para la opción de procesamiento MARGINS SQL.
- Especifique la opción del compilador NOSEQ.

Este proceso produce un programa ejecutable en lenguaje C.

4. Envuelva el DBRM resultante en un paquete.

Ejemplo

Supongamos que define un procedimiento SQL externo emitiendo dinámicamente la siguiente instrucción CREATE PROCEDURE:

```
CREATE PROCEDURE DEVL7083.EMPDTLSS
(
  IN PEMPNO      CHAR(6)
 ,OUT PFIRSTNME  VARCHAR(12)
 ,OUT PMIDINIT   CHAR(1)
```

```

,OUT PLASTNAME      VARCHAR(15)
,OUT PWORKDEPT     CHAR(3)
,OUT PHIREDATE     DATE
,OUT PSALARY        DEC(9,2)
,OUT PSQLCODE       INTEGER
)
RESULT SETS 0
MODIFIES SQL DATA
FENCED
NO DBINFO
WLM ENVIRONMENT DB2AWLMR
STAY RESIDENT NO
COLLID DEVL7083
PROGRAM TYPE MAIN
RUN OPTIONS 'TRAP(OFF),RPTOPTS(OFF)'
COMMIT ON RETURN NO
LANGUAGE SQL
BEGIN
DECLARE SQLCODE INTEGER;
DECLARE SQLSTATE CHAR(5);
DECLARE EXIT HANDLER FOR SQLEXCEPTION SET PSQLCODE = SQLCODE;
SELECT
      FIRSTNME
    , MIDINIT
    , LASTNAME
    , WORKDEPT
    , HIREDATE
    , SALARY
  INTO  PFIRSTNME
    , PMIDINIT
    , PLASTNAME
    , PWORKDEPT
    , PHIREDATE
    , PSALARY
  FROM  EMP
 WHERE  EMPNO = PEMPNO
;
END

```

Puede utilizar un JCL similar al siguiente para preparar el procedimiento:

```

//ADMF001S JOB (999,POK),'SQL C/L/B/E',CLASS=A,MSGCLASS=T,
// NOTIFY=ADMF001,TIME=1440,REGION=0M
/*JOBPARM SYSAFF=SC63,L=9999
// JCLLIB ORDER=(DB2AU.PROCLIB)
/***
//JOBLIB DD DSN=DB2A.SDSNEXIT,DISP=SHR
//          DD DSN=DB2A.SDSNLOAD,DISP=SHR
//          DD DSN=CEE.SCEERUN,DISP=SHR
/***
//** STEP 01: PRECOMP, COMP, LKED AN SQL PROCEDURE
//**-
//SQL01 EXEC DSNIHSQ,MEM=EMPDTLSS,
// PARM.PC='HOST(SQL),SOURCE,XREF,MAR(1,80),STDSQL(NO)',
// PARM.PCC='HOST(C),SOURCE,XREF,MAR(1,80),STDSQL(NO),TWOPASS',
// PARM.C='SOURCE LIST MAR(1,80) NOSEQ LO RENT',
//          PARM.LKED='AMODE=31,RMODE=ANY,MAP,RENT'
//PC.SYSLIB DD DUMMY
//PC.SYSUT2 DD DSN=&&SPDML,DISP=(,PASS), <=MAKE IT PERMANENT, IF YOU
//          UNIT=SYSDA,SPACE=(TRK,1),           WANT TO USE IT LATER
//          DCB=(RECFM=FB,LRECL=80)
//PC.SYSIN   DD DISP=SHR,DSN=SG247083.PROD.DDL(&MEM.)
//PC.SYSCIN   DD DISP=SHR,DSN=SG247083.TEST.C.SOURCE(&MEM.)
//PCC.SYSIN   DD DISP=SHR,DSN=SG247083.TEST.C.SOURCE(&MEM.)
//PCC.SYSLIB  DD DUMMY
//PCC.DBRLLIB DD DISP=SHR,DSN=SG247083.DEVL.DBRM(&MEM.)
//LKED.SYSLMOD DD DISP=SHR,DSN=SG247083.DEVL.LOAD(&MEM.)
//LKED.SYSIN  DD * INCLUDE SYSLIB(DSNRLI)      NAME EMPDTLSS(R)
/*
//**-
//** STEP 02: BIND THE PROGRAM
//**-
//SQL02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB  DD DSN=SG247083.DEVL.DBRM,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//REPORT    DD SYSOUT=*
//SYSIN    DD *
//SYSTSIN  DD *

```

```
DSN SYSTEM(DB2A)
BIND PACKAGE(DEVL7083) MEMBER(EMPDTLSS) VALIDATE(BIND) -
OWNER(DEVL7083)
END
/**
```

Conceptos relacionados

Cuerpo de un procedimiento SQL

El cuerpo de un procedimiento SQL contiene una o más sentencias SQL. En el cuerpo del procedimiento SQL, también puede declarar y utilizar variables, condiciones, códigos de retorno, sentencias, cursores y manejadores.

Db2 command line processor , (Db2 Commands)

Tareas relacionadas

Cambio de los valores predeterminados de SPUFI

Antes de ejecutar sentencias SQL en SPUFI, puede cambiar el comportamiento de ejecución predeterminado, como el terminador SQL y el nivel de aislamiento.

Creación de un procedimiento SQL externo utilizando DSNTPSMP

El procesador de procedimiento SQL, DSNTPSMP, es uno de los distintos métodos que puede utilizar para crear y preparar un procedimiento SQL externo. DSNTPSMP es un procedimiento almacenado REXX que puede invocar desde el programa de aplicación.

Desarrollo de rutinas de base de datos (IBM Data Studio, IBM Optim Database Administrator, IBM infoSphere Data Architect, IBM Optim Development Studio)

Referencia relacionada

Descripciones de opciones de proceso de SQL

Puede especificar cualquier opción de proceso de SQL tanto si utiliza el precompilador de Db2 como si utiliza el coprocesador de Db2. Sin embargo, es probable que el coprocesador de Db2 omita ciertas opciones ya que existen opciones del compilador del lenguaje principal que proporcionan la misma información.

Programas de ejemplo DSNTEP2 y DSNTEP4

Puede utilizar los programas DSNTEP2 o DSNTEP4 para ejecutar sentencias SQL de forma dinámica.

Programa de ejemplo DSNTIAD

Puede utilizar el programa DSNTIAD para ejecutar sentencias de SQL dinámico que no sean sentencias SELECT.

BIND PACKAGE subcomando (DSN) (Comandos de Db2)

Instrucción CREATE PROCEDURE (SQL - procedimiento externo) (obsoleta) (Db2 SQL)

Programas de ejemplo para ayudar a preparar y ejecutar procedimientos SQL externos

Db2 proporciona trabajos de ejemplo para ayudarle a preparar y ejecutar procedimientos SQL externos. Todos los ejemplos están en el conjunto de datos de DSN1210.SDSNSAMP. Antes de poder ejecutar los ejemplos, debe personalizarlos para la instalación.

Función en desuso: Los procedimientos de SQL externo están en desuso y no están totalmente soportados como procedimientos SQL nativos. Para obtener mejores resultados, cree procedimientos de SQL nativo en su lugar. Para obtener más información, consulte “[Creación de procedimientos SQL nativos](#)” en la página 244 y “[Migración de un procedimiento de SQL externo a un procedimiento de SQL nativo](#)” en la página 307.

Consulte el prólogo de cada muestra para obtener instrucciones específicas.

La siguiente tabla muestra los ejemplos de trabajos que Db2 proporciona para procedimientos SQL externos.

Tabla 58. Ejemplos de procedimientos SQL externos enviados con Db2

Miembro que contiene código fuente	Contenido	Finalidad
DSNHSQL	Procedimiento JCL	Precompila, compila, preedita enlaces y edita enlaces un procedimiento SQL externo
DSNTEJ63	Trabajo JCL	Invoca el procedimiento JCL DSNHSQL para preparar el procedimiento SQL externo DSN8ES1 para su ejecución
DSN8ES1	Procedimiento SQL externo	Un procedimiento almacenado que acepta un número de departamento como entrada y devuelve un conjunto de resultados que contiene información salarial para cada empleado de ese departamento
DSNTEJ64	Trabajo JCL	Prepara el programa del cliente DSN8ED3 para su ejecución
DSN8ED3	Programa C	Llama al procedimiento SQL DSN8ES1
DSN8ES2	Procedimiento SQL externo	Procedimiento almacenado que acepta un parámetro de entrada y devuelve dos parámetros de salida. El parámetro de entrada especifica una bonificación que se concederá a los gerentes. El procedimiento SQL externo actualiza la columna BONUS de DSN1210.SDSENSAMP. Si no se produce ningún error SQL cuando se ejecuta el procedimiento SQL externo, el primer parámetro de salida contiene el total de todas las bonificaciones concedidas a los gerentes y el segundo parámetro de salida contiene un valor nulo. Si se produce un error SQL, el segundo parámetro de salida contiene un SQLCODE.
DSN8ED4	Programa C	Llama al procesador de procedimientos SQL, DSNTPSMP, para preparar DSN8ES2 para su ejecución
DSN8WLMP	Procedimiento JCL	Un ejemplo de procedimiento de inicio para el espacio de direcciones de procedimientos almacenados establecido por WLM en el que se ejecuta DSNTPSMP
DSN8ED5	Programa C	Llama al procedimiento SQL externo DSN8ES2
DSNTEJ65	Trabajo JCL	Prepara y ejecuta programas DSN8ED4 y DSN8ED5. DSNTEJ65 Db2 , utiliza DSNTPSMP, el procesador de procedimientos SQL, que requiere que el CCSID EBCDIC predeterminado que utiliza también sea compatible con el compilador C. No ejecute DSNTEJ65 si el CCSID EBCDIC predeterminado para Db2 no es compatible con el compilador C. Algunos ejemplos de CCSID incompatibles son 290, 930, 1026 y 1155.
DSNTEJ67	Trabajo JCL	Prepara un procedimiento SQL externo existente (DSN8.DSN8ES2) para convertirlo en un procedimiento SQL nativo. DSNTEJ67 obtiene la fuente de la DSN8.DSN8ES2 de procedimiento SQL externo del catálogo y la formatea en un conjunto de datos. DSNTEJ67 ejecuta DSNHPSM con HOST(SQLPL), obtiene una lista de la fuente y reemplaza las opciones de procedimiento ofensivas en el conjunto de datos de origen.

Tabla 58. Ejemplos de procedimientos SQL externos enviados con Db2 (continuación)

Miembro que contiene código fuente	Contenido	Finalidad
DSNTIJRT	Trabajo JCL	Prepara un servidor de e Db2 for z/OS s para su funcionamiento con el depurador de SQL y el depurador unificado

DSN8ED4

Demuestra cómo utilizar un programa de aplicación para llamar a DSNTPSMP, el procesador de procedimientos SQL de Db2 .

```
***** 00010000
* Module name = DSN8ED4 (sample program) 00020000
* 00030000
* DESCRIPTIVE NAME: Sample client for: 00040000
* DSNTPSMP (DB2 SQL Procedures Processor) 00050000
*
* LICENSED MATERIALS - PROPERTY OF IBM 00060000
* 5625-DB2 00070000
* (C) COPYRIGHT 1982, 2003 IBM CORP. ALL RIGHTS RESERVED. 00080000
*
* STATUS = VERSION 8 00090000
*
* Function: Demonstrates how to use an application program to call 00100000
* DSNTPSMP, the DB2 SQL Procedures Processor. DSN8ED4 00110000
* collects and passes user-provided SQL Procedure source 00120000
* code and prep options to DSNTPSMP, and outputs the 00130000
* report(s), if any, returned from DSNTPSMP by result set. 00140000
*
* Notes: 00150000
* Dependencies: Requires SYSPROC.DSNTPSMP 00160000
*
* Restrictions: 00170000
*
* Module type: C program 00180000
* Processor: DB2 Precompiler 00190000
* IBM C/C++ for OS/390 V1R3 or higher 00200000
* Module size: See linkedit output 00210000
* Attributes: Reentrant and reusable 00220000
*
* Entry point: DSN8ED4 00230000
* Purpose: See Function 00240000
* Linkage: Standard MVS program invocation, three parameters. 00250000
*
* Parameters: DSN8ED4 uses the C "main" argument convention of 00260000
* argv (argument vector) and argc (argument count). 00270000
*
* - ARGV[0]: (input) pointer to a char[9], 00280000
* null-terminated string having the name of 00290000
* this program (DSN8ED4) 00300000
* - ARGV[1]: (input) pointer to a char[21], 00310000
* null-terminated string having the action 00320000
* that DSNTPSMP is to perform: 00330000
* - BUILD: Prepare a new SQL Procedure 00340000
* - REBUILD: Prepare an existing SQL 00350000
* - QUERYLEVEL: Verify DSNTPSMP level @05* 00360000
* - DESTROY: Remove an SQL Procedure 00370000
* - REBIND: Rebind the package of an exist- 00380000
* ing SQL Procedure 00390000
* - ARGV[2]: (input) pointer to a char[262], 00400000
* null-terminated string having the schema 00410000
* and name of the SQL Procedure to be 00420000
* processed by DSNTPSMP (e.g. DSN8.DSN8ES2) 00430000
* - ARGV[3]: (input) pointer to a char[9], 00440000
* null-terminated string having the author- 00450004
* ization id to be used for BUILDFOWNER and 00460000
* for calling DSNTPSMP. 00470000
* - ARGV[4]: (input) pointer to a char[17], 00480000
* null-terminated string having the name of 00490000
* the server where DSNTPSMP is to be run. 00500000
* This is an optional parameter; the local 00510000
* server is used if no argument is provided. 00520000
* * 00530000
* * 00540000
* * 00550003
* * 00560003
* * 00570000
* * 00580000
* * 00590000
* * 00600000
* * 00610000
* * 00620000
```

```

*      Inputs: DSN8ED4 allocates these input DDs:                                * 00630000
*              - PCOPTS : Options for the DB2 precompiler                         * 00640000
*              - COPTS  : Options for the C compiler                            * 00650000
*              - PLKDOPTS: Options for the pre-link editor                         * 00660000
*              - LKEDOPTS: Options for the link editor                           * 00670000
*              - BINDOPTS: Options for the DB2 BIND                             * 00680000
*              - SQLIN   : Source code for the SQL Procedure                      * 00690000
*                                         * 00700000
*      Outputs: DSN8ED4 allocates these output DD                                * 00710000
*              - REPORT01: First report data set from DSNTPSMP                   * 00720000
*              - REPORT02: Second report data set from DSNTPSMP                  * 00730000
*              - REPORT03: Third report data set from DSNTPSMP                   * 00740000
*                                         * 00750000
* Normal Exit: Return Code: 0000                                              * 00760000
*             - Message: DSNTPSMP has completed with return code 0    * 00770000
*             - Message: SQL changes have been committed @04* 00780000
*                                         * 00790000
* Normal with Warnings Exit: Return Code: 0004                               +@04* 00800000
*             - Message: DSNTPSMP has completed with return code 4    * 00810000
*             - Message: SQL changes have been committed -@04* 00820000
*                                         * 00830000
* Error Exit: Return Code: 0012                                              * 00840000
*             - Message: DSNTPSMP has completed with return code <n>* 00850000
*             - Message: The length of the argument specified for      * 00860000
*                           the <parameter-name> does not fall within     * 00870000
*                           the required bounds of <minimum-length> * 00880000
*                           and <maximum-length>           * 00890000
*             - Message: The argument specified for the action      * 00900000
*                           parameter is invalid                * 00910000
*             - Message: Invalid sequence number <sequence-number> * 00920000
*                           specified for REPORTnn DD          * 00930000
*             - Message: DSN8ED4 was invoked with <parameter-count> * 00940000
*                           parameters. At least 3 parameters are   * 00950000
*                           required                     * 00960000
*             - Message: Unable to open <DD-name>                    * 00970000
*             - Message: Unable to close <DD-name>                   * 00980000
*             - Message: <formatted SQL text from DSNTIAR>        * 00990000
*             - Message: SQL changes have been rolled back         @04* 01000000
*                                         * 01010000
* External References:                                                       * 01020000
*             - Routines/Services: DSNTIAR: DB2 msg text formatter * 01030000
*             - Data areas       : None                          * 01040000
*             - Control blocks  : None                          * 01050000
*                                         * 01060000
*                                         * 01070000
* Pseudocode:
* DSN8ED4:
*             - call getCallParms to receive and validate call parm arguments* 01100000
*             - case action                                         * 01110000
*             - when BUILD, call getReBuildData                  * 01120000
*             - when DESTROY, call getDestroyData                 * 01130000
*             - when REBUILD, call getReBuildData                 * 01140000
*             - when REBIND, call getRebindData                  * 01150000
*             - when QUERYLEVEL, call getLevelData                * 01160003
*             - otherwise call issueInvalidActionError           * 01170000
*             - call connectToLocation                            * 01180000
*             - call setAuthID to set the current authorization id @pq53353 * 01190003
*             - call callDSNTPSMP to invoke the DB2 SQL Procedures Processor * 01200000
*             - call processDSNTPSMPresultSet to write reports from DSNTPSMP * 01210000
*             - If no errors, call processSqlCommit to commit work @04 * 01220000
*             Else call processSqlRollback to undo work        @04 * 01230000
* End DSN8ED4                                                               * 01240000
*                                         * 01250000
*                                         * 01260000
* Change activity =
* PQ46962 03/28/2001 changed line feed character to hex 25 @01 * 01280000
* PQ43444 04/12/2001 Disable LEOPTS DD (LE options are not @02 * 01290000
* processed by DSNTPSMP). Remission the @02 * 01300000
* leOptions hostvar as alterStmt. @02 * 01310000
* PQ56601 03/06/2002 Trim +/- continuation characters from @03 * 01320000
* BIND options to prevent BIND errors. @03 * 01330000
* These characters are often used to con- @03 * 01340000
* tinue BIND statements being processed @03 * 01350000
* by the DB2 DSN command processor (which @03 * 01360000
* uses TSO i/o services that recognize @03 * 01370000
* them as continuation characters) but @03 * 01380000
* they are not otherwise valid in DB2 @03 * 01390000
* commands. @03 * 01400000
* PQ61782 07/16/2002 Distinguish between DSNTPSMP return code @04 * 01410000
* and DSN8ED4 return code; Issue SQL COMMIT@04 * 01420000
* when DSNTPSMP returns rc = 0 or rc = 4; @04 * 01430000
* Otherwise issue SQL ROLLBACK @04 * 01440000

```

```

*      D55199 12/08/2003 Adjust to use DSNTPSMP 1.2x interface    @05 * 01450004
*      D56462 02/12/2004 Allocate maximum of 6 output reports    @06 * 01460005
***** 01470000
***** 01480000
***** C library definitions *****
#include      <errno.h>          01500000
#include      <stdio.h>           01510000
#include      <stdlib.h>          01520000
#include      <string.h>          01530000
                                         01540000
***** Constants *****
#define      NULLCHAR        '\0' /* Null character          */ 01560000
#define      RETNRM          0   /* Normal return code     */ @04*/ 01570000
#define      RETWRN          4   /* Warning return code    */ */ 01580000
#define      RETERR          8   /* Error return code     */ */ 01590000
#define      RETSEV          12  /* Severe error return code */ */ 01600000
#define      INTERFACE        "1.2" /* DSNTPSMP interface level */ */ 01610003
                                         01620000
enum flag      {No, Yes}; /* Settings for flags */ */ 01630000
                                         01640000
                                         01650000
***** Input: SQL Procedure Source Code *****
FILE          *sqlInFile; /* Pointer to SQL source DD */ */ 01670000
                                         01680000
                                         01690000
***** Output: DB2 SQL Procedures Processor Reports *****
FILE          *reportDD; /* Pointer to curr report DD */ */ 01710000
char          reportDDName[12]; /* For generated DD name */ 01720000
unsigned short reportLRECL; /* length req'd for output rec*/ 01730000
                                         01740000
                                         01750000
***** Working variables *****
unsigned short resultSetReturned = 0; /* DSNTPSMP result set stat */ @04*/ 01770000
long int      DSNTPSMP_rc      = -1; /* DSNTPSMP return code */ @04*/ 01780000
long int      rc              = 0; /* program return code */ */ 01790000
char          levelquery      = 'N'; /* Is this a level check? */ @05*/ 01800004
                                         01810000
                                         01820000
***** DB2 SQL Communication Area *****
EXEC SQL INCLUDE SQLCA;
                                         01840000
                                         01850000
                                         01860000
***** DB2 Host Variables *****
EXEC SQL BEGIN DECLARE SECTION;
                                         01870000
                                         01880000
                                         01890000
char          authID[9]; /* Authorization id-BUILDOWNER*/ 01900000
                                         01910000
char          locationName[17]; /* Server location name */ */ 01920000
                                         01930000
char          action[21]; /* Command for PSM processor */ 01940000
char          routineName[262]; /* SQL Procedure schema.name */ 01950000
SQL TYPE IS CLOB(2M) sqlSource; /* SQL Procedure source */ @05*/ 01960004
                                         01970000
char          precompOptions[256]; /* precompiler options */ */ 01980000
char          compileOptions[256]; /* compilation parameters */ */ 01990000
char          prelinkOptions[256]; /* prelink options */ */ 02000000
char          linkOptions[256]; /* link-edit options */ */ 02010000
char          bindOptions[1025]; /* DB2 bind options */ */ 02020000
char          alterStmt[32672]; /* ALTER PROC text */ @02*/ 02030000
                                         02040000
char          sqlSourceDsn[81]; /* Source data set name */ */ 02050000
char          outputString[256]; /* DSNTPSMP status area */ */ 02060000
                                         02070000
char          DSNTPSMP_pname[19]; /* DSNTPSMP procedure-name */ */ 02080000
= "SYSPROC.DSNTPSMP\0";
                                         02090000
                                         02100000
char          stepName[17]; /* DSNTPSMP stepname */ */ 02110000
char          fileName[9]; /* DSNTPSMP output DD name */ */ 02120000
long int      reportLineNumber; /* DSNTPSMP report line no. */ */ 02130000
char          reportLine[256]; /* DSNTPSMP report line */ */ 02140000
                                         02150000
                                         02160000
                                         02170000
                                         02180000
***** DB2 Result Set Locator Host Variables *****
EXEC SQL BEGIN DECLARE SECTION;
                                         02190000
                                         02200000
static volatile SQL TYPE IS RESULT_SET_LOCATOR *DSNTPSMP_rs_loc1;
                                         02210000
EXEC SQL END DECLARE SECTION;
                                         02220000
                                         02230000
                                         02240000
***** DSN8ED4 Function Models *****
int main; /* DSN8ED4 driver */ */ 02250000
                                         02260000

```

```

        ( int argc,
          char *argv[]
        );
void getCallParms
        ( int argc,
          char *argv[]
        );
void getReBuildData( void );
void getDestroyData( void );
void getRebindData( void );
void getLevelData( void );
void getOptions
        ( char *options,
          int maxBytes,
          char *optionsDDname
        );
void getSqlSource( void );
void setAuthID( void );
void connectToLocation( void );
void callDSNTPSMP( void );
void listDSNTPSMPcallParms( void );
void processDSNTPSMPresultSet( void );
void associateResultSetLocator(void);
void allocateResultSetCursor( void );
void writeDSNTPSMPreports( void );
void fetchFromResultSetCursor( void );
void openReportDataSet
        ( short int reportNumber
        );
void closeReportDataSet( void );
void trimTrailingBlanks
        ( char           *string
        );
void stripContinuationCharacter
        ( char           *string
        );
void processSqlCommit( void );
void processSqlRollback( void );
void issueDataSetClosingError
        ( char           *DDname,
          int            LEerrno
        );
void issueDataSetOpeningError
        ( char           *DDname,
          int            LEerrno
        );
void issueDataSetReadingError
        ( char           *DDname,
          int            LEerrno
        );
void issueInvalidCallParmCountError
        ( int argc
        );
void issueInvalidActionError
        ( char *action
        );
void issueInvalidLevelError
        ( char *level
        );
void issueInvalidDDnumError
        ( short          invalidDDnum
        );
void issueInvalidParmLengthError
        ( char *parmName,
          int minLength,
          int maxLength
        );
void issueSqlError
        ( char *locMsg
        );

int main
        ( int argc,
          char *argv[]
        );
/**************************************************************************
 * Main Driver:
 * - Gets arguments for call parms
 * - Gets processing options and data
 * - Connects to remote location, if one was specified
 * - Calls the DB2 SQL Procedure Processor, DSNTSPMP
 */

```

/* - Input argument count */ 02270000
 /* - Input argument vector */ 02280000
 02290000
 /* Process args to call parms */ 02300000
 /* - Input argument count */ 02310000
 /* - Input argument vector */ 02320000
 02330000
 /* Get SQL Proc re/build data */ 02340000
 /* Get SQL Proc destroy data */ 02350000
 /* Get SQL Proc rebind data */ 02360000
 /* Get DSNTPSMP level data */ 02370003
 /* Read specified options file*/ 02380000
 /* -out: list of options read */ 02390000
 /* - in: max size of list */ 02400000
 /* - in: name of DD to read */ 02410000
 02420000
 /* Read SQL Procedure Source */ 02430000
 /* Set the current DB2 auth id*/ 02440003
 /* Connect to DB2 location */ 02450000
 /* Run SQL Procedure Processor*/ 02460000
 /* List parms sent to DSNTPSMP*/ 02470000
 /* Process DSNTPSMP rslt sets */ 02480000
 /* Assoc DSNTPSMP RS locator */ 02490000
 /* Alloc DSNTPSMP RS cursor */ 02500000
 /* Output a DSNTPSMP report */ 02510000
 /* Read DSNTPSMP RS cursor */ 02520000
 /* Alloc DD for a report */ 02530000
 /* - in: Seqeunce number */ 02540000
 02550000
 /* Dealloc DD for a report */ 02560000
 /* Strip off trailing blanks */ 02570000
 /* - in: string to be trimmed */ 02580000
 02590000
 /* Strip off trailing - or + */ 02600000
 /* - in: string to be trimmed */ 02610000
 /*@03*/ 02620000
 /* Commit SQL changes @04*/ 02630000
 /* Rollback SQL changes @04*/ 02640000
 /* Handler for ds close error */ 02650000
 /* - in: name of errant DD */ 02660000
 /* - in: LE diagnostic errno */ 02670000
 02680000
 /* Handler for ds open error */ 02690000
 /* - in: name of errant DD */ 02700000
 /* - in: LE diagnostic errno */ 02710000
 02720000
 /* Handler for ds read error */ 02730000
 /* - in: name of errant DD */ 02740000
 /* - in: LE diagnostic errno */ 02750000
 02760000
 /* Handler for parm count err */ 02770000
 /* - in: no. parms received */ 02780000
 02790000
 /* Handler for unknown action */ 02800000
 /* - in: action specified */ 02810000
 02820000
 /* Handler for wrong DSNTPSMP */ 02830003
 /* - in: level encountered */ 02840003
 02850003
 /* Handler for unknown DD seq */ 02860000
 /* - in: invalid DD sequ. no. */ 02870000
 02880000
 /* Handler for parm len error */ 02890000
 /* - in: identify of parm */ 02900000
 /* - in: min valid length */ 02910000
 /* - in: max valid length */ 02920000
 02930000
 /* Handler for SQL error */ 02940000
 /* - in: Call location */ 02950000
 02960000
 02970000
 02980000
 /* DSN8ED4 driver */ 02990000
 /* - Input argument count */ 03000000
 /* - Input argument vector */ 03010000
 03020000
 /*@03030000 */ 03030000
 /* 03040000 */ 03040000
 /* 03050000 */ 03050000
 /* 03060000 */ 03060000
 /* 03070000 */ 03070000
 /* 03080000 */ 03080000

```

* - Processes any result set(s) returned from DSNTPSMP           * 03090000
*
***** */ 03110000
{ /***** 03120000
* Extract the following information from the call parms:      * 03130000
* (1) DB2 location name where where SQL Procedure is to be built,* 03140000
* destroyed, rebuilt, rebound, etc.)                          * 03150000
* (2) DB2 SQL Procedure Processor action (Build,Destroy,...)   * 03160000
* (3) Name of SQL Procedure to be built, destroyed, rebound, etc.* 03170000
***** */ 03180000
getCallParms( argc,argv );
03190000
03200000
03210000
03220000
***** */ 03230000
if( rc < RETSEV )                                              03240000
{ if( memcmp( action,"BUILD",5 ) == 0 )                         03250000
{ getReBuildData();                                            03260000
}
else if( memcmp( action,"DESTROY",7 ) == 0 )                   03280000
{ getDestroyData();                                            03290000
}
else if( memcmp( action,"REBUILD",7 ) == 0 )                  03310000
{ getReBuildData();                                            03320000
}
else if( memcmp( action,"REBIND",6 ) == 0 )                   03330000
{ getRebindData();                                            03340000
}
else if( memcmp( action,"QUERYLEVEL",10 ) == 0 )              03350000
{ getLevelData();
  levelquery='Y';
}
else
{ issueInvalidActionError( action );
}
}
03410000
03420000
03430000
03440000
03450000
03460000
03470000
***** */ 03480000
if( rc < RETSEV && strlen(locationName) > 0 )            03490000
connectToLocation();                                         03500000
03510003
03520003
03530003
03540003
/*@pq53353*/ 03550003
setAuthID();                                                 03560003
03570000
03580000
03590000
***** */ 03600000
if( rc < RETSEV )                                              03610000
callDSNTPSMP();                                             03620000
03630000
03640000
03650000
***** */ 03660000
if( resultSetReturned )                                       /*@04*/
processDSNTPSMPrresultSet();                                03670000
03680000
03690000
/*@04** 03700000
* If DSNTPSMP returns either 0 (normal) or 4 (warnings), commit * 03710000
* the SQL changes; Otherwise, rollback the SQL changes        * 03720000
***** */ 03730000
if( DSNTPSMP_rc == RETNRM || DSNTPSMP_rc == RETWRN )       03740000
{ processSqlCommit();
  if( rc < DSNTPSMP_rc )
    rc = DSNTPSMP_rc;
}
03750000
03760000
03770000
03780000
03790000
03800000
03810000
03820000
/*-@04*/ 03830000
03840000
03850000
03860000
***** */ 03870000
return( rc );
03880000
03890000
03900000
} /* end of main */

```

```

void getCallParms                  /* Process args to call parms */ 03910000
( int argc,                      /* - Input argument count */ 03920000
  char *argv[]                   /* - Input argument vector */ 03930000
)
/********************************************* 03940000
* Verifies that correct call parms have been passed in:      * 03950000
* - Three parameters (action, routine name, and authorization id) * 03960000
*   require arguments                                              * 03970000
* - The fourth parameter (location name) is optional           * 03980000
***** 03990000
{ if( argc < 4 || argc > 5 )                                         04000000
    { issueInvalidCallParmCountError( argc );                           04010000
    }
    else if( strlen( argv[1] ) < 1 || strlen( argv[1] ) > 20 )          04020000
    { issueInvalidParmLengthError("DSNTPSMP Action",1,20);             04030000
    }
    else if( strlen( argv[2] ) < 1 || strlen( argv[2] ) > 261 )          04040000
    { issueInvalidParmLengthError("SQL Procedure schema.name",1,261); 04050000
    }
    else if( strlen( argv[3] ) < 1 || strlen( argv[3] ) > 8 )            04060000
    { issueInvalidParmLengthError("Authorization ID",1,8);              04070000
    }
    else
    { strcpy( action, argv[1] );                                         04080000
        strcpy( routineName, argv[2] );                                     04090000
        strcpy( authID, argv[3] );                                         04100000
    }
}
04110000
04120000
04130000
04140000
04150000
04160000
04170000
04180000
04190000
04200000
04210000
04220000
04230000
04240000
04250000
04260000
04270000
04280000
04290000
04300000
04310000
04320000
04330000
04340000
04350000
04360000
04370000
04380000
04390000
04400000
04410000
04420000
04430000
04440000
04450000
04460000
04470000
04480000
04490000
04500000
04510000
04520000
04530000
04540000
04550000
04560000
04570000
04580000
04590000
04600000
04610000
04620000
04630000
04640000
04650000
04660000
04670000
04680000
04690000
04700000
04710004
04720004
}

void getReBuildData( void )          /* Get SQL Proc re/build data */ 04730000
/********************************************* 04740000
* Collects the prep options and source data needed by DSNTPSMP to * 04750000
* perform a BUILD or REBUILD operation.                         * 04760000
***** 04770000
{
    /***** 04780000
    * Get program prep, bind, and runtime options                 * 04790000
    ***** 04800000
    getOptions( precompOptions,255,"PCOPTS" );
    if( rc < RETSEV )
        getOptions( compileOptions,255,"COPTS" );
    if( rc < RETSEV )
        getOptions( prelinkOptions,255,"PLKDOPTS" );
    if( rc < RETSEV )
        getOptions( linkOptions,255,"LKEDOPTS" );
    if( rc < RETSEV )
        getOptions( bindOptions,1024,"BINDOPTS" );
    /* if( rc < RETSEV ) @02*/ 04810000
    /*     getOptions( LeOptions,254,"LEOPTS" ); @02*/ 04820000
    */
    /***** 04830000
    * Get the source for the SQL procedure to be prepared       * 04840000
    ***** 04850000
    if( rc < RETSEV )
        getSqlSource();
}
04860000
04870000
04880000
04890000
04900000
04910000
04920000
04930000
04940000
04950000
04960000
04970000
04980000
04990000
05000000
05010000
05020000
05030000
05040000
05050000
05060000
05070000
05080000
05090000
05100000
05110000
05120000
05130000
05140000
05150000
05160000
05170000
05180000
05190000
05200000
05210000
05220000
05230000
05240000
05250000
05260000
05270000
05280000
05290000
05300000
05310000
05320000
05330000
05340000
05350000
05360000
05370000
05380000
05390000
05400000
05410000
05420000
05430000
05440000
05450000
05460000
05470000
05480000
05490000
05500000
05510000
05520000
05530000
05540000
05550000
05560000
05570000
05580000
05590000
05600000
05610000
05620000
05630000
05640000
05650000
05660000
05670000
05680000
05690000
05700000
05710000
05720000
05730000
05740000
05750000
05760000
05770000
05780000
05790000
05800000
05810000
05820000
05830000
05840000
05850000
05860000
05870000
05880000
05890000
05900000
05910000
05920000
05930000
05940000
05950000
05960000
05970000
05980000
05990000
06000000
06010000
06020000
06030000
06040000
06050000
06060000
06070000
06080000
06090000
06100000
06110000
06120000
06130000
06140000
06150000
06160000
06170000
06180000
06190000
06200000
06210000
06220000
06230000
06240000
06250000
06260000
06270000
06280000
06290000
06300000
06310000
06320000
06330000
06340000
06350000
06360000
06370000
06380000
06390000
06400000
06410000
06420000
06430000
06440000
06450000
06460000
06470000
06480000
06490000
06500000
06510000
06520000
06530000
06540000
06550000
06560000
06570000
06580000
06590000
06600000
06610000
06620000
06630000
06640000
06650000
06660000
06670000
06680000
06690000
06700000
06710000
06720000
06730000
06740000
06750000
06760000
06770000
06780000
06790000
06800000
06810000
06820000
06830000
06840000
06850000
06860000
06870000
06880000
06890000
06900000
06910000
06920000
06930000
06940000
06950000
06960000
06970000
06980000
06990000
07000000
07010000
07020000
07030000
07040000
07050000
07060000
07070000
07080000
07090000
07100000
07110000
07120000
07130000
07140000
07150000
07160000
07170000
07180000
07190000
07200000
07210000
07220000
07230000
07240000
07250000
07260000
07270000
07280000
07290000
07300000
07310000
07320000
07330000
07340000
07350000
07360000
07370000
07380000
07390000
07400000
07410000
07420000
07430000
07440000
07450000
07460000
07470000
07480000
07490000
07500000
07510000
07520000
07530000
07540000
07550000
07560000
07570000
07580000
07590000
07600000
07610000
07620000
07630000
07640000
07650000
07660000
07670000
07680000
07690000
07700000
07710000
07720000
07730000
07740000
07750000
07760000
07770000
07780000
07790000
07800000
07810000
07820000
07830000
07840000
07850000
07860000
07870000
07880000
07890000
07900000
07910000
07920000
07930000
07940000
07950000
07960000
07970000
07980000
07990000
08000000
08010000
08020000
08030000
08040000
08050000
08060000
08070000
08080000
08090000
08100000
08110000
08120000
08130000
08140000
08150000
08160000
08170000
08180000
08190000
08200000
08210000
08220000
08230000
08240000
08250000
08260000
08270000
08280000
08290000
08300000
08310000
08320000
08330000
08340000
08350000
08360000
08370000
08380000
08390000
08400000
08410000
08420000
08430000
08440000
08450000
08460000
08470000
08480000
08490000
08500000
08510000
08520000
08530000
08540000
08550000
08560000
08570000
08580000
08590000
08600000
08610000
08620000
08630000
08640000
08650000
08660000
08670000
08680000
08690000
08700000
08710000
08720000
08730000
08740000
08750000
08760000
08770000
08780000
08790000
08800000
08810000
08820000
08830000
08840000
08850000
08860000
08870000
08880000
08890000
08900000
08910000
08920000
08930000
08940000
08950000
08960000
08970000
08980000
08990000
09000000
09010000
09020000
09030000
09040000
09050000
09060000
09070000
09080000
09090000
09100000
09110000
09120000
09130000
09140000
09150000
09160000
09170000
09180000
09190000
09200000
09210000
09220000
09230000
09240000
09250000
09260000
09270000
09280000
09290000
09300000
09310000
09320000
09330000
09340000
09350000
09360000
09370000
09380000
09390000
09400000
09410000
09420000
09430000
09440000
09450000
09460000
09470000
09480000
09490000
09500000
09510000
09520000
09530000
09540000
09550000
09560000
09570000
09580000
09590000
09600000
09610000
09620000
09630000
09640000
09650000
09660000
09670000
09680000
09690000
09700000
09710000
09720000
09730000
09740000
09750000
09760000
09770000
09780000
09790000
09800000
09810000
09820000
09830000
09840000
09850000
09860000
09870000
09880000
09890000
09900000
09910000
09920000
09930000
09940000
09950000
09960000
09970000
09980000
09990000
09999999
}

```

```

precompOptions[0]      = NULLCHAR;                                04730000
compileOptions[0]      = NULLCHAR;                                04740000
prelinkOptions[0]      = NULLCHAR;                                04750000
linkOptions[0]          = NULLCHAR;                                04760000
alterStmt[0]           = NULLCHAR;                                /*@02*/ 04770000
sqlSourceDsn[0]         = NULLCHAR;                                04780000
outputString[0]         = NULLCHAR;                                04790000
                                         04800000
/********************************************* 04810000
* Get name of package to free                         * 04820000
***** 04830000
getOptions( bindOptions,1024,"BINDOPTS" );
                                         04840000
                                         04850000
} /* end of getDestroyData */                                04860000
                                         04870000
                                         04880000
void getRebindData( void )           /* Rebind an SQL Procedure */ 04890000
/********************************************* 04900000
* Gets the name of the package to be rebound by DSNTPSMP during a * 04910000
* REBIND operation.                                         * 04920000
***** 04930000
{
                                         04940000
/********************************************* 04950000
* Set program prep and runtime options to NULLCHAR        * 04960000
***** 04970000
sqlSource.length        = 0;                                     /*@05*/ 04980004
sqlSource.data[0]         = NULLCHAR;                            /*@05*/ 04990004
precompOptions[0]         = NULLCHAR;                                05000000
compileOptions[0]         = NULLCHAR;                                05010000
prelinkOptions[0]         = NULLCHAR;                                05020000
linkOptions[0]           = NULLCHAR;                                05030000
alterStmt[0]             = NULLCHAR;                                /*@02*/ 05040000
sqlSourceDsn[0]           = NULLCHAR;                                05050000
outputString[0]           = NULLCHAR;                                05060000
                                         05070000
/********************************************* 05080000
* Get parameters to pass for rebind                      * 05090000
***** 05100000
getOptions( bindOptions,1024,"BINDOPTS" );
                                         05110000
                                         05120000
} /* end of getRebindData */                                05130000
                                         05140000
                                         05150000
void getLevelData( void )           /* QueryLevel of DSNTPSMP */ 05160003
/********************************************* 05170003
* Prepare for a DSNTPSMP QUERYLEVEL operation.            * 05180003
***** 05190003
{
                                         05200003
/********************************************* 05210003
* Set program prep and runtime options to NULLCHAR        * 05220003
***** 05230003
sqlSource.length        = 0;                                     /*@05*/ 05240004
sqlSource.data[0]         = NULLCHAR;                            /*@05*/ 05250004
precompOptions[0]         = NULLCHAR;                                05260003
compileOptions[0]         = NULLCHAR;                                05270003
prelinkOptions[0]         = NULLCHAR;                                05280003
linkOptions[0]           = NULLCHAR;                                05290003
alterStmt[0]             = NULLCHAR;                                /*@02*/ 05300003
sqlSourceDsn[0]           = NULLCHAR;                                05310003
outputString[0]           = NULLCHAR;                                05320003
                                         05330003
} /* end of getLevelData */                                05340003
                                         05350003
                                         05360003
void getOptions
( char *options,
  int maxBytes,
  char *optionsDDname
)
                                         /* Read processing options */ 05370000
                                         /* -out: list of options read */ 05380000
                                         /* -in: max size of list */ 05390000
                                         /* -in: nameof DD to read */ 05400000
                                         05410000
/********************************************* 05420000
* Reads up to maxBytes bytes of data from optionsDDname into the * 05430000
* options buffer.                                         * 05440000
***** 05450000
{
FILE           *optionsFile; /* Ptr to specified options DD*/ 05460000
char           optionsDD[12]; /* DD handle */ 05470000
char           optionsRec[80]; /* Options file input record */ 05480000
short int      recordLength = 0; /* Length of record */ 05490000
unsigned short moreRecords = Yes; /* EOF indicator */ 05500000
                                         05510000
sprintf( optionsDD,
          "DD:%s\0",
          optionsDDname );
                                         05520000
                                         05530000
                                         05540000

```

```

errno = 0; /* clear LE errno */ 05550000
optionsFile = fopen( optionsDD, "rb,lrecl=80,type=record" ); 05560000
if( optionsFile == NULL ) 05570000
    issueDataSetOpeningError( optionsDD,errno ); 05580000
    05590000
    05600000
    05610000
while( moreRecords == Yes && rc < RETSEV ) 05620000
{ recordLength 05630000
    = fread( optionsRec, /* Read into options rec area */ 05640000
        1, /* ..1 record */ 05650000
        80, /* ..of 80 bytes */ 05660000
        optionsFile ); /* ..from current options file*/ 05670000
    05680000
    if( ferror(optionsFile) ) /* Handle IO errors */ 05690000
        issueDataSetReadingError( optionsDD,errno ); 05700000
    05710000
    else if( feof(optionsFile) ) /* Handle EOF */ 05720000
        moreRecords = No; /* Discard bytes 73-80 and */ 05730000
    else /* strip off trailing blanks */ 05740000
        { strncat( options,optionsRec,72 ); 05750000
            trimTrailingBlanks( options ); 05760000
            /* Remove +/- continuation chars from BIND input @03*/ 05770000
            if( memcmp( optionsDDname,"BINDOPTS",8 ) == 0 ) /*@03*/ 05780000
                stripContinuationCharacter( options ); /*@03*/ 05790000
            05800000
        }
        /* Don't overfill return area */ 05810000
    if( rc < RETSEV && strlen(options) > maxBytes ) 05820000
        issueInvalidParmLengthError( optionsDD,0,maxBytes ); 05830000
    05840000
}
05850000
05860000
if( rc < RETSEV ) 05870000
    if( fclose( optionsFile ) != 0 ) 05880000
        issueDataSetClosingError( optionsDD,errno ); 05890000
    05900000
}
05910000
05920000
05930000
void getSqlSource( void ) /* Read SQL Procedure Source */ 05940000
/********************************************* 05950000
* Reads up to 2M bytes of SQL Procedure source code from the * 05960000
* SQLIN DD. * 05970000
********************************************/ 05980000
{ char sourceRec[80]; /* Source file input record */ 05990000
    short int recordLength = 0; /* Length of record */ 06000000
    unsigned short moreRecords = Yes; /* EOF indicator */ 06010000
06020000
/********************************************* 06030000
* Open the data set having the source for the SQL Procedure * 06040000
********************************************/ 06050000
errno = 0; /* clear LE errno */ 06060000
sqlInFile = fopen( "DD:SQLIN", "rb,lrecl=80,type=record" ); 06070000
if( sqlInFile == NULL ) 06080000
    issueDataSetOpeningError( "DD:SQLIN",errno ); 06090000
06100000
while( moreRecords == Yes && rc < RETSEV ) 06110000
{ recordLength 06120000
    = fread( sourceRec, /* Read into source rec area */ 06130000
        1, /* ..1 record */ 06140000
        80, /* ..of 80 bytes */ 06150000
        sqlInFile ); /* ..from SQL Proc source file*/ 06160000
    06170000
    06180000
    if( ferror(sqlInFile) ) /* Handle IO errors */ 06190000
        issueDataSetReadingError( "DD:SQLIN",errno ); 06200000
    06210000
    else if( feof(sqlInFile) ) /* Handle EOF */ 06220000
        moreRecords = No; /* Discard bytes 73-80, strip */ 06230000
    else /* trailing blanks,add NL char*/ 06240000
        { sourceRec[72] = NULLCHAR; 06250000
            trimTrailingBlanks( sourceRec ); 06260000
            strncat( sourceRec,"\\x25",1 ); 06270000
            strcat( sqlSource.data, sourceRec ); 06280000
            sqlSource.length = strlen(sqlSource.data); 06290000
        }
    /* Throw exception if not enough room for next record ... */ 06300003
    if( moreRecords == Yes && sqlSource.length >((2*1048576)-72) ) 06310000
        issueInvalidParmLengthError( "DD:SQLIN",0,((2*1048576)-72) ); 06320003
    06330003
}
06340000
06350000
06360000

```

```

if( rc < RETSEV )
    if( fclose( sqlInFile ) != 0 )
        issueDataSetClosingError( "DD:SQLIN",errno );
    } /* end of getSQLsource */

void connectToLocation( void )           /* Connect to DB2 location */ 06370000
/* **** */
* Connects to the DB2 location specified in call parm number 4      * 06380000
* **** 06390000
{ EXEC SQL
    CONNECT TO :locationName;
    if( SQLCODE != 0 )
        { issueSqlError( "Connect to location failed" );
    }
} /* end of connectToLocation */ 06400000
06410000
06420000
06430000
06440000
06450000
06460000
06470000
06480000
06490000
06500000
06510000
06520000
06530000
06540000
06550000
06560003
06570003
06580003
06590003
06600003
06610003
06620003
06630003
06640003
06650003
06660003
06670003
06680003
06690003
06700000
06710000
06720000
06730000
06740000
06750000
06760000
06770000
06780000
06790000
06800000
06810000
06820000
06830000
06840000
06850000
06860000
06870000
06880004
06890004
06900000
06910000
06920000
06930000
06940000
06950000
06960000
06970000
06980000
06990000
07000000
07010000
07020000
07030000
07040000
07050000
07060000
07070003
07080003
07090003
07100003
07110003
07120003
07130003
07140003
07150003
07160003
07170000
07180000
} /* end of callDSNTPSMP */

```

```

void listDSNTPSMPcallParms( void ) /* List parms sent to DSNTPSMP*/ 07190000
/****** */
* Displays the arguments of parameters being passed to DSNTPSMP * 07210000
***** */ 07220000
{ printf( "*****\n" ); 07230000
printf( "*****\n" ); 07240000
printf( "*****\n" ); 07250000
"*****\n" ); 07260000
printf( "* DSN8ED4 is now invoking the DB2 SQL Procedures " 07270000
"Processor (SYSPROC.DSNTPSMP)\n" ); 07280000
printf( "*\n" ); 07290000
printf( "* Location name: %s\n", locationName ); 07300000
printf( "*\n" ); 07310000
printf( "* Action specified: %s\n", action ); 07320000
printf( "*\n" ); 07330000
printf( "* SQL Procedure name: %s\n", routineName ); 07340000
printf( "*\n" ); 07350000
printf( "* DB2 Precompiler Options:\n* %s\n", precompOptions ); 07360000
printf( "*\n" ); 07370000
printf( "* Compiler Options:\n* %s\n", compileOptions ); 07380000
printf( "*\n" ); 07390000
printf( "* Prelink Editor Options:\n* %s\n", prelinkOptions ); 07400000
printf( "*\n" ); 07410000
printf( "* Link Editor Options:\n* %s\n", linkOptions ); 07420000
printf( "*\n" ); 07430000
printf( "* DB2 Bind Options:\n* %s\n", bindOptions ); 07440000
printf( "*\n" ); 07450000
if( strlen(alterStmt) > 0 ) /*@02*/ 07460000
{
    /*@02*/ 07470000
    printf( "* ALTER statement:\n* %s\n", alterStmt ); /*@02*/ 07480000
    printf( "*\n" ); 07490000
} /*@02*/ 07500000
07510000
} /* end of listDSNTPSMPcallParms */ 07520000
07530000
07540000
void processDSNTPSMPresultSet( void ) /* Handle DSNTPSMP result sets*/ 07550000
/****** */
* Outputs data from the result set returned by DSNTPSMP * 07560000
***** */ 07570000
07580000
{
/****** */
* Associate a locator with the result set from DSNTPSMP * 07590000
***** */
associateResultSetLocator(); 07600000
07610000
07620000
07630000
07640000
/****** */
* Allocate a cursor for the result set * 07650000
***** */
07660000
07670000
if( rc < RETSEV ) 07680000
    allocateResultSetCursor(); 07690000
07700000
/****** */
* Output reports returned in the result set * 07710000
***** */
07720000
07730000
if( rc < RETSEV ) 07740000
    writeDSNTPSMPreports(); 07750000
07760000
} /* end of processDSNTPSMPresultSet */ 07770000
07780000
07790000
void associateResultSetLocator(void) /* Assoc DSNTPSMP RS locator */ 07800000
/****** */
* Associates the result set from DSNTPSMP with a result set locator* 07810000
***** */ 07820000
07830000
{
EXEC SQL 07840000
ASSOCIATE
LOCATORS( :DSNTPSMP_rs_loc1 )
WITH PROCEDURE SYSPROC.DSNTPSMP;
07850000
07860000
07870000
07880000
if( SQLCODE != 0 ) 07890000
{
issueSqlError( "Associate locator call failed" );
07900000
07910000
07920000
} /* end of associateResultSetLocator */ 07930000
07940000
07950000
void allocateResultSetCursor( void ) /* Alloc DSNTPSMP RS cursor */ 07960000
/****** */
* Allocates a cursor to the locator for the DSNTPSMP result set * 07970000
***** */ 07980000
07990000
{
EXEC SQL 08000000

```

```

ALLOCATE DSNTPSMP_RS_CSR1          08010000
    CURSOR FOR RESULT SET :DSNTPSMP_rs_loc1;      08020000
                                                08030000
if( SQLCODE != 0 )                  08040000
{ issueSqlError( "Allocate result set cursor "
                 "call failed" );
}
}
} /* end of allocateResultSetCursor */

void writeDSNTPSMPReports( void )     /* Print DSNTPSMP report */ 08120000
/********************************************* 08130000
* Outputs the reports returned in the result set from DSNTPSMP * 08140000
***** 08150000
* The result set returned by DSNTPSMP contains one or more reports.* 08160000
*                                         * 08170000
* Within the result set, reports are distinguished from one another* 08180000
* er by the STEP and FILE columns:                                * 08190000
* - STEP refers to the phase (e.g. precompile, compile, bind, etc.)* 08200000
* of DSNTPSMP that generated the report.                         * 08210000
* - FILE distinguishes reports that are generated by the same STEP.* 08220000
*                                         * 08230000
* Report line data are stored in the LINE column, and arranged ac-* 08240000
* cording to the sequence number in the SEQN column.             * 08250000
*                                         * 08260000
* In summary, STEPs contain FILEs, FILEs contain LINEs, and LINEs * 08270000
* are ordered according to SEQN (sequence).                         * 08280000
***** 08290000
{ short int reportNumber = 1;           /* Sequence number of report */ 08300000
char     prevStepName[17];            /* Track step name changes */ 08310000
char     prevFileName[9];             /* Track file name changes */ 08320000
short int recordLength = 0;           /* Length of record */ 08330000
                                         08340000
/********************************************* 08350000
* Get the first entry in the result set                          * 08360000
***** 08370000
fetchFromResultSetCursor();          08380000
                                         08390000
/********************************************* 08400000
* Allocate an output DD for the first report                   * 08410000
***** 08420000
if( rc < RETSEV )                08430000
    openReportDataSet( reportNumber );
                                         08440000
                                         08450000
/********************************************* 08460000
* Save step and file, to monitor for when they change        * 08470000
***** 08480000
if( rc < RETSEV )                08490000
{ strncpy( prevStepName,stepName,17 );
    strncpy( prevFileName,fileName,9 );
}
                                         08500000
                                         08510000
                                         08520000
                                         08530000
/********************************************* 08540000
* Process all rows in the result set                          * 08550000
***** 08560000
while( SQLCODE == 0 && rc < RETSEV ) 08570000
{ if( ( strcmp( prevStepName,stepName ) != 0
        || strcmp( prevFileName,fileName ) != 0 )
    && reportNumber < 6 )           /*@06*/ 08580005
    /********************************************* 08600005
     * If the step or file changes, allocate next report DD      * 08620000
     * up to and including report no. 6 @06* 08630005
***** 08640000
    { closeReportDataSet();
        if( rc < RETSEV )
            openReportDataSet( ++reportNumber );
        if( rc < RETSEV )
            { strncpy( prevStepName,stepName,17 );
                strncpy( prevFileName,fileName,9 );
            }
    }
}
/********************************************* 08730000
* Write the current report line to the current report DD      * 08740000
***** 08750000
if( rc < RETSEV )                08760000
{ recordLength
    = fwrite( reportLine, /* write from reportLine */ 08780000
              1,           /* ..a record */ 08790000
              sizeof( reportLine ), 08800000
              reportDD ); /* ..into the report data set */ 08810000
}
                                         08820000

```

```

        if( rc < RETSEV )
            { fetchFromResultSetCursor(); }
    }

    if( rc < RETSEV )
        { closeReportDataSet(); }

} /* end of writeDSNTPSPMReports */

void fetchFromResultSetCursor( void ) /* Read DSNTSPMP RS cursor */ 08830000
/****** */ 08840000
* Reads the cursor for the DSNTSPMP result set * 08850000
***** 08860000
{ memset( reportLine,' ',256 ); 08870000
08880000
08890000
08900000
08910000
08920000
08930000
08940000
08950000
08960000
08970000
08980000
08990000
09000000
09010000
09020000
09030000
09040000
09050000
09060000
09070000
09080000
09090000
09100000
09110000
09120000
09130000
09140000
09150000
09160000
09170000
09180000
09190000
09200000
09210000
09220000
09230000
09240000
09250000
09260000
09270000
09280000
09290000
09300000
09310000
09320000
09330000
09340000
09350000
09360000
09370000
09380000
09390000
09400000
09410000
09420000
09430000
09440000
09450000
09460000
09470000
09480000
09490000
09500000
09510000
09520000
09530000
09540000
09550000
09560000
09570000
09580000
09590000
09600000
09610000
09620000
09630000
09640000

```

```

{ int i; 09650000
  for( i = strlen(string) - 1; string[i] == ' ' ; i-- );
  string[++i] = '\0';
} /* end of trimTrailingBlanks */ 09660000
                                         09670000
                                         09680000
                                         09690000
                                         /*begin @03*/
void stripContinuationCharacter /* Strip off trailing - or + */
( char *string /* - in: string to be trimmed */ 09710000
)
/****** 09720000
* Strips trailing '+' or '-' from a blank-trimmed string 09730000
***** 09740000
***** 09750000
***** 09760000
{ int i; 09770000
  i = strlen(string) - 1; 09780000
  if( string[i] == '+' || string[i] == '-' ) 09790000
    string[i] = '\0';
  trimTrailingBlanks( string );
} /* end of trimstripContinuationCharacter */ 09810000
                                         09820000
                                         /*end @03*/ 09830000
                                         09840000
                                         /*begin @04*/
void processSqlCommit( void ) /* Commit SQL changes */ 09850000
/****** 09860000
* Commits the current unit of SQL work 09870000
***** 09880000
***** 09890000
{ EXEC SQL 09900000
  COMMIT; 09910000
          09920000
  if( SQLCODE != 0 ) 09930000
    { issueSqlError( "*** Commit failed " ); 09940000
    }
  else 09950000
    { printf( "* SQL changes have been committed\n" ); 09960000
    }
} /* end of processSqlCommit */ 09970000
                                         09980000
                                         09990000
                                         10000000
                                         10010000
                                         10020000
void processSqlRollback( void ) /* Rollback SQL changes */ 10030000
/****** 10040000
* Rolls back the current unit of SQL work 10050000
***** 10060000
***** 10070000
{ EXEC SQL 10080000
  ROLLBACK; 10090000
  if( SQLCODE != 0 ) 10100000
    { issueSqlError( "*** Rollback failed " ); 10110000
    }
  else 10120000
    { printf( "* SQL changes have been rolled back\n" ); 10130000
    }
} /* end of processSqlRollback */ 10140000
                                         10150000
                                         10160000
                                         10170000
                                         /*end @04*/ 10180000
                                         10190000
void issueDataSetClosingError /* Handler for ds close error */ 10200000
( char *DDname, /* - in: name of errant DD */ 10210000
  int LEerrno /* - in: LE diagnostic errno */ 10220000
)
/****** 10230000
* Called when a TSO data set cannot be closed 10240000
***** 10250000
***** 10260000
{ printf( "ERROR: Unable to close %s\n", DDname ); 10270000
  printf( "%s \n", strerror(LEerrno) );
  printf( "-----> Processing halted\n" );
  rc = RETSEV;
} /* end of issueDataSetClosingError */ 10280000
                                         10290000
                                         10300000
                                         10310000
                                         10320000
                                         10330000
void issueDataSetOpeningError /* Handler for ds open error */ 10340000
( char *DDname, /* - in: name of errant DD */ 10350000
  int LEerrno /* - in: LE diagnostic errno */ 10360000
)
/****** 10370000
* Called when a TSO data set cannot be opened 10380000
***** 10390000
***** 10400000
{ printf( "ERROR: Unable to open %s\n", DDname );
  printf( "%s \n", strerror(LEerrno) );
  printf( "-----> Processing halted\n" );
  rc = RETSEV;
} /* end of issueDataSetOpeningError */ 10410000
                                         10420000
                                         10430000
                                         10440000
                                         10450000
                                         10460000

```

```

void issueDataSetReadingError      /* Handler for ds read error */ 10470000
( char          *DDname,           /* - in: name of errant DD */ 10480000
  int           LEerrno           /* - in: LE diagnostic errno */ 10490000
)
/* **** */
* Called when a TSO data set cannot be read                         * 10500000
***** 10510000
{ printf( "ERROR: Unable to read %s\n", DDname );                  10520000
  printf( "%s\n", strerror(LEerrno) );
  printf( "-----> Processing halted\n" );
  rc = RETSEV;
} /* end of issueDataSetReadingError */                                10530000
10540000
10550000
10560000
10570000
10580000
10590000
10600000
10610000
void issueInvalidCallParmCountError /* Handler for parm count err */ 10620000
( int argc             /* - in: no. parms received */ 10630000
)
/* **** */
* Called when this program is invoked with an inappropriate number * 10640000
* of call parms.                                                 * 10650000
***** 10660000
* DSN8ED4 was invoked with %i parameters\n",--argc ); 10670000
{ printf( "-----> The first three parms (action, routine "
  "name, and authid) are required\n" );
  printf( "-----> The fourth parm (location name "
  "is optional\n" );
  printf( "-----> Processing halted\n" );
  rc = RETSEV;
} /* end of issueInvalidCallParmCountError */                            10680000
10690000
10700000
10710000
10720000
10730000
10740000
10750000
10760000
10770000
10780000
void issueInvalidDDnumError      /* Handler for unknown DD seq */ 10790000
( short        invalidDDnum    /* - in: invalid DD sequ. no. */ 10800000
)
/* **** */
* Called when the sequence number for a report DD (REPORTnn, where * 10810000
* "nn" is the sequence number" is less than 1 or greater 99.     * 10820000
***** 10830000
* "number <%i specified " /* ..for DD REPORTnn */ 10840000
  "for REPORTnn DD\n",
  /* ..where nn is the sequence */ 10850000
  invalidDDnum );          /* ..number of the result set */
printf( "-----> Processing halted\n" );
rc = RETSEV;
} /* end of issueInvalidDDnumError */                                    10860000
10870000
10880000
10890000
10900000
10910000
10920000
10930000
10940000
void issueInvalidActionError     /* Handler for unknown action */ 10950000
( char *action            /* - in: action specified */ 10960000
)
/* **** */
* Called when an unexpected argument is specified for the DB2 SQL * 10970000
* Procedures Processor action                                         * 10980000
***** 10990000
{ printf( "-----> The argument specified for the action "
  "parameter is invalid\n",action );
  printf( "-----> Processing halted\n" );
  rc = RETSEV;
} /* end of issueInvalidActionError */                                 11010000
11020000
11030000
11040000
11050000
11060000
11070000
11080000
void issueInvalidParmLengthError /* Handler for parm len error */ 11090000
( char *parmName,           /* - in: identify of parm */ 11100000
  int minLength,            /* - in: min valid length */ 11110000
  int maxLength             /* - in: max valid length */ 11120000
)
/* **** */
* Called when the length of an argument specified for a DSNTPSMP * 11130000
* parameter (parmName) does not fall within the valid bounds for * 11140000
* size (minLength and maxLength) for that parameter                 * 11150000
***** 11160000
{ printf( "-----> The length of the argument specified for the %s "
  "parameter\n",parmName );
  printf( "-----> does not fall within the required bounds of %i "
  "and %i\n",minLength,maxLength );
  printf( "-----> Processing halted\n" );
  rc = RETSEV;
} /* end of issueInvalidParmLengthError */                             11170000
11180000
11190000
11200000
11210000
11220000
11230000
11240000
11250000
11260003
11270003
void issueInvalidLevelError      /* Handler for wrong DSNTPSMP */ 11280003

```

```

( char *level                  /* - in: level encountered */ 11290003
)
/********************************************* 11300003
* Called when a DSNTPSMP QUERYLEVEL request returns a level not      * 11320003
* handled by this sample client.                                         * 11330003
****************************************** 11340003
{ printf( "ERROR: The DSNTPSMP interface level %s is not "          11350003
      "supported by this client\n",level );                                11360003
  printf( "----> Processing halted\n" );                                 11370003
  rc = RETSEV;                                                       11380003
} /* end of issueInvalidLevelError */                                    11390003
                                                11400000
                                                11410000
11420000
#pragma linkage(dsntiar, OS)
void issueSqlError             /* Handler for SQL error */    11430000
( char *locMsg                 /* - in: Call location */   11440000
)
/********************************************* 11450000
* Called when an unexpected SQLCODE is returned from a DB2 call      * 11470000
****************************************** 11480000
{ struct error_struct {        /* DSNTIAR message structure */ 11490000
    short int error_len;           11500000
    char     error_text[10][80];    11510000
}     error_message = {10 * 80};                                11520000
                                                11530000
extern short int dsntiar( struct      sqlca      *sqlca,          11540000
                         struct      error_struct *msg,          11550000
                         int            *len );          11560000
                                                11570000
short int  DSNTIARrc;          /* DSNTIAR Return code */ 11580000
int      j;                  /* Loop control */        11590000
static int lrecl = 80;         /* Width of message lines */ 11600000
                                                11610000
/********************************************* 11620000
* print the locator message                                         * 11630000
****************************************** 11640000
printf( "ERROR: %-8os\n", locMsg );
printf( "----> Processing halted\n" );                                11660000
                                                11670000
/********************************************* 11680000
* format and print the SQL message                                * 11690000
****************************************** 11700000
DSNTIARrc = dsntiar( &sqlca, &error_message, &lrecl );
if( DSNTIARrc == 0 )                                                 11710000
    if( DSNTIARrc == 0 )
        for( j = 0; j <= 10; j++ )
            printf( "%.8os\n", error_message.error_text[j] );
    else
    {
        printf( " *** ERROR: DSNTIAR could not format the message\n" );11770000
        printf( " ***      SQLCODE is %d\n",SQLCODE );                11780000
        printf( " ***      SQLERRM is \n" );                            11790000
        for( j=0; j<sqlca.sqlerrml; j++ )
            printf( "%c", sqlca.sqlerrmc[j] );
        printf( "\n" );
    }
}
/* set severe error code                                         * 11860000
****************************************** 11870000
rc = RETSEV;
} /* end of issueSqlError */                                     11880000
                                                11890000
                                                11900000

```

Referencia relacionada

["Programas de ejemplo para ayudar a preparar y ejecutar procedimientos SQL externos"](#) en la página 323
Db2 proporciona trabajos de ejemplo para ayudarle a preparar y ejecutar procedimientos SQL externos.
Todos los ejemplos están en el conjunto de datos de DSN1210.SDSNSAMP. Antes de poder ejecutar los ejemplos, debe personalizarlos para la instalación.

DSN8WLMP

Este JCL se puede personalizar para establecer el PROC de inicio de WLM necesario para ejecutar DSNTPSMP, el procesador de procedimientos SQL de Db2 , y para ejecutar ADMIN_UPDATE_SYSPARM, el procedimiento almacenado de Db2 , que cambia los parámetros del subsistema.

```

//*****
//**  Name = DSN8WLMP
//**

```

```

/** Descriptive Name =
/** DB2 Sample WLM startup PROC for DSNTSPMP, the DB2 SQL Procedures
/** Processor, and for ADMIN_UPDATE_SYSPARM, the DB2 stored
/** procedure that changes subsystem parameters.
/** 
/** Licensed Materials - Property of IBM
/** 5635-DB2
/** (C) COPYRIGHT 1982, 2006 IBM Corp. All Rights Reserved.
/** 
/** STATUS = Version 11
/** 
/** Function =
/** This JCL can be customized to establish the WLM startup PROC
/** needed to run DSNTSPMP, the DB2 SQL Procedures Processor,
/** and to run ADMIN_UPDATE_SYSPARM, the DB2 stored procedure
/** that changes subsystem parameters.
/** 
/** Before you can use this procedure, you need to have defined a
/** WLM Application Environment for running DSNTSPMP and
/** ADMIN_UPDATE_SYSPARM.
/** 
/** *** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
/** For DSNTSPMP and ADMIN_UPDATE_SYSPARM, NUMTCB=1 is required.
/** Specify no other value. This assures concurrent executions
/** of DSNTSPMP and ADMIN_UPDATE_SYSPARM will run in their
/** own address space, which is needed for proper dataset
/** operation from within a REXX/TSO DB2 stored procedure.
/** 
/** (1) Customize this proc for use on your system by locating and
/** changing all occurrences of the following strings as
/** indicated:
/** (A) '!WLMENV!' to the name of the WLM Application Environment
/**      you have chosen for running DSNTSPMP and
/**      ADMIN_UPDATE_SYSPARM
/** (B) '!DSN8WLMP!' to the name of the WLM Procedure associated
/**      with that environment
/** (C) '!DSN!' to the name of your DB2 subsystem
/** (D) 'CBC!!' to the prefix of your target library for
/**      IBM C/C++ for z/OS
/** (E) 'CEE!!' to the prefix of your target library for
/**      IBM Language Environment for z/OS
/** (F) 'DSN!!0' to the prefix of your target library for
/**      DB2 for z/OS
/** 
/** (2) Copy the customized proc to your MVS proclib, to the member
/**      you specified as the WLM procedure name for the WLM
/**      application environment you have chosen for running DSNTSPMP
/**      and ADMIN_UPDATE_SYSPARM
/** Note: This should be the same value as you specified in
/**      step 1B, above.
/** 
/** CHANGE LOG:
/** 09/20/2012 Add ZPMDFLTS for ADMIN_UPDATE_SYSPARM DK1557/PM71114
/** 
/** ****
/** !DSN8WLMP! PROC DB2SSN!=DSN!,NUMTCB=1,APPLENv!=WLMENV!
/** 
//*/NUMTCB@1 SET NUMTCB=                                     <== Null NUMTCB symbol
//*/ 
//DSNTSPMP EXEC PGM=DSNX9WLM,TIME=1440,
//          PARM='&DB2SSN,1,&APPLENv',           <== Use 1, not NUMTCB
//          REGION=0M,DYNAMNBR=5                <== Allow for Dyn Allocs
//** Include SDSNEXIT to use Secondary Authids (DSN3@ATH DSN3@SGN exits)
//STEPLIB  DD DISP=SHR,DSN=DSN!!0.SDSNEXIT
//          DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
//          DD DISP=SHR,DSN=CBC!!..SCCNCMP      <== C Compiler
//          DD DISP=SHR,DSN=CEE!!..SCEERUN       <== LE runtime
//SYSEXEC  DD DISP=SHR,                               <== Location of DSNTSPMP
//          DSN=DSN!!0.SDSNCLST                  and DSNADMUZ
//SYSTSPRT DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSABEND DD DUMMY
//DSNTRACE DD SYSOUT=*
//** Data sets required by the SQL Procedures Processor
//SQLDBRM  DD DISP=SHR,                               <== DBRM Library
//          DSN=DSN!!0.DBRLMLIB.DATA
//SQLCSRC  DD DISP=SHR,                               <== Generated C Source
//          DSN=DSN!!0.SRCLIB.DATA
//SQLMOD   DD DISP=SHR,                               <== Application Loadlib
//          DSN=DSN!!0.RUNLIB.LOAD

```

```

//SQLLIBC DD DISP=SHR,                                <== C header files
//          DSN=CEE!!_.SCEEH.H
//          DD DISP=SHR,
//          DSN=CEE!!_.SCEEH.SYS.H
//          DD DISP=SHR,                                <== Debug header file
//          DSN=DSN!!0.SDSNC.H
//SQLLIBL DD DISP=SHR,                                <== Linkedit includes
//          DSN=CEE!!_.SCEELKED
//          DD DISP=SHR,
//          DSN=DSN!!0.SDSNLOAD
//SYSMSGS DD DISP=SHR,                                <== Prelinker msg file
//          DSN=CEE!!_.SCEEMSGP(EDCPMSGE)
///*
//***** DSNTPSMP Configuration File - CFGTPSMP (optional)
//*          A site provided sequential dataset or member, used to
//*          define customized operation of DSNTPSMP in this APPLENV.
//*/CFGTPSMP DD DISP=SHR,DSN=
///*
//***** Workfiles required by the SQL Procedures Processor
//SQLSRC  DD UNIT=SYSALLDA,SPACE=(23440,(20,20)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLPRINT DD UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLTERM  DD UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLOUT   DD UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLCPRT  DD UNIT=SYSALLDA,SPACE=(23476,(20,20)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=23476)
//SQLUT1   DD UNIT=SYSALLDA,SPACE=(23440,(20,20)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLUT2   DD UNIT=SYSALLDA,SPACE=(23440,(20,20)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
//SQLCIN   DD UNIT=SYSALLDA,SPACE=(32000,(20,20))
//SQLLIN   DD UNIT=SYSALLDA,SPACE=(3200,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SQLDUMMY DD DUMMY
//SYSMOD   DD UNIT=SYSALLDA,SPACE=(23440,(20,20)),    <= PRELINKER
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=23440)
///*
//***** Data sets required by ADMIN_UPDATE_SYSPARM
//ZPMDFLTS DD DISP=SHR,                                <== Defaults file
//          DSN=DSN!!0.NEW.SDSNSAMP(DSNADMZW)
//*/

```

Referencia relacionada

["Programas de ejemplo para ayudar a preparar y ejecutar procedimientos SQL externos"](#) en la página 323
 Db2 proporciona trabajos de ejemplo para ayudarle a preparar y ejecutar procedimientos SQL externos.
 Todos los ejemplos están en el conjunto de datos de DSN1210.SDSNSAMP. Antes de poder ejecutar los ejemplos, debe personalizarlos para la instalación.

DSN8ED5

Demuestra cómo llamar al procedimiento SQL de muestra DSN8.

```

***** 00010000
* Module name = DSN8ED5 (DB2 sample program)      * 00020000
*                                                 * 00030000
* DESCRIPTIVE NAME = Client for sample SQL Procedure DSN8.DSN8ES2 * 00040000
*                                                 * 00050000
*                                                 * 00060000
* LICENSED MATERIALS - PROPERTY OF IBM           * 00070000
* 5675-DB2                                         * 00080000
* (C) COPYRIGHT 1999, 2000 IBM CORP. ALL RIGHTS RESERVED. * 00100000
*                                                 * 00130000
* STATUS = VERSION 7                             * 00140000
*                                                 * 00170000
* Function: Demonstrates how to call the sample SQL procedure * 00230000
*             DSN8.DSN8ES2 using static SQL.        * 00240000
*                                                 * 00250000
* Notes:                                           * 00260000
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher * 00270000
*                                                 * 00280000
* Restrictions:                                    * 00290000
*                                                 * 00300000
* Module type: C program                         * 00310000
* Processor: IBM C/C++ for OS/390 V1R3 or higher * 00320000
* Module size: See linkedit output                * 00330000
* Attributes: Re-entrant and re-usable           * 00340000

```

```

/*
 * Entry Point: DSN8ED5                                * 00350000
 * Purpose: See Function                               * 00360000
 * Linkage: Standard MVS program invocation, one parameter. * 00370000
 *
 *
 * Parameters: DSN8ED5 uses the C "main" argument convention of * 00380000
 *              argv (argument vector) and argc (argument count). * 00390000
 *              *
 *              - ARGV[0]: (input) pointer to a char[9],           * 00400000
 *                            null-terminated string having the name of * 00410000
 *                            this program (DSN8ED5)                      * 00420000
 *              - ARGV[1]: (input) pointer to a char[10],           * 00430000
 *                            null-terminated string that contains the * 00440000
 *                            amount of the base bonus for sample   * 00450000
 *                            managers. The format is: nnnnnn.nn      * 00460000
 *              - ARGV[2]: (input) pointer to a char[17],           * 00470000
 *                            null-terminated string having the name of * 00480000
 *                            the server where DSN8.DSN8ES2 resides.    * 00490000
 *                            This is an optional parameter; the local   * 00500000
 *                            server is used if no argument is provided. * 00510000
 *
 * Normal Exit: Return Code: 0000                      * 00520000
 *              - Message: none                           * 00530000
 *
 * Error Exit: Return Code: 0008                      * 00540000
 *              - Message: DSN8ED5 failed: Invalid parameter count * 00550000
 *              - Message: DSN8ED5 failed: Argument to parameter 1   * 00560000
 *                            exceeds 9 bytes                  * 00570000
 *              - Message: DSN8ED5 failed: No result from DSN8.DSN8ES2 * 00580000
 *              - Message: <formatted SQL text from DSNTIAR>        * 00590000
 *
 *
 * External References:                                * 00600000
 *              - Routines/Services: DSNTIAR: DB2 msg text formatter * 00610000
 *              - Data areas       : None                     * 00620000
 *              - Control blocks  : None                     * 00630000
 *
 *
 * Pseudocode:
 * DSN8ED5:
 *              - Verify that 2 or 3 input parameters (program name, base bonus * 00640000
 *                            amount and, optionally, remote location name) were passed. * 00650000
 *              - if not, issue diagnostic message and end with code 0008      * 00660000
 *              - Connect to the remote location, if one was specified          * 00670000
 *              - Call sample SQL Procedure DSN8.DSN8ES2, passing the base bonus * 00680000
 *                            amount as the argument of the first (input) parameter. * 00690000
 *              - if unsuccessful, call sql_error to issue a diagnostic mes- * 00700000
 *                            sage, then end with code 0008.                         * 00710000
 *              - Report the value returned by DSN8.DSN8ES2 in its second * 00720000
 *                            (output) parameter.                                * 00730000
 * End DSN8ED5                                         * 00740000
 *
 * sql_error:
 *              - call DSNTIAR to format the unexpected SQLCODE.            * 00750000
 * End sql_error                                         * 00760000
 *
 **** C library definitions ****/
#include <stdio.h>                                     00910000
#include <stdlib.h>                                      00920000
#include <string.h>                                       00930000
#include <decimal.h>                                     00940000
                                         00950000
                                         00960000
                                         00970000
**** Equates ****/
#define NULLCHAR '\0' /* Null character */             00980000
                                         00990000
                                         01000000
#define OUTLEN     80 /* Length of output line */        01010000
#define DATA_DIM   10 /* Number of message lines */      01020000
                                         01030000
#define NOT_OK     0 /* Run status indicator: Error */  01040000
#define OK         1 /* Run status indicator: Good */   01050000
                                         01060000
                                         01070000
**** DB2 SQL Communication Area ****/
EXEC SQL INCLUDE SQLCA;
                                         01080000
                                         01090000
                                         01100000
                                         01110000
**** DB2 Host Variables ****/
EXEC SQL BEGIN DECLARE SECTION;
char      locationName[17]; /* Server location name */ 01120000
                                         01130000
decimal(15,2) hvBonusBase = 0; /* base bonus for managers */ 01140000
                                         01150000
                                         01160000

```

```

short int    niBonusBase      = 0; /* Indic var for hvBonusBase */ 01170000
                                                01180000
decimal(15,2) hvBonuses       = 0; /* tot bonuses rtnd by DSN8ES2*/ 01190000
short int    niBonuses        = 0; /* Indic var for hvBonuses */ 01200000
                                                01210000
long int     hvSqlErrCd      = 0; /* Err SQLCODE from DSN8ES2 */ 01220000
short int    niSqlErrCd      = 0; /* Indic var for hvSqlErrCd */ 01230000
                                                01240000
EXEC SQL END DECLARE SECTION;                                01250000
                                                01260000
                                                01270000
***** DB2 Message Formatter *****
struct        error_struct    /* DSNTIAR message structure */ 01280000
{
    short int   error_len;
    char        error_text[DATA_DIM][OUTLEN];
}                error_message = {DATA_DIM * (OUTLEN)};           01300000
                                                01310000
#pragma        linkage( dsntiar, OS )                           01320000
                                                01330000
                                                01340000
extern short int dsntiar( struct      sqlca      *sqlca,
                          struct      error_struct *msg,
                          int          *len );           01350000
                                                01360000
                                                01400000
                                                01410000
***** DSN8ED5 Global Variables *****
short int     status = OK; /* DSN8ED5 run status */ 01420000
                                                01430000
                                                01440000
long int      completion_code = 0; /* DSN8ED5 return code */ 01450000
                                                01460000
                                                01470000
***** DSN8ED5 Function Prototypes *****
int main( int argc, char *argv[] );
void sql_error( char locmsg[] );
                                                01480000
int main( int argc, char *argv[] )                           01490000
/* Get input parms, pass them to DSN8ES2, and process the results */ 01500000
***** *****
{                                         01510000
    printf( "**** DSN8ED5: Sample client for DB2 SQL Procedure Sample "
            "(DSN8.DSN8ES2)\n\n" );           01520000
    printf( "*\n" );
    if( argc < 2 || argc > 3 )           01530000
    {
        printf( "DSN8ED5 failed: Invalid parameter count\n" );
        status = NOT_OK;                  01540000
    }
    else if( strlen(argv[1]) > 9 )
    {
        printf( "DSN8ED5 failed: Bonus base exceeds 9 bytes. "
                "Use format: nnnnn.nn\n" );
        status = NOT_OK;                  01550000
    }
    else
    { /* Convert the input parameter from a string to a decimal */
        hvBonusBase = atof( argv[1] );
    }
    /* Validate remote location name, if one is specified */
    if( argc == 3 && status == OK )
    {
        if( strlen( argv[2] ) < 1 || strlen( argv[2] ) > 16 )
        {
            printf( "DSN8ED5 failed: Length of location name must be "
                    "1 to 16 bytes\n" );
            status = NOT_OK;
        }
        else
        {
            strcpy( locationName,argv[2] );
            printf( "* Processing at location: %s\n",locationName );
            printf( "*\n" );
        }
    }
    else
    {
        locationName[0] = NULLCHAR;
    }
    if( status == OK )
    {

```

```

        printf( "* Base bonus amount: %D(15,2)\n",hvBonusBase );          01990000
        printf( "*\n" );                                              02000000
    }

/***** Connect to the remote location, if one was specified *****/
* Connect to the remote location, if one was specified           * 02040000
***** Process the call to DSN8.DSN8ES2                           * 02140000
***** Process the call to DSN8.DSN8ES2                           / 02150000
if( status == OK )
{
    EXEC SQL CONNECT TO :locationName;                            02080000
    if( SQLCODE != 0 )                                           02090000
        sql_error( " *** Connect to remote server" );           02100000
}

/***** Call DSN8.DSN8ES2 *****/
* Call DSN8.DSN8ES2                                            * 02130000
***** Call DSN8.DSN8ES2                                            / 02160000
if( status == OK )
{
    EXEC SQL CALL DSN8.DSN8ES2( :hvBonusBase :niBonusBase,          02180000
                                :hvBonuses  :niBonuses,             02190000
                                :hvSqlErrCd :niSqlErrCd );          02200000
    if( SQLCODE != 0 )                                           02210000
        sql_error( " *** Call DSN8.DSN8ES2" );                  02220000
    else if( niSqlErrCd == 0 )
    {
        printf( "DSN8ED5 failed: Error SQLCODE from DSN8.DSN8ES2 " 02250000
                "is %i\n", hvSqlErrCd );                         02260000
        status = NOT_OK;                                         02270000
    }
    else if( niBonuses != 0 )
    {
        printf( "DSN8ED5 failed: No result from DSN8.DSN8ES2\n" ); 02310000
        status = NOT_OK;                                         02320000
    }
    else
    {
        printf( "* Total bonuses paid to management: $%D(15,2)\n",   02360000
                hvBonuses );                                     02370000
    }
}

if( status != OK )
    completion_code = 8;

return( completion_code );
} /* end main */
}

/***** SQL error handler *****/
** SQL error handler                                               ** 02510000
***** SQL error handler                                           / 02520000
void sql_error( char locmsg[] )                                     /*proc*/ 02540000
{
    short int    rc;                                              /* DSNTIAR Return code      */ 02580000
    int         j,k;                                              /* Loop control              */ 02590000
    static int   lrecl = OUTLEN;                                    /* Width of message lines */ 02600000
                                                               02610000

/***** Set status to prevent further processing *****/
* set status to prevent further processing                         * 02630000
***** Set status to prevent further processing                   / 02640000
status = NOT_OK;                                                 02650000
                                                               02660000

/***** Print the locator message *****/
* print the locator message                                       * 02680000
***** Print the locator message                                 / 02690000
printf( ".%8s\n", locmsg );                                         02700000
                                                               02710000

/***** Format and print the SQL message *****/
* format and print the SQL message                             * 02730000
***** Format and print the SQL message                         / 02740000
rc = dsntiar( &sqlca, &error_message, &lrecl );                 02750000
if( rc == 0 )
    for( j=0; j<DATA_DIM; j++ )
    {
        for( k=0; k<OUTLEN; k++ )
            putchar(error_message.error_text[j][k] );
}

```

```

        putchar('\n');
    }
else
{
    printf( " *** ERROR: DSNTIAR could not format the message\n" );
    printf( " ***      SQLCODE is %d\n",SQLCODE );
    printf( " ***      SQLERRM is \n" );
    for( j=0; j<sqlca.sqlerrml; j++ )
        printf("%c", sqlca.sqlerrmc[j] );
    printf( "\n" );
}
} /* end of sql_error */

```

Referencia relacionada

["Programas de ejemplo para ayudar a preparar y ejecutar procedimientos SQL externos"](#) en la página 323
 Db2 proporciona trabajos de ejemplo para ayudarle a preparar y ejecutar procedimientos SQL externos.
 Todos los ejemplos están en el conjunto de datos de DSN1210.SDSNSAMP. Antes de poder ejecutar los ejemplos, debe personalizarlos para la instalación.

DSNTEJ67

Este trabajo muestra dos pasos importantes a seguir cuando se considera la conversión de un procedimiento SQL externo a un procedimiento SQL nativo.

```

//*****
//**  Name = DSNTEJ67
//**
//**  Descriptive Name =
//**      DB2 Sample Application
//**      Phase 6
//**      REXX and SQL PL
//**
//**      Licensed Materials - Property of IBM
//**      5615-DB2
//**      (C) COPYRIGHT 2010, 2013 IBM Corp. All Rights Reserved.
//**
//**      STATUS = Version 12
//**
//**  Function = This job demonstrates two important steps to follow when
//**              considering the conversion of an external SQL procedure
//**              to a native SQL procedure. It all begins with a copy of
//**              the external SQL procedure source:
//**                  1) Modify the SQL procedure options in the source
//**                      a) REMOVE options that relate only to external
//**                         SQL procedures
//**                      b) ADD native SQL PL options that relate to DB2
//**                         precompiler options
//**                      c) ADD native SQL PL options that relate to DB2
//**                         BIND PACKAGE options
//**                  2) Review the SQL procedure source logic. Address
//**                     any identified syntax issues or published semantic
//**                     incompatibilities.
//**
//**  Pseudocode =
//**      This sample assists in this activity by performing the following:
//**
//**      PH067S00 Step
//**          Define the DB2 SSID to use for this job.
//**      PH067S01 Step
//**          Define Input. Identify the name of an external SQL SP with
//**                      source saved in DB2 (SYSIBM.SYSROUTINES_SRC).
//**      PH067S02 Step
//**          Define Output. Specify an output data set where the extracted
//**                      and modified SQL SP source is to be placed.
//**      PH067S03 Step
//**          Setup the sample REXX services to used for this job.
//**      PH067S04 Step
//**          Execute the DSNTEJ67 sample conversion process.
//**              - Validate the SP name           (using the NAMPARTS service)
//**              - Verify the output file is usable (using the CHKANYFV service)
//**              - Deploy DSN8EN1, a sample native
//**                  SQL SP helper for use later (using the CRSQLPL service)
//**              - Extract external SQL SP source (using the SQLPLSRC service)
//**              - Save the source in the output file
//**                  - for a RECFM V output file   (using the ANY2SQLV service)
//**                  - for a RECFM F output file   (using the SQLV2F service)
//**              - Validate and inspect the source (using the CHKSQPL service)

```

```

/*
   - Produce a table of contents to
   describe the DDL syntax elements
   present in the SQL PL source      (using the SQLPLTOC service)
/*
   - Dissect the external SQL SP source
   removing all the SP options
/*
   - Get the replacement options for
   native SQL PL use by calling the
   helper SQL SP deployed earlier    (using the SQLCALL  service)
/*
   - Reassemble the SQL SP source as a
   string and write it to a RECFM V
   temporary file (aka SQLV)         (using the STR2SQLV service)
/*
   - Update the output file
   - for a RECFM V output          (using the ANY2SQLV service)
   - for a RECFM F output          (using the SQLV2F   service)
/*
   - Write a special format temp file
   (aka s80) for the precompiler     (using the SQLV2F   service)
/*
   - Obtain a HOST(SQLPL) Checkout
   precompiler listing of the SQL SP
   source for job log output.       (using the CHKSQLPL service)
/*
   - Set the Job step RC.
*/
/*
*/
/*
Dependencies =
(1) Run sample job DSNTEJ65 prior to running this job.
That job uses the DB2 SQL procedure processor DSNTPSMP to
deploy the sample external SQL procedure DSN8.DSN8ES2, which
is the external SQL Procedure this job processes by default.
*/
/*
Note: Run this job at the same site where DSNTEJ65 created
DSN8.DSN8ES2. Otherwise this job will terminate in
job step PH067S04 with rc=8 and the following
messages:
*SQLPLSRC* Error obtaining Source, SQLPL procedure
was not found
*SQLPLSRC* RC=6
DSNTEJ67 Cannot extract SQL procedure source
*/
/*
Notes =
Prior to running this job, customize it for your system:
(1) Add a valid job card
(2) Locate and change all occurrences of the following strings
as indicated:
(A) '!DSN!'      to the subsystem name of your DB2. This is
located in Step 0.
(B) 'DSN!!0'      to the prefix of the target library for the
current DB2 release. This is located in the
JOBLIB, Step 3 and Step 4.
(3) (Optional) Change either of the following to customize the
input and output of this job for your particular purposes:
*/
/*
(A) Change the name of the external SQL procedure to be
processed by this job. This is defined in job Step 1.
The name must include the schema qualifier. It must
designate an operational external SQL SP which was
deployed using the DB2 SQL procedure processor DSNTPSMP.
*/
/*
(B) Change the data set where the extracted and modified
SQL procedure source will be written. This is defined in
job Step 2. The specification can be for an existing
data set or data set member, qualified or not qualified.
or represented by a DD descriptor (in the form of
DD:ddname). Any existing data set or ddname allocation
must be for a RECFM=F,FB,V,VB sequential data set or
data set member. (If an unallocated ddname is provided
a temporary sequential data set will be allocated.)
*/
/*
Change Activity = ( V11 base pm76443 )
*/
Apr2013 - Add version activation of native SQL PL helper routine
Aug2013 - Clarify prolog notes on DSN8ES2 dependency      PM92730
*/
*****
//JOBLIB    DD DISP=SHR,DSN=DSN!!0.SDSNEXIT
//          DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
/*
*****
//* Step 0: Store the default DB2 System SSID in a temporary data set
//*          for use by various steps and services that are run.
//*****
//PH067S00 EXEC PGM=IEBGENER
//SYSUT1   DD *      Enter the desired DB2 SSID or Group attachment name
!DSN!
//SYSUT2   DD DSN=&&PARM0,DISP=(NEW,PASS),SPACE=(TRK,1),

```

```

//          DCB=(LRECL=80,RECFM=FB,BLKSIZE=160)
//SYSPRINT DD DUMMY
//SYSIN  DD DUMMY
//*****Step 1: Store the desired external SQL SP name to processs.
//*          Specify a fully qualified SP name (2-parts, schema+name).
//*****PH067S01 EXEC PGM=IEBGENER
//SYSUT1  DD *           For a long name, wrap the input at column 72
//          DSN8.DSN8ES2
//*****-----1-----2-----3-----4-----5-----6-----7-----
//SYSUT2  DD DSN=&&PARML,DISP=(NEW,PASS),SPACE=(TRK,1),
//          DCB=(LRECL=72,RECFM=FB,BLKSIZE=576)
//SYSPRINT DD DUMMY
//SYSIN  DD *
//          GENERATE MAXFLDS=1
//          RECORD FIELD=(72)
//*****Step 2: Identify the desired data set name to store the extracted
//*          and modified SQL procedure source. Must be Recfm F or V,
//*          sequential or member, or a non-existing data set/member.
//*          The spcification can be a qualified name, a non-qualified
//*          name or a DD descriptor (in the form of DD:ddname).
//*****PH067S02 EXEC PGM=IEBGENER
//SYSUT1  DD *
//          DD:TEMPSRC
//SYSUT2  DD DSN=&&PARML,DISP=(NEW,PASS),SPACE=(TRK,1),
//          DCB=(LRECL=72,RECFM=FB,BLKSIZE=576)
//SYSPRINT DD DUMMY
//SYSIN  DD *
//          GENERATE MAXFLDS=1
//          RECORD FIELD=(72)
//*****Step 3: Populate a temporary PDS with REXX services used locally
//*****PH067S03 EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD DUMMY
//SYSUT2  DD DSN=&&REXXPDS,DISP=(NEW,PASS),
//          SPACE=(TRK,(5,5,2)),DCB=(LRECL=80,RECFM=FB,DSORG=PO)
//SYSIN  DD DSN=DSN!!0.SDSNMAC(SDSN8ERL1),
//          DISP=SHR
//          DD DATA,DLM='@@'
./ ADD NAME=DSNTEJ67
/* ***** REXX ***** DSNTEJ67 command **** */
address TSO
PREPSSID '. V9 NFM'
if rc>=8 then do;
  Say 'DSNTEJ67 Unable to establish a connection to DB2';
  exit 8;
end;

/* From DD:SPNAME read the stored procedure name to extract from DB2.
 * The name must be a schema qualified SP name (2-part name).
 * The SP must be for an external SQL procedure, with source in DB2.
 */
'EXECIO * DISKR SPNAME (OPEN FINIS STEM TEMP.)';
spname='';
do i = 1 to TEMP.0;
  spname = spname || TEMP.i;
end;
spname = "STRIP"(spname,'B');
/* Process the passed name to get the name parts,
 * plus the string and delimited forms.
 */
parse value "NAMPARTS"( spname ) with p# . namSpec
if p#<>2 then do;
  say 'DSNTEJ67 the passed SP name' spname,
      'was not schema qualified (2-parts)'
  exit 8;
end;
parse var namSpec a b c d e . ':' +1 sNam +(a) sSch +(b) . +(c),
      qNam +(d) qSch +(e)
spname = qSch'.qNam /* 2-part fully qualified form now */

/* From DD:SOURCEDS read the data set specification for where the
 * extracted and modified SQL proc source should be written at JOB end.
 */
'EXECIO * DISKR SOURCEDS (OPEN FINIS STEM TEMP.)';
sourceFile='';
do i = 1 to TEMP.0;
  sourceFile = sourceFile || TEMP.i;

```

```

end;
sourceFile = "STRIP"(sourceFile,'B');

/* Find the status of the target data set for the source. It must be a
 * Sequential data set, F or V record format (or capable of same).
 */
parse value "CHKANYFV"(sourceFile) with oRecfm oLrecl oEmpty;
if oRecfm=' ' then do;
  say 'DSNTEJ67 Cannot use the designated data set' sourceFile,
       'for SQL PL source'
  exit 8;
end;

prepConv:
/* From DD:HELPERSP1 deploy the native SQL SP DSN8.DSN8EN1 that will
 * provide native options for the external SQL proc to be migrated.
 * Use the subroutine form of the CRSQLPL service, asking for return
 * of a version activation statement, for processing after deployment.
 */
Say 'Setting up the native SQL PL helper routine...'
call "CRSQLPL" 'DD:HELPERSP1', , , 'REACTIVATE';
parse var result rc . 1 tok1 VerStmt ;
if "DATATYPE"(rc,'W')=0 then select; /* OK, 1st word not a number */
  when tok1='/*CP*/' then rc=0; /* Skip ACTIVATE for CREATE */
  when "WORDPOS"(tok1,'/*APR*/ /*APA*/')>0 /* ALTER REP, ALTER ADD */
    then rc = ActivateRtnVer( VerStmt );
  otherwise rc=8; /* Problem, force an error. */
end /* select */;
if RC>4 then do;
  say 'DSNTEJ67 cannot deploy the native SP used for migration.';
  exit 8;
end;

allocList=''; /* List of DDnames we allocate */

extractSQLproc:
/* Extract the SP source to a temporary SQLV file. Also get an S80
 * format edition to use with the precompiler for inspection purposes.
 */
'SQLPLSRC' spname 'DD:SQLVE' 'DD:S80E' 'ASIS';
if RC>4 then do;
  say 'DSNTEJ67 Cannot extract SQL procedure source';
  exit 8;
end;
allocList='SQLVE,S80E'; /* allocated FOR us */
/* Write extracted source ASIS now to the output data set.
 * We will rewrite it again later after reaching the point of editing.
 */
if oRecfm='F' then do;
  if oLrecl=80 then seq='SEQ'; else seq='';
  "SQLV2F" 'DD:SQLVE' sourceFile seq
end;
else "ANY2SQLV" 'DD:SQLVE' sourceFile 'EXTEND'
if RC<>0 then do;
  say 'DSNTEJ67 Cannot write extracted source to data set' sourceFile
  exit 8;
end;
else say 'Source for SP' spname 'written to data set' sourceFile

verifyExtSQL:
/* Verify the extracted source is valid external SQL procedure source
 * before going to far. Use the HOST(SQL) precompiler.
 */
/* Keep DD:LISTING active till then end. */
'ALLOCATE DDNAME(LISTING) NEW REUSE';
'CHKSQPL' 'DD:S80E' 'DD:LISTING' '.' 'MAR(1,80) HOST(SQL)'
if RC>4 then do;
  msg='DSNTEJ67 Extracted external SQL procedure source has errors';
  call endWithListing msg, allocList;
end;

chkoutSQLPL1:
/* Inspect the extracted external SQL procedure source without change
 * using the HOST(SQLPL) Checkout precompiler.
 * RC=8 errors are anticipated.
 */
allocList = allocList||',UT1';
'ALLOCATE DDNAME(UT1) NEW REUSE';
'CHKSQPL' 'DD:S80E' 'DD:LISTING' 'DD:UT1' 'MAR(1,80)'
if RC>8 then do;
  msg='DSNTEJ67 Fatal error running HOST(SQLPL) precompiler',
      'with external SQL procedure source'

```

```

call endWithListing msg, allocList;
end;
/* Getting no UT1 content typically represents a native SQL PL syntax
 * issue. In this context, that could be caused by some unforeseen
 * difference between valid external SQL PL and native SQL PL syntax.
 */
parse value "CHKANYFV"('DD:UT1') with utR . utE;
if utE<>'1' then do;
  msg='DSNTEJ67 Syntax error in external SQL proc source',
      'When viewed as native SQL PL';
  call endWithListing msg, allocList;
end;

editPrep:
/* Obtain an SQLPL TOC description, to use for editing the source. */
'SQLPLTOC' 'DD:S80E' 'DD:UT1' 'DD:TOC'
if RC<>0 then do;
  msg='DSNTEJ67 Unable to prepare for source editing (no TOC).'
  call endWithListing msg, allocList;
end;
allocList = allocList||',TOC';
/* Read TOC to get the OPTIONS element descriptor */
opts=''
"EXECIO * DISKR TOC (OPEN FINIS STEM TOC."
do i = 1 to TOC.0;
  parse var TOC.i elem desc;
  if elem='OPTS:' then do;
    opts = desc;
    leave;
  end;
end;
parse var opts o1 o2 o3 .
parse var o1 or1 ':' oc1; parse var o2 or2 ':' oc2;
/* Bring original external source into memory now, splitting into
 * three parts, Front (ahead of options), Back (after options)
 * and Middle (the options which will be replaced).
 */
"EXECIO 0 DISKR SQLVE (OPEN"
/* Front: all complete lines before options, into stem FR. */
FR.=''; FR.0=0;
if or1>1
  then "EXECIO" or1-1 "DISKR SQLVE (STEM FR.";
/* Middle: All records that have options on them
 * This will be at least one record (where options WOULD go).
 */
if o3='0' /* no options were present */
  then "EXECIO" 1 "DISKR SQLVE (STEM MD.)";
  else "EXECIO" 1+or2-or1 "DISKR SQLVE (STEM MD.)";
/* Back: all the remaining complete lines */
BK.=''; BK.0=0;
"EXECIO * DISKR SQLVE (FINIS STEM BK.)";
/* The Middle likely has portions of the Front, the Back, or both.
 * Separate those now, and then toss the middle. Process Back first.
 */
if FRm='' then FRm=''; /* collapse existing option indentation */
i=MD.0;
if o3='0'
  then j=oc1; /* When no options, the BK middle starts at oc1. */
  else j=oc2+1; /* With options, the BK middle starts after oc2. */
parse var MD.i MD.i =(j) BKm
if oc1<2
  then FRm='';
  else parse var MD.1 FRm =(oc1) MD.1

If MD.0 > 0 then do;
  say 'Removing these external SQL procedure options:'
  do i = 1 to MD.0; say MD.i; end;
end;

drop MD..TOC.
/* We now have the external source in stems FR., BK.
 * and strings FRm and BKm. Release the old options.
 * Free our allocated data sets now. Current LISTING remains...
 */
'FREE DDNAME('allocList')';
allocList='';

/* Get the replacement options from the helper routine DSN8.DSN8EN1
 * Use the FUNCTION invocation of the SQLCALL service to obtain
 * the value of the last parameter (the SP output parm).
 */
call "OUTTRAP" 'TEMP.';
```

```

nat_opt=SQLCALL("DSN8.DSN8EN1('sSch', 'sNam', VARCHAR('?', 5120))");
call "OUTTRAP" 'OFF';
if nat_opt=' ' | nat_opt='8' then do;
  msg="DSNTEJ67 Unable to obtain native options for replacement";
  call endWithListing msg ;
end;
say 'Inserting these native SQL PL options:';
temp=nat_opt;
do while temp<>'';
  parse var temp opn '25'x temp;
  say opn;
end;

/* Rewrite Original source using the new Native Options */
new_src='';
do i = 1 to FR.0; new_src=new_src || FR.i || '25'x; end; drop FR.
  new_src=new_src || FRm ; drop FRm
  new_src=new_src || nat_opt ; drop nat_opt
  new_src=new_src || BKm ; drop BKm
do i = 1 to BK.0; new_src=new_src || BK.i || '25'x; drop BK.i; end;
call "STR2SQLV" new_src, 'DD:SQLV'
if oRecfm='F' then do;
  if oLrecl=80 then seq='SEQ'; else seq='';
  "SQLV2F" 'DD:SQLV' sourceFile seq
end;
else "ANY2SQLV" 'DD:SQLV' sourceFile 'EXTEND'
"SQLV2F" 'DD:SQLV' 'DD:S80' 'S80'

chkoutSQLPL2:
/* Inspect the modified procedure source one last time
 * using the HOST(SQLPL) Checkout precompiler.
 */
'CHKSQLPL' 'DD:S80' 'DD:LISTING' '.' 'MAR(1,80)'
rrc=RC;
say 'Final HOST(SQLPL) Checkout Precompile ended with RC='rrc;
if rrc>0 then say 'Inspect the Listing for',
  'additional SQLPL source coding issues';
'EXECIO * DISKR LISTING (OPEN FINIS STEM LISTING.)';
call trimListing;
'EXECIO' listing.0 'DISKW LISTOUT (OPEN FINIS STEM LISTING.)';
'FREE DDNAME(LISTING,SQLV,S80)';
exit rRC;
/* ----- */

activateRtnVer: procedure
/* Process the Activate Version statement passed. Returns 0 or 4.    */
parse arg AVstmt ;
rcod=0;
"EXECIO * DISKR DB2SSID (OPEN FINIS STEM TEMP.)";
parse var TEMP.1 ssid .;
say 'Issuing...' AVstmt;
CALL SQLEDBS 'ATTACH TO' ssid;
call SQLEXEC "EXECUTE IMMEDIATE :AVSTMT";
if result<0 then do;
  say 'Trouble activating the native SQL PL routine version';
  say '==>`sqlca.sqlcode'<';
  say '==>`sqlca.sqlerrm'<';
  call SQLEXEC "ROLLBACK";
  rcod=4;
end;
else call SQLEXEC "COMMIT";
call SQLEDBS 'DETACH';
return rcod;

endWithListing:
'EXECIO * DISKR LISTING (OPEN FINIS STEM LISTING.)';
call trimListing;
'EXECIO' listing.0 'DISKW LISTOUT (OPEN FINIS STEM LISTING.)';
'FREE DDNAME(LISTING)';
if arg(2,'E')
  then 'FREE DDNAME(`arg(2)`); /* other DDnames to free */'
if arg(1,'E')
  then say arg(1); /* Message to end with */
exit 8;

/* trimListing: Reduce the occurrence of repeated headers, CC and page
 * numbers in the listing, so it appears like one
 * continuous stream.
 */
trimListing: procedure expose LISTING.
  hdrtypes = 'VERSION SYMBOL MESSAGES STATISTICS';
  j=0; k=LISTING.0; do i = 1 to k;

```

```

parse var LISTING.i 1 cc +1 line;
if cc='1' & "LEFT"(line,19)='DB2 SQL PRECOMPILER' then do; /*Hdr*/
  /* Reduce page header occurrences */
  key="WORD"(Line,4);
  loc="WORDPOS"(key,hdrtypes);
  if loc>0 then do;                                /* First hdr usage. */
    parse var line line 'PAGE' .                  /* Keep, w/o page num.*/
    hdrtypes = "DELWORD"(hdrtypes,loc,1);        /* Header now used. */
  end;
  else iterate;                                     /* Skip redundant hdr.*/
  end /*Hdr*/;
j=j+1; LISTING.j="STRIP"(line,'T');
end /* do i ... */;
do i=j+1 to k; drop LISTING.i; end;
LISTING.0=j;
return 1+k-j; /* lines trimmed */

/* ----- end DSNTEJ67 ----- */
@@
./ ENDUP
//***** Step 4: Run the sample process to extract external SQL SP source
//**      from DB2, convert the source to native SQL PL, save the
//**      modified source in a data set and finish with a source
//**      listing written to the job log.
//***** listing written to the job log.
//*****
//PH067S04 EXEC PGM=IKJEFT01,DYNAMNBR=30
//SYSEXEC DD DSN=&&REXXPDS,DISP=(OLD,PASS)
//SYSTSPRT DD SYSOUT=*
//DB2SSID DD DSN=&&PARM0,DISP=(OLD,PASS)
//SPNAME DD DSN=&&PARM1,DISP=(OLD,PASS)
//SOURCEDS DD DSN=&&PARM2,DISP=(OLD,PASS)
//LISTOUT DD SYSOUT=*
//HELPERSP1 DD DSN=DSN!0.SDSNIVPD(DSN8EN1),
//             DISP=SHR
//SYSTSIN DD *
%DSNTEJ67
/*

```

Referencia relacionada

["Programas de ejemplo para ayudar a preparar y ejecutar procedimientos SQL externos"](#) en la página 323
 Db2 proporciona trabajos de ejemplo para ayudarle a preparar y ejecutar procedimientos SQL externos. Todos los ejemplos están en el conjunto de datos de DSN1210.SDSENSAMP. Antes de poder ejecutar los ejemplos, debe personalizarlos para la instalación.

Creación de múltiples versiones de procedimientos externos

Para los procedimientos nativos de SQL, puede utilizar Db2 para crear y mantener varias versiones del procedimiento. Sin embargo, para los procedimientos externos, incluidos los procedimientos SQL externos, si necesita varias versiones de un procedimiento, deberá mantenerlas manualmente.

Antes de empezar

Función en desuso: Los procedimientos de SQL externo están en desuso y no están totalmente soportados como procedimientos SQL nativos. Para obtener mejores resultados, cree procedimientos de SQL nativo en su lugar. Para obtener más información, consulte ["Creación de procedimientos SQL nativos"](#) en la página 244 y ["Migración de un procedimiento de SQL externo a un procedimiento de SQL nativo"](#) en la página 307.

Procedimiento

Para crear varias versiones de procedimientos externos, incluidos procedimientos SQL externos, utilice una de las siguientes técnicas:

- Definir varios procedimientos con el mismo nombre en diferentes esquemas. Posteriormente, puede utilizar la ruta SQL para determinar qué versión del procedimiento debe utilizar un programa de llamada.

- Definir varias versiones del código ejecutable. Posteriormente, puede utilizar una versión concreta especificando el nombre del módulo de carga para la versión que desea utilizar en la cláusula EXTERNAL de la instrucción CREATE PROCEDURE o ALTER PROCEDURE.
- Definir varios paquetes para un procedimiento. Posteriormente, puede utilizar la opción COLLID, el registro especial CURRENT PACKAGESET o el registro especial CURRENT PACKAGE PATH para especificar qué versión del procedimiento debe utilizar la aplicación que realiza la llamada.
- Configure varios entornos WLM para utilizar diferentes versiones de un procedimiento.

Adición y modificación de datos en tablas de programas de aplicación

El programa de aplicación puede agregar, modificar o suprimir datos en cualquier tabla de Db2 para la que tenga el acceso apropiado.

Inserción de datos en tablas

Puede utilizar varios métodos para insertar datos en una tabla. Decida qué método utilizar basándose en la cantidad de datos que necesita insertar y otras operaciones que su programa tiene que llevar a cabo.

Acerca de esta tarea

Además de utilizar sentencias INSERT independientes, puede utilizar las siguientes formas de insertar datos en una tabla:

- Puede utilizar la instrucción MERGE para insertar nuevos datos y actualizar los existentes en la misma operación.
- Puede escribir un programa de aplicación para solicitar e introducir grandes cantidades de datos en una tabla.
- También puede utilizar la utilidad LOAD (cargar) de Db2 , para introducir datos de otras fuentes.

Tareas relacionadas

[Inserción de datos y actualización de datos en una sola operación](#)

Puede actualizar datos existentes e insertar nuevos datos en una única operación. Esta operación es útil cuando desee actualizar una tabla con un conjunto de filas, algunas de las cuales representan cambios de filas existentes y otras son nuevas filas.

Referencia relacionada

[LOAD \(Db2 Utilities\)](#)

Inserción de filas utilizando la sentencia INSERT

Una forma de insertar datos en tablas es utilizar la sentencia SQL INSERT. Este método es útil para insertar pequeñas cantidades de datos o insertar datos de otra tabla o vista.

Procedimiento

Emitir una declaración INSERT utilizando uno de los siguientes métodos:

- Especifique los valores de columna para insertar una sola fila. Puede especificar constantes, variables de host, expresiones, DEFAULT o NULL mediante la cláusula VALUES.
- En un programa de aplicación, especifique matrices de valores de columna para insertar varias filas en una tabla. Utilice matrices de variables de host en la cláusula VALUES de la instrucción *INSERT FOR n ROWS* para añadir varias filas de valores de columna a una tabla.
- Incluya una sentencia SELECT en la sentencia INSERT para indicar a Db2 que otra tabla o vista contiene los datos para la nueva fila o filas.

En cada caso, por cada fila que inserte, debe proporcionar un valor para cualquier columna que no tenga un valor predeterminado. Para una columna que cumpla una de las siguientes condiciones, especifique DEFAULT para indicar a Db2 que inserte el valor predeterminado para esa columna:

- La columna es anulable.
- La columna se define con un valor predeterminado.
- La columna tiene el tipo de datos ROWID. Las columnas ROWID siempre tienen valores predeterminados.
- La columna es una columna de identidad. Las columnas de identidad siempre tienen valores predeterminados.
- La columna es una columna de marca de tiempo de cambio de fila.

Los valores que puede insertar en una columna ROWID, una columna de identidad o una columna de cambio de marca de tiempo de fila dependen de si la columna está definida con GENERATED ALWAYS o GENERATED BY DEFAULT.

Puede utilizar la cláusula VALUES de la instrucción INSERT para insertar una sola fila de valores de columna en una tabla. Puede nombrar todas las columnas para las que está proporcionando valores u omitir la lista de nombres de columnas. Si omite la lista de nombres de columna, debe especificar valores para **todas las columnas**.

Recomendación: Para las sentencias INSERT estáticas, nombre todas las columnas para las que está proporcionando valores por las siguientes razones:

- Su declaración INSERT es independiente del formato de la tabla. (Por ejemplo, no es necesario cambiar el extracto cuando se añade una columna a la tabla)
- Puede verificar que está especificando los valores en orden.
- Sus declaraciones de origen son más autoexplicativas.

Si no nombra las columnas en una instrucción INSERT estática y se añade una columna a la tabla, puede producirse un error si se vuelve a ejecutar la instrucción INSERT. Se producirá un error después de cualquier reasignación de la instrucción INSERT a menos que cambie la instrucción INSERT para incluir un valor para la nueva columna. Esto es así incluso si la nueva columna tiene un valor predeterminado.

Cuando enumere los nombres de las columnas, debe especificar sus valores correspondientes en el mismo orden que en la lista de nombres de columnas.

Ejemplo INSERT statements

- La siguiente sentencia inserta información sobre un nuevo departamento en la tabla YDEPT.

```
INSERT INTO YDEPT (DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION)
VALUES ('E31', 'DOCUMENTATION', '000010', 'E01', '');
```

Después de insertar una nueva fila de departamento en su tabla YDEPT, puede utilizar una instrucción SELECT para ver lo que ha cargado en la tabla. La siguiente instrucción SQL muestra todas las nuevas filas de departamento que ha insertado:

```
SELECT *
FROM YDEPT
WHERE DEPTNO LIKE 'E%'
ORDER BY DEPTNO;
```

La tabla de resultados se parece a la siguiente salida:

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
E01	SUPPORT SERVICES	000050	A00	-----
E11	OPERATIONS	000090	E01	-----
E21	SOFTWARE SUPPORT	000100	E01	-----
E31	DOCUMENTATION	000010	E01	-----

- La siguiente declaración inserta información sobre un nuevo empleado en la tabla YEMP. Debido a que la columna WORKDEPT es una clave externa, el valor que se inserta para esa columna (E31) debe ser un valor en la columna de clave primaria, que es DEPTNO en la tabla YDEPT.

```
INSERT INTO YEMP
VALUES ('000400', 'RUTHERFORD', 'B', 'HAYES', 'E31', '5678', '1998-01-01',
       'MANAGER', 16, 'M', '1970-07-10', 24000, 500, 1900);
```

- La siguiente declaración también inserta una fila en la tabla YEMP. Debido a que las columnas no especificadas permiten valores nulos, Db2 inserta valores nulos en las columnas que no especifique.

```
INSERT INTO YEMP
(EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, JOB)
VALUES ('000410', 'MILLARD', 'K', 'FILLMORE', 'D11', '4888', 'MANAGER');
```

Conceptos relacionados

[Reglas para insertar datos en una columna de identidad](#)

Una columna de *identidad* contiene un valor numérico único para cada fila de la tabla. Si puede insertar datos en una columna de identidad y cómo se insertan esos datos depende de cómo se define la columna.

[Reglas para insertar datos en una columna ROWID](#)

Una columna *ROWID* contiene valores únicos que identifican cada fila de una tabla. Si puede insertar datos en una columna ROWID y cómo se insertan esos datos depende de cómo se define la columna.

Tareas relacionadas

[Inserción de varias filas de datos desde matrices de variables de host](#)

Utilice las variables de host en la sentencia INSERT cuando no conozca al menos algunos de los valores que se han de insertar hasta que el programa se ejecuta.

[Inserción de filas de una tabla en otra tabla](#)

Puede copiar una o más filas de una tabla a otra tabla.

[Cargar datos mediante la emisión de sentencias INSERT \(Guía de administración de Db2 \)](#)

Referencia relacionada

[INSERT declaración \(Db2 SQL\)](#)

[CREATE TABLE declaración \(Db2 SQL\)](#)

Inserción de filas de una tabla en otra tabla

Puede copiar una o más filas de una tabla a otra tabla.

Procedimiento

Utilice un fullselect dentro de una instrucción INSERT.

ejemplos

Ejemplo

La siguiente instrucción SQL crea una tabla llamada TELE:

```
CREATE TABLE TELE
(NAME2  VARCHAR(15)  NOT NULL,
 NAME1  VARCHAR(12)  NOT NULL,
 PHONE  CHAR(4));
```

La siguiente declaración copia datos de DSN8C10.EMP en la tabla recién creada:

```
INSERT INTO TELE
SELECT LASTNAME, FIRSTNME, PHONENO
FROM DSN8C10.EMP
WHERE WORKDEPT = 'D21';
```

Las dos sentencias anteriores crean y rellenan una tabla, TELE, que se parece a la siguiente tabla:

NAME2	NAME1	PHONE
PULASKI	EVA	7831
JEFFERSON	JAMES	2094
MARINO	SALVATORE	3780
SMITH	DANIEL	0961
JOHNSON	SYBIL	8953
PEREZ	MARIA	9001
MONTEVERDE	ROBERT	3780

El ejemplo de la instrucción CREATE TABLE crea una tabla que, al principio, está vacía. La tabla tiene columnas para apellidos, nombres y números de teléfono, pero no tiene filas.

La instrucción INSERT llena la tabla recién creada con datos seleccionados de la base de datos de la aplicación (DSN8C10).EMP tabla: los nombres y números de teléfono de los empleados del departamento de D21.

Ejemplo

La siguiente instrucción CREATE crea una tabla que contiene el nombre del departamento y el número de teléfono de un empleado. La sentencia fullselect dentro de la sentencia INSERT llena la tabla DLIST con datos de filas que se seleccionan de dos tablas existentes, DSN8C10.DEPT y DSN8C10.EMP.

```

CREATE TABLE DLIST
  (DEPT    CHAR(3)      NOT NULL,
   DNAME   VARCHAR(36),
   LNAME   VARCHAR(15)   NOT NULL,
   FNAME   VARCHAR(12)   NOT NULL,
   INIT    CHAR          ,
   PHONE   CHAR(4)  );

INSERT INTO DLIST
  SELECT DEPTNO, DEPTNAME, LASTNAME, FIRSTNAME, MIDINIT, PHONENO
    FROM DSN8C10.DEPT, DSN8C10.EMP
   WHERE DEPTNO = WORKDEPT;

```

Reglas para insertar datos en una columna ROWID

Una columna ROWID contiene valores únicos que identifican cada fila de una tabla. Si puede insertar datos en una columna ROWID y cómo se insertan esos datos depende de cómo se define la columna.

Una columna ROWID es una columna que se define con un tipo de datos ROWID. Debe tener una columna con un tipo de datos ROWID en una tabla que contenga una columna LOB. La columna ROWID se almacena en la tabla base y se utiliza para buscar los datos LOB reales en el espacio de tabla LOB. Además, una columna ROWID le permite escribir consultas que navegan directamente a una fila en una tabla. Para obtener información sobre el uso de columnas ROWID para el acceso directo a filas, consulte “Especificación del acceso a filas directo utilizando los ID de fila” en la página 455.

Antes de insertar datos en una columna ROWID, debe saber cómo está definida la columna ROWID. Las columnas ROWID pueden definirse como GENERATED ALWAYS (generadas siempre) o GENERATED BY DEFAULT (generadas por defecto). GENERADO SIEMPRE significa que Db2 genera un valor para la columna y no se pueden insertar datos en esa columna. Si la columna se define como GENERATED BY DEFAULT, puede insertar un valor, y Db2 proporciona un valor predeterminado si no proporciona uno.

Ejemplo : Supongamos que las tablas T1 y T2 tienen dos columnas: una columna de enteros y una columna ROWID. Para que la siguiente declaración se ejecute correctamente, ROWIDCOL2 debe estar definido como GENERATED BY DEFAULT.

```

INSERT INTO T2 (INTCOL2,ROWIDCOL2)
  SELECT * FROM T1;

```

Si ROWIDCOL2 se define como GENERATED ALWAYS, no puede insertar los datos de la columna ROWID de T1 en T2, pero puede insertar los datos de la columna entera. Para insertar solo los datos enteros, utilice uno de los siguientes métodos:

- Especifique solo la columna entera en su instrucción INSERT, como en la siguiente instrucción:

```
INSERT INTO T2 (INTCOL2)
SELECT INTCOL1 FROM T1;
```

- Especifique la cláusula OVERRIDING USER VALUE en su sentencia INSERT para indicar a Db2 que ignore cualquier valor que proporcione para las columnas generadas por el sistema, como en la siguiente sentencia:

```
INSERT INTO T2 (INTCOL2,ROWIDCOL2) OVERRIDING USER VALUE
SELECT * FROM T1;
```

Conceptos relacionados

[Acceso directo de fila \(PRIMARY_ACCESTYPE='D'\) \(Db2 Performance\)](#)

[Tipo de datos ROWID \(Introducción a Db2 para z/OS\)](#)

Tareas relacionadas

[Especificación del acceso a filas directo utilizando los ID de fila](#)

En algunas aplicaciones, es posible utilizar el valor de una columna ROWID para ir directamente a una fila.

Reglas para insertar datos en una columna de identidad

Una columna de identidad contiene un valor numérico único para cada fila de la tabla. Si puede insertar datos en una columna de identidad y cómo se insertan esos datos depende de cómo se define la columna.

Una columna de identidad es una columna numérica, definida en una instrucción CREATE TABLE o ALTER TABLE, que tiene valores ascendentes o descendentes. Para que una columna de identidad sea lo más útil posible, sus valores también deben ser únicos. La columna tiene un tipo de datos SMALLINT, INTEGER o DECIMAL(*p,0*) y se define con la cláusula AS IDENTITY. La cláusula AS IDENTITY especifica que la columna es una columna de identidad. Para obtener información sobre el uso de columnas de identidad para identificar filas de forma exclusiva, consulte [“Columnas de identidad” en la página 133](#)

Antes de insertar datos en una columna de identidad, debe saber cómo está definida la columna. Las columnas de identidad se definen con la cláusula GENERATED ALWAYS o GENERATED BY DEFAULT. GENERADO SIEMPRE significa que Db2 genera un valor para la columna y no puede insertar datos en esa columna. Si la columna se define como GENERATED BY DEFAULT, puede insertar un valor, y Db2 proporciona un valor predeterminado si no proporciona uno.

Ejemplo: insertar datos en columnas de identidad

Supongamos que las tablas T1 y T2 tienen dos columnas: una columna de caracteres y una columna de enteros que se define como una columna de identidad. Para que la siguiente declaración se ejecute correctamente, IDENTCOL2 debe estar definido como GENERATED BY DEFAULT.

```
INSERT INTO T2 (CHARCOL2,IDENTCOL2)
SELECT * FROM T1;
```

Si IDENTCOL2 se define como GENERATED ALWAYS, no puede insertar los datos de la columna de identidad de T1 en T2, pero puede insertar los datos de la columna de caracteres. Para insertar solo los datos de caracteres, utilice uno de los siguientes métodos:

- Especifique solo la columna de caracteres en su instrucción INSERT, como en la siguiente instrucción:

```
INSERT INTO T2 (CHARCOL2)
SELECT CHARCOL1 FROM T1;
```

- Especifique la cláusula OVERRIDING USER VALUE en su sentencia INSERT para indicar a Db2 que ignore cualquier valor que proporcione para las columnas generadas por el sistema, como en la siguiente sentencia:

```
INSERT INTO T2 (CHARCOL2,IDENTCOL2) OVERRIDING USER VALUE
SELECT * FROM T1;
```

Conceptos relacionados

[Columnas de identidad](#)

Una columna de identidad contiene un valor numérico exclusivo para cada fila de la tabla. Db2 puede generar automáticamente valores numéricos secuenciales para esta columna cuando las filas se insertan en la tabla. Así, las columnas de identidad son ideales para los valores de clave primaria, como los números de empleado o los números de producto.

Tareas relacionadas

[Suministro de clave única para una tabla](#)

Si una tabla no tiene valores de columna únicos, puede proporcionar un identificador único utilizando columnas ROWID o columnas de identidad para almacenar valores únicos para cada fila de una tabla.

Restricciones al asignar valores a columnas con tipos distintos

Se requieren ciertas condiciones cuando se asigna un valor de columna a otra columna o cuando se asigna una constante a una columna de un tipo distinto. Si no se cumplen las condiciones, no podrá asignar el valor.

Al asignar un valor de columna a otra columna o una constante a una columna de un tipo distinto, el tipo del valor que se va a asignar debe coincidir con el tipo de columna, o debe poder convertir un tipo en otro. De lo contrario, no podrá asignar el valor.

Si necesita asignar un valor de un tipo distinto a una columna de otro tipo distinto, debe existir una función que convierta el valor de un tipo a otro. Db2 , al proporcionar funciones de conversión solo entre tipos distintos y sus tipos de origen, debe escribir la función para convertir de un tipo distinto a otro.

Asignar valores de columna a columnas con distintos tipos

Supongamos que las tablas JAPAN_SALES y JAPAN_SALES_03 se definen así:

```
CREATE TABLE JAPAN_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1990),
   TOTAL         JAPANESE_YEN);

CREATE TABLE JAPAN_SALES_03
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR);
```

Debe insertar los valores de la columna TOTAL en JAPAN_SALES en la columna TOTAL de JAPAN_SALES_03. Debido a que las sentencias INSERT siguen reglas de asignación, Db2 no le permite insertar los valores directamente de una columna a otra porque las columnas son de tipos distintos. Supongamos que se ha escrito una función definida por el usuario llamada US_DOLLAR que acepta valores de tipo JAPANESE_YEN como entrada y devuelve valores de tipo US_DOLLAR. A continuación, puede utilizar esta función para insertar valores en la tabla " JAPAN_SALES_03 ":

```
INSERT INTO JAPAN_SALES_03
  SELECT PRODUCT_ITEM, US_DOLLAR(TOTAL)
    FROM JAPAN_SALES
   WHERE YEAR = 2003;
```

Asignación de valores de columna con tipos distintos a variables de host

Las reglas para asignar tipos distintos a variables de host o variables de host a columnas de tipos distintos difieren de las reglas para constantes y columnas.

Puede asignar un valor de columna de un tipo distinto a una variable anfitriona si puede asignar un valor de columna del tipo de origen del tipo distinto a la variable anfitriona. En el siguiente ejemplo, puede asignar SIZECOL1 y SIZECOL2, que tiene un tipo SIZE distinto, a variables host de tipo double y short porque el tipo de origen de SIZE, que es INTEGER, puede asignarse a variables host de tipo double o short.

```
EXEC SQL BEGIN DECLARE SECTION;
  double hv1;
  short hv2;
EXEC SQL END DECLARE SECTION;
```

```

CREATE DISTINCT TYPE SIZE AS INTEGER;
CREATE TABLE TABLE1 (SIZECOL1 SIZE, SIZECOL2 SIZE);
:
SELECT SIZECOL1, SIZECOL2
  INTO :hv1, :hv2
   FROM TABLE1;

```

Asignación de valores de variables de host a columnas con tipos distintos

Cuando asignas un valor en una variable host a una columna con un tipo distinto, el tipo de la variable host debe poder convertirse al tipo distinto.

En este ejemplo, los valores de la variable de host hv2 se pueden asignar a las columnas SIZECOL1 y SIZECOL2, porque el tipo de datos C short es equivalente al tipo de datos C Db2 , SMALLINT, y SMALLINT es promocionable al tipo de datos INTEGER. Sin embargo, los valores de hv1 no se pueden asignar a SIZECOL1 y SIZECOL2, porque el tipo de datos C doble, que es equivalente al tipo de datos DOUBLE de Db2 , no se puede promover al tipo de datos INTEGER.

```

EXEC SQL BEGIN DECLARE SECTION;
  double hv1;
  short  hv2;
EXEC SQL END DECLARE SECTION;
CREATE DISTINCT TYPE SIZE AS INTEGER;
CREATE TABLE TABLE1 (SIZECOL1 SIZE, SIZECOL2 SIZE);
:
INSERT INTO TABLE1
  VALUES (:hv1,:hv1);      /* Invalid statement */
INSERT INTO TABLE1
  VALUES (:hv2,:hv2);      /* Valid statement */

```

Conceptos relacionados

[Promoción de tipos de datos \(Db2 SQL\)](#)

Inserción de datos y actualización de datos en una sola operación

Puede actualizar datos existentes e insertar nuevos datos en una única operación. Esta operación es útil cuando desee actualizar una tabla con un conjunto de filas, algunas de las cuales representan cambios de filas existentes y otras son nuevas filas.

Acerca de esta tarea

Puede actualizar datos existentes e insertar datos nuevos en una sola operación utilizando la instrucción MERGE.

Por ejemplo, una aplicación puede solicitar un conjunto de filas de una base de datos, permitir que un usuario modifique los datos a través de una interfaz gráfica de usuario y, a continuación, almacenar los datos modificados en la base de datos. Algunos de estos datos modificados son actualizaciones de filas existentes, y algunos de estos datos son filas nuevas. Puede realizar estas operaciones de actualización e inserción en un solo paso.

Procedimiento

Emitir una instrucción MERGE.

Para actualizar datos existentes e insertar datos nuevos, especifique una sentencia MERGE con las cláusulas WHEN MATCHED y WHEN NOT MATCHED. Estas cláusulas especifican cómo maneja Tiffany & Co. (Db2) los datos coincidentes y no coincidentes. Si Db2 encuentra una fila coincidente, esa fila se actualiza. Si Db2 no encuentra una fila coincidente, se inserta una nueva fila.

Ejemplo

Supongamos que necesita actualizar el inventario en un concesionario de automóviles. Debe añadir nuevos modelos de automóviles al inventario y actualizar la información sobre los modelos de automóviles que ya están en el inventario.

Podría realizar estos cambios con la siguiente serie de instrucciones:

```
UPDATE INVENTORY
  SET QUANTITY = QUANTITY + :hv_delta
 WHERE MODEL = :hv_model;

--begin pseudo code
if sqlcode >= 0
then do
  GD
    if rc = 0  then INSERT..
  end
-- end pseudo code

GET DIAGNOSTICS :rc = ROW_COUNT;

IF rc = 0 THEN
  INSERT INTO INVENTORY VALUES (:hv_model, :hv_delta);
END IF;
```

La instrucción MERGE simplifica la actualización y la inserción en una sola instrucción:

```
MERGE INTO INVENTORY
USING ( VALUES (:hv_model, :hv_delta) ) AS SOURCE(MODEL, DELTA)
ON INVENTORY.MODEL = SOURCE.MODEL
  WHEN MATCHED THEN UPDATE SET QUANTITY = QUANTITY + SOURCE.DELTA
  WHEN NOT MATCHED THEN INSERT VALUES (SOURCE.MODEL, SOURCE.DELTA)
NOT ATOMIC CONTINUE ON SQLEXCEPTION;
```

Referencia relacionada

[MERGE declaración \(Db2 SQL\)](#)

Selección de valores al fusionar datos

Cuando actualiza datos existentes e inserta datos nuevos en una sola operación de combinación, puede seleccionar valores de esas filas al mismo tiempo.

Procedimiento

Especificando la instrucción MERGE en la cláusula FROM de la instrucción SELECT.

Cuando fusionas una o más filas en una tabla, puedes recuperar:

- El valor de una columna generada automáticamente, como una ROWID o una columna de identidad
- Cualquier valor predeterminado para las columnas
- Todos los valores de una fila combinada, sin especificar nombres de columnas individuales
- Valores calculados basados en los cambios en las filas combinadas

Especifique la cláusula FINAL TABLE con sentencias SELECT FROM MERGE. La TABLA FINAL consiste en las filas de la tabla o vista después de que se produce la fusión.

Ejemplo

Supongamos que necesita introducir datos en la tabla STOCK, que contiene los símbolos de las acciones de la empresa y los precios de las acciones de su cartera de valores. Algunos de sus datos de entrada se refieren a empresas que ya están en la tabla STOCK; algunos de los datos se refieren a empresas que está añadiendo a su cartera de acciones. Si el símbolo bursátil existe en la columna SYMBOL de la tabla STOCK, debe actualizar la columna PRICE. Si el símbolo de las acciones de la empresa aún no está en la tabla ACCIONES, debe insertar una nueva fila con el símbolo de las acciones y el precio de las acciones. Además, debe agregar un nuevo valor DELTA a su salida para mostrar el cambio en el precio de las acciones.

Supongamos que la tabla STOCK contiene los datos que se muestran en [Tabla 59 en la página 360](#).

Tabla 59. Tabla STOCK antes de la sentencia SELECT FROM MERGE

SÍMBOLO	PRECIO
XCOM	95.00
YCOM	24.50

Ahora, supongamos que :hv_symbol y :hv_price son matrices de variables de host que contienen datos actualizados que se corresponden con los datos que se muestran en [Tabla 59 en la página 360](#). [Tabla 60 en la página 360](#) muestra los datos de la variable de host para la actividad de existencias.

Tabla 60. Matrices de variables de host de actividad de stock

hv_symbol	hv_price
XCOM	97.00
NUEVO	30.00
XCOM	107.00

NEWC es nuevo en la tabla STOCK, por lo que su símbolo y precio deben insertarse en la tabla STOCK. Las filas de XCOM en [Tabla 60 en la página 360](#) representan los precios de las acciones modificados, por lo que estos valores deben actualizarse en la tabla STOCK. Además, el resultado debe mostrar la variación de los precios de las acciones como un valor DELTA.

La siguiente instrucción SELECT FROM MERGE actualiza el precio de XCOM, inserta el símbolo y el precio de NEWC y devuelve un resultado que incluye un valor DELTA para el cambio en el precio de las acciones.

```

SELECT SYMBOL, PRICE, DELTA FROM FINAL TABLE
(MERGE INTO STOCK AS S INCLUDE (DELTA DECIMAL(5,20)
USING (:hv_symbol, :hv_price) FOR :hv_nrows ROWS) AS R (SYMBOL, PRICE)
ON S.SYMBOL = R.SYMBOL
WHEN MATCHED THEN UPDATE SET
    DELTA = R.PRICE - S.PRICE, PRICE=R.PRICE
WHEN NOT MATCHED THEN INSERT
    (SYMBOL, PRICE, DELTA) VALUES (R.SYMBOL, R.PRICE, R.PRICE)
NOT ATOMIC CONTINUE ON SQLEXCEPTION);

```

La cláusula INCLUDE especifica que se puede devolver una columna adicional, DELTA, en el resultado sin añadir una columna a la tabla STOCK. La parte UPDATE de la instrucción MERGE establece el valor DELTA en el diferencial del precio de las acciones anterior con el valor establecido para la operación de actualización. La parte INSERT de la instrucción MERGE establece el valor DELTA en el mismo valor que la columna PRICE.

Después de procesar la instrucción SELECT FROM MERGE, la tabla STOCK contiene los datos que se muestran en [Tabla 61 en la página 360](#).

Tabla 61. Tabla STOCK después de la sentencia SELECT FROM MERGE

SÍMBOLO	PRECIO
XCOM	107.00
YCOM	24.50
NUEVO	30.00

La siguiente salida de la sentencia SELECT FROM MERGE incluye tanto actualizaciones de XCOM como un valor DELTA para cada fila de salida.

```

SYMBOL      PRICE      DELTA
=====
XCOM        97.00     2.00
NEWC        30.00     30.00
XCOM        107.00    10.00

```

Selección de valores al insertar datos

Cuando inserta filas en una tabla, también puede seleccionar valores de las filas insertadas al mismo tiempo.

Acerca de esta tarea

Cuando inserta una o más filas nuevas en una tabla, también puede recuperar filas, incluidos los siguientes valores:

- El valor de una columna generada automáticamente, como una ROWID o una columna de identidad
- Cualquier valor predeterminado para las columnas
- Todos los valores de una fila insertada, sin especificar nombres de columnas individuales
- Todos los valores que se insertan mediante una operación INSERT de varias filas
- Valores que se modifican por un desencadenante BEFORE INSERT

Procedimiento

Especifique la instrucción INSERT en la cláusula FROM de la instrucción SELECT.

Las filas que se insertan en la tabla de destino producen una tabla de resultados cuyas columnas se pueden referenciar en la lista SELECT de la consulta. Las columnas de la tabla de resultados se ven afectadas por las columnas, restricciones y desencadenantes que se definen para la tabla de destino:

- La tabla de resultados incluye valores generados por el motor de base de datos (Db2) para las columnas de identidad, las columnas ROWID o las columnas de marca de tiempo de cambio de fila.
- Antes de generar la tabla de resultados, Db2 aplica cualquier restricción que afecte a la operación de inserción (es decir, restricciones de verificación, restricciones de índice único y restricciones de integridad referencial).
- La tabla de resultados incluye cualquier cambio que resulte de un desencadenante BEFORE que se activa mediante la operación de inserción. Un desencadenante AFTER no afecta a los valores de la tabla de resultados.

Ejemplos

Además de los ejemplos que utilizan las tablas de muestra de la base de datos de muestras de productos (Db2), los ejemplos de este tema utilizan una tabla EMPSAMP que tiene la siguiente definición:

```
CREATE TABLE EMPSAMP
  (EMPNO    INTEGER GENERATED ALWAYS AS IDENTITY,
   NAME     CHAR(30),
   SALARY   DECIMAL(10,2),
   DEPTNO   SMALLINT,
   LEVEL    CHAR(30),
   HIREDATE VARCHAR(30) NOT NULL WITH DEFAULT 'New Hire',
   HIREDATE DATE NOT NULL WITH DEFAULT );
```

Ejemplo 1: Recuperación de valores de columna generados

Supongamos que necesita insertar una fila para un nuevo empleado en la tabla EMPSAMP. Para averiguar los valores de las columnas EMPNO, HIREDATE y HIREDATE generadas, utilice la siguiente instrucción SELECT FROM INSERT:

```
SELECT EMPNO, HIREDATE, HIREDATE
  FROM FINAL TABLE (INSERT INTO EMPSAMP (NAME, SALARY, DEPTNO, LEVEL)
                    VALUES('Mary Smith', 35000.00, 11, 'Associate'));
```

La sentencia SELECT devuelve el valor de identidad generado por el sistema (Db2) para la columna EMPNO, el valor predeterminado «New Hire» para la columna HIREDATE y el valor del registro especial CURRENT DATE para la columna HIREDATE.

Recomendación: Utilice la instrucción SELECT FROM INSERT para insertar una fila en una tabla principal y recuperar el valor de una clave principal que fue generada por Db2 (una columna ROWID

o de identidad). En otra sentencia INSERT, especifique este valor generado como valor para una clave externa en una tabla dependiente.

Ejemplo 2: Recuperación de valores actualizados por activadores

Supongamos que se crea un disparador BEFORE INSERT en la tabla EMPSAMP para dar a todos los nuevos empleados de nivel asociado un aumento de salario de 5000 \$. El activador tiene la siguiente definición:

```
CREATE TRIGGER NEW_ASSOC
NO CASCADE BEFORE INSERT ON EMPSAMP
REFERENCING NEW AS NEWSALARY
FOR EACH ROW MODE DB2SQL
WHEN (NEWSALARY.LEVEL = 'ASSOCIATE')
BEGIN ATOMIC
    SET NEWSALARY.SALARY = NEWSALARY.SALARY + 5000.00;
END;
```

La sentencia INSERT en la cláusula FROM de la siguiente sentencia SELECT inserta un nuevo empleado en la tabla EMPSAMP:

```
SELECT NAME, SALARY
FROM FINAL TABLE (INSERT INTO EMPSAMP (NAME, SALARY, LEVEL)
VALUES('Mary Smith', 35000.00, 'Associate'));
```

La sentencia SELECT devuelve un salario de 40000.00 para Mary Smith en lugar del salario inicial de 35000.00 que se especificó explícitamente en la sentencia INSERT.

Seleccionar valores al insertar una sola fila:

Cuando inserta una nueva fila en una tabla, puede recuperar cualquier columna en la tabla de resultados de la instrucción SELECT FROM INSERT. Cuando incrusta esta declaración en una aplicación, recupera la fila en variables de host utilizando el SELECT... EN forma de extracto.

Ejemplo 4: Recuperar todos los valores de una fila insertada en una estructura.

Puede recuperar todos los valores de una fila que se inserta en una estructura. Por ejemplo, en la siguiente declaración: empstruct es una estructura de variables de host que se declara con variables para cada una de las columnas de la tabla EMPSAMP.

```
EXEC SQL SELECT * INTO :empstruct
FROM FINAL TABLE (INSERT INTO EMPSAMP (NAME, SALARY, DEPTNO, LEVEL)
VALUES('Mary Smith', 35000.00, 11, 'Associate'));
```

Ejemplo 4: Selección de valores al insertar datos en una vista

Si la sentencia INSERT hace referencia a una vista que está definida con una condición de búsqueda, esa vista debe estar definida con la opción WITH CASCDED CHECK OPTION. Cuando inserta datos en la vista, la tabla de resultados de la instrucción SELECT FROM INSERT incluye solo las filas que satisfacen la definición de la vista.

Debido a que view V1 se define con la opción WITH CASCDED CHECK OPTION, puede hacer referencia a V1 en la instrucción INSERT:

```
CREATE VIEW V1 AS
SELECT C1, I1 FROM T1 WHERE I1 > 10
WITH CASCDED CHECK OPTION;

SELECT C1 FROM
FINAL TABLE (INSERT INTO V1 (I1) VALUES(12));
```

El valor 12 satisface la condición de búsqueda de la definición de vista, y la tabla de resultados consta del valor de C1 en la fila insertada.

Si utiliza un valor que no cumple la condición de búsqueda de la definición de vista, la operación de inserción falla y Db2 devuelve un error.

Ejemplo 5: Selección de valores ROWID al insertar varias filas

En un programa de aplicación, para recuperar valores de la inserción de varias filas, declare un cursor para que la instrucción INSERT esté en la cláusula FROM de la instrucción SELECT del cursor.

Para ver los valores de las columnas ROWID que se insertan en la tabla de fotos y currículums de los empleados, puede declarar el siguiente cursor:

```
EXEC SQL DECLARE CS1 CURSOR FOR
  SELECT EMP_ROWID
    FROM FINAL TABLE (INSERT INTO DSN8C10.EMP_PHOTO_RESUME (EMPNO)
                      SELECT EMPNO FROM DSN8C10.EMP);
```

Ejemplo 6: Uso de la cláusula **FETCH FIRST**

Para ver solo las primeras cinco filas que se insertan en la tabla de fotos y currículums del empleado, utilice la cláusula **FETCH FIRST**:

```
EXEC SQL DECLARE CS2 CURSOR FOR
  SELECT EMP_ROWID
    FROM FINAL TABLE (INSERT INTO DSN8C10.EMP_PHOTO_RESUME (EMPNO)
                      SELECT EMPNO FROM DSN8C10.EMP)
                      FETCH FIRST 5 ROWS ONLY;
```

Ejemplo 7: Uso de la cláusula **INPUT SEQUENCE**

Para recuperar filas en el orden en que se insertan, utilice la cláusula **INPUT SEQUENCE**:

```
EXEC SQL DECLARE CS3 CURSOR FOR
  SELECT EMP_ROWID
    FROM FINAL TABLE (INSERT INTO DSN8C10.EMP_PHOTO_RESUME (EMPNO)
                      VALUES(:hva_empno)
                      FOR 5 ROWS)
                      ORDER BY INPUT SEQUENCE;
```

La cláusula **INPUT SEQUENCE** solo puede especificarse si hay una instrucción **INSERT** en la cláusula **FROM** de la instrucción **SELECT**. En este ejemplo, las filas se insertan a partir de una matriz de números de empleados.

Ejemplo 8: Insertar filas con múltiples CCSID de codificación

Supongamos que desea llenar una tabla ASCII con valores de una tabla EBCDIC y, a continuación, ver los valores seleccionados de la tabla ASCII. Puede utilizar el siguiente cursor para seleccionar las columnas EBCDIC, llenar la tabla ASCII y, a continuación, recuperar los valores ASCII:

```
EXEC SQL DECLARE CS4 CURSOR FOR
  SELECT C1, C2
    FROM FINAL TABLE (INSERT INTO ASCII_TABLE
                      SELECT * FROM EBCDIC_TABLE);
```

Ejemplo 9: Selección de columnas adicionales al insertar datos

Puede utilizar la cláusula **INCLUDE** para introducir una nueva columna en la tabla de resultados, pero no para añadir una columna a la tabla de destino.

Supongamos que necesita insertar datos de números de departamento en la tabla de proyectos. Supongamos también que desea recuperar el número de departamento y el número de gerente correspondiente para cada departamento. Dado que MGRNO no es una columna de la tabla del proyecto, puede utilizar la cláusula **INCLUDE** para incluir el número de gerente en el resultado, pero no en la operación de inserción. La siguiente instrucción **SELECT FROM INSERT** realiza la operación de inserción y recupera los datos.

```
DECLARE CS1 CURSOR FOR
  SELECT manager_num, projname FROM FINAL TABLE
  (INSERT INTO PROJ (DEPTNO) INCLUDE(manager_num CHAR(6))
   SELECT DEPTNO, MGRNO FROM DEPT);
```

Ejemplo 10: Tabla de resultados del cursor cuando se insertan varias filas

En un programa de aplicación, cuando se insertan varias filas en una tabla, se declara un cursor para que la instrucción **INSERT** esté en la cláusula **FROM** de la instrucción **SELECT** del cursor. La tabla de resultados del cursor se determina durante el procesamiento del cursor ABIERTO. La tabla de resultados puede verse afectada o no por otros procesos de su aplicación.

Cuando se declara un cursor desplazable, el cursor debe declararse con la palabra clave **INSENSITIVE** si hay una instrucción **INSERT** en la cláusula **FROM** de la especificación del cursor. La tabla de

resultados se genera durante el procesamiento del cursor ABIERTO y no refleja ningún cambio futuro. No puede declarar el cursor con las palabras clave SENSITIVE DYNAMIC o SENSITIVE STATIC.

Cuando declara un cursor no desplazable, las actualizaciones o eliminaciones buscadas no afectan a la tabla de resultados del cursor. Las filas de la tabla de resultados se determinan durante el procesamiento del cursor ABIERTO.

Por ejemplo, supongamos que su aplicación declara un cursor, abre el cursor, realiza una obtención, actualiza la tabla y, a continuación, obtiene filas adicionales:

```
EXEC SQL DECLARE CS1 CURSOR FOR
  SELECT SALARY
    FROM FINAL TABLE (INSERT INTO EMPSAMP (NAME, SALARY, LEVEL)
                      SELECT NAME, INCOME, BAND FROM OLD_EMPLOYEE);

EXEC SQL OPEN CS1;
EXEC SQL FETCH CS1 INTO :hv_salary;
/* print fetch result */
...
EXEC SQL UPDATE EMPSAMP SET SALARY = SALARY + 500;
while (SQLCODE == 0) {
  EXEC SQL FETCH CS1 INTO :hv_salary;
  /* print fetch result */
  ...
}
```

Las recuperaciones que se producen después de las actualizaciones devuelven las filas que se generaron cuando se abrió el cursor. Si utiliza un SELECT simple (sin instrucción INSERT en la cláusula FROM), las recuperaciones podrían devolver los valores actualizados, dependiendo de la ruta de acceso que utilice Db2 .

Ejemplo 11: Efecto de WITH HOLD

Cuando declara un cursor con la opción CON RETENCIÓN y abre el cursor, todas las filas se insertan en la tabla de destino. La opción WITH HOLD no tiene efecto en la instrucción SELECT FROM INSERT de la definición del cursor. Después de que su aplicación realice una confirmación, puede continuar recuperando todas las filas insertadas.

Supongamos que la tabla de empleados de la aplicación de muestra Db2 tiene cinco filas. Su aplicación declara un cursor CON RETENCIÓN, abre el cursor, recupera dos filas, realiza una confirmación y, a continuación, recupera la tercera fila correctamente:

```
EXEC SQL DECLARE CS2 CURSOR WITH HOLD FOR
  SELECT EMP_ROWID
    FROM FINAL_TABLE (INSERT INTO DSN8C10.EMP_PHOTO_RESUME (EMPNO)
                      SELECT EMPNO FROM DSN8C10.EMP);
EXEC SQL OPEN CS2;          /* Inserts 5 rows */
EXEC SQL FETCH CS2 INTO :hv_rowid; /* Retrieves ROWID for 1st row */
EXEC SQL FETCH CS2 INTO :hv_rowid; /* Retrieves ROWID for 2nd row */
EXEC SQL COMMIT;           /* Commits 5 rows */
EXEC SQL FETCH CS2 INTO :hv_rowid; /* Retrieves ROWID for 3rd row */
```

Ejemplo 12: Efecto de SAVEPOINT y ROLLBACK

Un punto de guardado es un punto en el tiempo dentro de una unidad de recuperación al que se pueden revertir los cambios de la base de datos relacional. Puede establecer un punto de guardado con la instrucción SAVEPOINT.

Cuando estableces un punto de guardado antes de abrir el cursor y luego vuelves a ese punto de guardado, se deshacen todas las inserciones.

Supongamos que su aplicación declara un cursor, establece un punto de guardado, abre el cursor, establece otro punto de guardado, retrocede al segundo punto de guardado y luego retrocede al primer punto de guardado:

```
EXEC SQL DECLARE CS3 CURSOR FOR
  SELECT EMP_ROWID
    FROM FINAL_TABLE (INSERT INTO DSN8C10.EMP_PHOTO_RESUME (EMPNO)
                      SELECT EMPNO FROM DSN8C10.EMP);
EXEC SQL SAVEPOINT A ON ROLLBACK RETAIN CURSORS;      /* Sets 1st savepoint */
EXEC SQL OPEN CS3;
EXEC SQL SAVEPOINT B ON ROLLBACK RETAIN CURSORS;      /* Sets 2nd savepoint */
...
```

```

EXEC SQL ROLLBACK TO SAVEPOINT B; /* Rows still in DSN8C10.EMP_PHOTO_RESUME */
...
EXEC SQL ROLLBACK TO SAVEPOINT A; /* All inserted rows are undone */

```

Ejemplo 13: Errores durante el procesamiento SELECT INTO

En un programa de aplicación, cuando se insertan una o más filas en una tabla mediante la instrucción SELECT FROM INSERT, la tabla resultante de la operación de inserción puede verse afectada o no, dependiendo de dónde se haya producido el error en el procesamiento de la aplicación.

Si el procesamiento de inserción o el procesamiento de selección falla durante una instrucción SELECT INTO, no se insertan filas en la tabla de destino y no se devuelven filas de la tabla de resultados de la operación de inserción. Por ejemplo, supongamos que la tabla de empleados de la aplicación de muestra Db2 tiene una fila y que la columna SALARIO tiene un valor de 99999000.00.

```

EXEC SQL SELECT EMPNO INTO :hv_empno
  FROM FINAL TABLE (INSERT INTO EMPSAMP (NAME, SALARY)
                     SELECT FIRSTNAME || MIDINIT || LASTNAME,
                           SALARY + 10000.00
                      FROM DSN8C10.EMP)

```

La adición de 10000.00 provoca un desbordamiento decimal y no se insertan filas en la tabla EMPSAMP.

Ejemplo 14: Errores durante el procesamiento del cursor ABIERTO

Si falla la inserción de cualquier fila durante el procesamiento del cursor ABRIR, se desharán todas las inserciones que se hayan realizado correctamente anteriormente. La tabla de resultados de la inserción está vacía.

Ejemplo 15: Errores durante el procesamiento FETCH

Si la sentencia FETCH falla al recuperar filas de la tabla de resultados de la operación de inserción, se devuelve un SQLCODE negativo a la aplicación, pero la tabla de resultados sigue conteniendo el número original de filas que se determinó durante el procesamiento del cursor OPEN. En este punto, puede deshacer todas las inserciones.

Supongamos que la tabla de resultados contiene 100 filas y la fila " 90th " que se está recuperando del cursor devuelve un SQLCODE negativo:

```

EXEC SQL DECLARE CS1 CURSOR FOR
  SELECT EMPNO
    FROM FINAL TABLE (INSERT INTO EMPSAMP (NAME, SALARY)
                       SELECT FIRSTNAME || MIDINIT || LASTNAME, SALARY + 10000.00
                          FROM DSN8C10.EMP);
EXEC SQL OPEN CS1; /* Inserts 100 rows */
while (SQLCODE == 0)
  EXEC SQL FETCH CS1 INTO :hv_empno;
  if (SQLCODE == -904) /* If SQLCODE is -904, undo all inserts */
    EXEC SQL ROLLBACK;
  else /* Else, commit inserts */
    EXEC SQL COMMIT;

```

Conceptos relacionados

Cursos retenidos y no retenidos

Un cursor retenido no se cierra después de una operación de confirmación. Un cursor no retenido se cierra después de una operación de confirmación. Especifica si desea que un cursor esté retenido o no incluyendo u omitiendo la cláusula WITH HOLD cuando declara el cursor.

Uso de variables host en sentencias SQL

Utilice variables de host escalares en sentencias de SQL incorporado para representar un único valor. Las variables host son útiles para almacenar datos recuperados o para pasar valores que van a asignar o utilizar las comparaciones.

Columnas de identidad

Una columna de identidad contiene un valor numérico exclusivo para cada fila de la tabla. Db2 puede generar automáticamente valores numéricos secuenciales para esta columna cuando las filas se insertan en la tabla. Así, las columnas de identidad son ideales para los valores de clave primaria, como los números de empleado o los números de producto.

Tipos de cursores

Puede declarar cursores colocados por fila o por conjunto de filas de varias formas. Estos cursores pueden ser desplazables o no desplazables, retenidos o no retenidos, o retornables o no retornables.

Tareas relacionadas

[Inserción de varias filas de datos desde matrices de variables de host](#)

Utilice las variables de host en la sentencia INSERT cuando no conozca al menos algunos de los valores que se han de insertar hasta que el programa se ejecuta.

[Recuperación de un conjunto de filas utilizando un cursor](#)

En un programa de aplicación, puede recuperar un conjunto de filas de una tabla o una tabla de resultados que devuelve el procedimiento almacenado. Puede recuperar una o más filas a la vez.

[Deshacer los cambios seleccionados en una unidad de trabajo utilizando puntos de salvaguarda](#)

Los puntos de guardado le permiten deshacer los cambios seleccionados dentro de una unidad de trabajo. Su aplicación puede establecer cualquier número de puntos de salvaguarda y, a continuación, especificar un punto de salvaguarda para indicar los cambios que se han de deshacer en la unidad de trabajo.

Referencia relacionada

[Db2 command line processor BIND, mandato](#)

Utilice el comando BIND de Db2 command line processor para vincular DBRM que están en z/OS Archivos HFS de UNIX a paquetes.

Preservar el orden de una tabla derivada

Cuando especifique SELECT FROM INSERT, SELECT FROM UPDATE, SELECT FROM DELETE o SELECT FROM MERGE, puede conservar el orden de la tabla derivada. Esta acción garantiza que las filas de resultados de una fullselect sigan el mismo orden que la tabla de resultados de una subconsulta dentro de la fullselect.

Procedimiento

Para conservar el orden de la tabla derivada, especifique la cláusula ORDER OF con la cláusula ORDER BY.

Estas dos cláusulas garantizan que las filas de resultados de un fullselect sigan el mismo orden que la tabla de resultados de una subconsulta dentro del fullselect.

Puede utilizar la cláusula ORDER OF en cualquier consulta que utilice una cláusula ORDER BY, pero la cláusula ORDER OF es más útil con consultas que contienen un operador de conjunto, como UNION.

ejemplos

Ejemplo

El siguiente ejemplo recupera las siguientes filas:

- Filas de la tabla T1 en ningún orden específico
- Filas de la tabla T2 en el orden de la primera columna de la tabla T2

La consulta de ejemplo realiza entonces una operación UNION ALL en los resultados de las dos subconsultas. La cláusula ORDER BY ORDER OF UTABLE en la consulta especifica que las filas de resultados fullselect deben devolverse en el mismo orden que las filas de resultados de la sentencia UNION ALL.

```
SELECT * FROM
  (SELECT * FROM T1
    UNION ALL
    (SELECT * FROM T2 ORDER BY 1)
  ) AS UTABLE
ORDER BY ORDER OF UTABLE;
```

Ejemplo

El siguiente ejemplo une datos de la tabla T1 a la tabla de resultados de una expresión de tabla anidada. La expresión de la tabla anidada está ordenada por la segunda columna en la tabla T2. La

cláusula ORDER BY ORDER OF TEMP en la consulta especifica que las filas de resultados fullselect deben devolverse en el mismo orden que la expresión de tabla anidada.

```
SELECT T1.C1, T1.C2, TEMP.Cy, TEMP.Cx
FROM T1, (SELECT T2.C1, T2.C2 FROM T2 ORDER BY 2) as TEMP(Cx, Cy)
WHERE Cy = T1.C1
ORDER BY ORDER OF TEMP;
```

Alternativamente, puede producir el mismo resultado indicando explícitamente el ORDER BY column TEMP.Cy en el fullselect en lugar de utilizar la sintaxis ORDER OF.

```
SELECT T1.C1, T1.C2, TEMP.Cy, TEMP.Cx
FROM T1, (SELECT T2.C1, T2.C2 FROM T2 ORDER BY 2) as TEMP(Cx, Cy)
WHERE Cy = T1.C1
ORDER BY TEMP.Cy;
```

Referencia relacionada

[fullselect \(Db2 SQL\)](#)

[cláusula-order-by \(Db2 SQL\)](#)

Añadir datos al final de una tabla

En una base de datos relacional, las filas de una tabla no están ordenadas y, por lo tanto, la tabla "no tiene fin". Sin embargo, dependiendo de su objetivo, puede realizar varias acciones para simular la adición de datos al final de una tabla.

Acerca de esta tarea

Pregunta : ¿Cómo puedo añadir datos al final de una tabla?

Respuesta : Aunque la pregunta se hace a menudo, no tiene sentido en una base de datos relacional. Las filas de una tabla base no están ordenadas; por lo tanto, la tabla no tiene "un final".

Sin embargo, dependiendo de su objetivo, puede realizar una de las siguientes acciones para simular la adición de datos al final de una tabla:

- Si su objetivo es obtener una tabla de resultados ordenada según el momento en que se insertaron las filas, defina un índice único en una columna TIMESTAMP en la definición de la tabla. A continuación, cuando recuperes datos de la tabla, utiliza una cláusula ORDER BY que nombre esa columna. La inserción más reciente aparece en último lugar.
- Si su objetivo es que Db2 inserte filas en el siguiente espacio libre disponible, sin conservar el orden de agrupación, especifique la opción APPEND YES cuando cree o modifique la tabla. Especificar esta opción podría reducir el tiempo que se tarda en insertar filas, ya que Db2 no pierde tiempo buscando espacio libre.

Almacenamiento de datos que no tienen un formato tabular

Db2 proporciona varias opciones para almacenar grandes volúmenes de datos que no están definidos como un conjunto de columnas en una tabla.

Acerca de esta tarea

Pregunta : ¿Cómo puedo almacenar un gran volumen de datos que no se definen como un conjunto de columnas en una tabla?

Respuesta : Puede almacenar los datos en una tabla en una cadena binaria, un LOB o una columna XML.

Actualización de datos de tabla

Puede cambiar un valor de columna a otro valor o suprimir el valor de columna.

Procedimiento

Para cambiar los datos de una tabla, utilice la instrucción UPDATE.

Por ejemplo, supongamos que un empleado se traslada. Para actualizar varios elementos de los datos del empleado en la tabla de trabajo YEMP para reflejar el traslado, puede ejecutar la siguiente instrucción:

```
UPDATE YEMP  
  SET JOB = 'MANAGER ',  
      PHONENO = '5678'  
 WHERE EMPNO = '000400';
```

También puede utilizar la instrucción UPDATE para eliminar un valor de una columna (sin eliminar la fila) cambiando el valor de la columna a nulo.

No puede actualizar filas en una tabla temporal creada, pero puede actualizar filas en una tabla temporal declarada.

La cláusula SET nombra las columnas que desea actualizar y proporciona los valores que desea asignar a esas columnas. Puede reemplazar un valor de columna en la cláusula SET con cualquiera de los siguientes elementos:

- Un valor nulo

La columna a la que asignas el valor nulo no debe definirse como NOT NULL.

- Una expresión, que puede ser cualquiera de los siguientes elementos:

- Una columna
- Una constante
- una selección completa escalar
- Una variable del lenguaje principal
- Un registro especial

- Un valor predeterminado. Si especifica DEFAULT, Db2 determina el valor en función de cómo se defina la columna correspondiente en la tabla.

Además, puede reemplazar uno o más valores de columna en la cláusula SET con los valores de columna en una fila que devuelve un fullselect.

A continuación, identifique las filas que desea actualizar:

- Para actualizar una sola fila, utilice una cláusula WHERE que localice una, y solo una, fila.
- Para actualizar varias filas, utilice una cláusula WHERE que localice solo las filas que desea actualizar.

Si omite la cláusula WHERE, Db2 actualiza **cada fila** de la tabla o vista con los valores que usted proporciona.

Si Db2 encuentra un error al ejecutar la instrucción UPDATE (por ejemplo, un valor de actualización demasiado grande para la columna), detiene la actualización y devuelve un error. No hay filas en la tabla que cambien. Las filas que ya se habían cambiado, si las hay, se restauran a sus valores anteriores. Si la instrucción UPDATE se ejecuta correctamente, SQLERRD(3) se establece en el número de filas que se actualizan.

Ejemplo de instrucciones UPDATE

- La siguiente declaración proporciona una inicial intermedia que falta y cambia el trabajo del empleado 000200.

```
UPDATE YEMP  
  SET MIDINIT = 'H', JOB = 'FIELDREP'  
 WHERE EMPNO = '000200';
```

- La siguiente declaración otorga a todos los miembros del departamento de D11 un aumento de 400.00. El extracto puede actualizar varias filas.

```
UPDATE YEMP  
SET SALARY = SALARY + 400.00  
WHERE WORKDEPT = 'D11';
```

- La siguiente declaración establece el salario del empleado 000190 en el salario medio y establece la bonificación en la bonificación mínima para todos los empleados.

```
UPDATE YEMP  
SET (SALARY, BONUS) =  
(SELECT AVG(SALARY), MIN(BONUS)  
FROM EMP)  
WHERE EMPNO = '000190';
```

Referencia relacionada

[UPDATE declaración \(Db2 SQL\)](#)

Selección de valores al actualizar datos

Cuando actualiza filas en una tabla, puede seleccionar los valores actualizados de esas filas al mismo tiempo.

Procedimiento

Especifique la instrucción UPDATE en la cláusula FROM de la instrucción SELECT.

Cuando actualiza una o más filas en una tabla, puede recuperar:

- El valor de una columna generada automáticamente, como una ROWID o una columna de identidad
- Cualquier valor predeterminado para las columnas
- Todos los valores de una fila actualizada, sin especificar nombres de columnas individuales

En la mayoría de los casos, se recomienda utilizar la cláusula FINAL TABLE con las sentencias SELECT FROM UPDATE. La TABLA FINAL consiste en las filas de la tabla o vista después de que se produce la actualización.

ejemplos

Ejemplo: SELECCIONAR DE LA TABLA FINAL

Supongamos que todos los empleados de una empresa están recibiendo aumentos del 5 por ciento. Puede utilizar la siguiente instrucción SELECT FROM UPDATE para aumentar el salario de cada diseñador en un 5 % y obtener el aumento salarial total de la empresa.

```
SELECT SUM(SALARY) INTO :salary FROM FINAL TABLE  
(UPDATE EMP SET SALARY = SALARY * 1.05  
WHERE JOB = 'DESIGNER');
```

Ejemplo: recuperar datos fila por fila a partir de datos actualizados

Para recuperar la salida fila por fila de los datos actualizados, utilice un cursor con una instrucción SELECT FROM UPDATE. Por ejemplo, supongamos que todos los diseñadores de una empresa están recibiendo un aumento del 30 % en su bonificación. Puede utilizar la siguiente instrucción SELECT FROM UPDATE para aumentar la bonificación de cada empleado en un 30 por ciento y recuperar la bonificación de cada empleado.

```
DECLARE CS1 CURSOR FOR  
SELECT LASTNAME, BONUS FROM FINAL TABLE  
(UPDATE EMP SET BONUS = BONUS * 1.3  
WHERE JOB = 'CLERK');  
FETCH CS1 INTO :lastname, :bonus;
```

Ejemplo: INCLUIR una nueva columna en la tabla de resultados, pero no en la tabla de destino

Puede utilizar la cláusula INCLUDE para introducir una nueva columna en la tabla de resultados, pero no para añadir la columna a la tabla de destino. Por ejemplo, supongamos que los representantes de ventas recibieron un aumento del 20 por ciento en su comisión. Debe actualizar la comisión (COMM) de los representantes de ventas (SALESREP) en la tabla EMP y recuperar la comisión antigua

y la nueva comisión de cada representante de ventas. Puede utilizar la siguiente instrucción SELECT FROM UPDATE para realizar la actualización y recuperar los datos necesarios.

```
DECLARE CS2 CURSOR FOR
SELECT LASTNAME, COMM, old_comm FROM FINAL TABLE
(UPDATE EMP INCLUDE(old_comm DECIMAL (7,2))
 SET COMM = COMM * 1.2, old_comm = COMM
 WHERE JOB = 'SALESREP');
```

Referencia relacionada

[table-reference \(Db2 SQL\)](#)

[UPDATE declaración \(Db2 SQL\)](#)

Actualización de miles de filas

Cuando actualice grandes volúmenes de datos, tenga en cuenta ciertas acciones recomendadas para aumentar la concurrencia.

Acerca de esta tarea

Pregunta : ¿Existen técnicas especiales para actualizar grandes volúmenes de datos?

Respuesta : Sí. Al actualizar grandes volúmenes de datos utilizando un cursor, puede minimizar la cantidad de tiempo que mantiene bloqueados los datos declarando el cursor con la opción HOLD y realizando confirmaciones con frecuencia.

Supresión de datos de tablas

Puede suprimir datos de una tabla suprimiendo una o más filas de la tabla, suprimiendo todas las filas de la tabla o descartando las columnas de la tabla.

Procedimiento

Para eliminar una o más filas de una tabla:

- Utilice la instrucción DELETE con una cláusula WHERE para especificar una condición de búsqueda.

La sentencia DELETE elimina cero o más filas de una tabla, dependiendo de cuántas filas cumplen la condición de búsqueda que se especifica en la cláusula WHERE.

Puede utilizar DELETE con una cláusula WHERE para eliminar solo las filas seleccionadas de una tabla temporal declarada, pero no de una tabla temporal creada.

La siguiente instrucción DELETE elimina cada fila de la tabla YEMP que tenga un número de empleado «000060».

```
DELETE FROM YEMP
WHERE EMPNO = '000060';
```

Cuando se ejecuta esta sentencia, Db2 elimina de la tabla YEMP cualquier fila que cumpla la condición de búsqueda.

Si Db2 encuentra un error al ejecutar la instrucción DELETE, detiene la eliminación de datos y devuelve códigos de error en las variables SQLCODE y SQLSTATE o en los campos relacionados en SQLCA. Los datos de la tabla no cambian.

Si la ELIMINACIÓN se realiza correctamente, SQLERRD(3) en el SQLCA contiene el número de filas eliminadas. Este número incluye solo el número de filas eliminadas en la tabla que se especifica en la instrucción DELETE. Las filas que se eliminan (en otras tablas) de acuerdo con la regla CASCADE no se incluyen en SQLERRD(3).

Para eliminar todas las filas de una tabla:

- Utilice la instrucción DELETE sin especificar una cláusula WHERE.

Con espacios de tablas segmentados, eliminar todas las filas de una tabla es muy rápido.

La siguiente instrucción DELETE elimina todas las filas de la tabla YDEPT:

```
DELETE FROM YDEPT;
```

Si la instrucción se ejecuta, la tabla sigue existiendo (es decir, se pueden insertar filas en ella), pero está vacía. Todas las vistas y autorizaciones existentes en la tabla permanecen intactas al utilizar DELETE.

- Utilice la instrucción TRUNCATE.

La sentencia TRUNCATE puede proporcionar las siguientes ventajas sobre una sentencia DELETE:

- La instrucción TRUNCATE puede ignorar los desencadenantes de eliminación
- La instrucción TRUNCATE puede realizar una confirmación inmediata
- La sentencia TRUNCATE puede mantener el almacenamiento asignado para la tabla

Sin embargo, la instrucción TRUNCATE no restablece el recuento de un valor generado automáticamente para una columna de identidad en la tabla. Si 14872 fuera el siguiente valor de columna de identidad que se generara antes de una sentencia TRUNCATE, 14872 sería el siguiente valor generado después de la sentencia TRUNCATE.

Supongamos que necesita vaciar los datos de una tabla de inventario antigua, independientemente de cualquier activador de eliminación existente, y que necesita hacer que el espacio asignado a la tabla esté disponible para otros usos. Utilice la siguiente instrucción TRUNCATE.

```
TRUNCATE INVENTORY_TABLE  
IGNORE DELETE TRIGGERS  
DROP STORAGE;
```

Supongamos que necesita vaciar los datos de una tabla de inventario antigua de forma permanente, independientemente de los desencadenantes de eliminación existentes, y que necesita conservar el espacio asignado a la tabla. Necesitas que los datos vaciados no estén disponibles en absoluto, de modo que una sentencia ROLLBACK no pueda devolverlos. Utilice la siguiente instrucción TRUNCATE.

```
TRUNCATE INVENTORY_TABLE  
REUSE STORAGE  
IGNORE DELETE TRIGGERS  
IMMEDIATE;
```

- Utilice la instrucción DROP TABLE.

DROP TABLE elimina la tabla especificada y todas las vistas y autorizaciones relacionadas, lo que puede invalidar planes y paquetes.

Para eliminar columnas de una tabla:

- Utilice la instrucción ALTER TABLE con la cláusula DROP COLUMN.

Dado que eliminar una columna de una tabla es un cambio pendiente en la definición de la tabla, el espacio de la tabla se coloca en estado de aviso REORG-pending (AREOR). Cuando se aplica el cambio pendiente (ejecutando la utilidad REORG con las opciones SHRLEVEL CHANGE o REFERENCE), la columna se elimina de la tabla y se invalidan todos los paquetes y sentencias dependientes de la caché de sentencias dinámicas.

Conceptos relacionados

[Área de comunicación de SQL \(SQLCA\) \(Db2 SQL\)](#)

Tareas relacionadas

[Descarte de tablas](#)

Cuando descarta una tabla, suprime los datos y la definición de tabla. También suprime todos los sinónimos, vistas, índices, restricciones referenciales y restricciones de comprobación asociadas con esa tabla.

Referencia relacionada

[DELETE declaración \(Db2 SQL\)](#)

[DROP declaración \(Db2 SQL\)](#)

[TRUNCATE declaración \(Db2 SQL\)](#)
[ALTER TABLE declaración \(Db2 SQL\)](#)

Selección de valores al eliminar datos

Cuando elimina filas de una tabla, puede seleccionar los valores de esas filas al mismo tiempo.

Procedimiento

Especifique la instrucción DELETE en la cláusula FROM de la instrucción SELECT.

Cuando elimina una o más filas en una tabla, puede recuperar:

- Cualquier valor predeterminado para las columnas
- Todos los valores de una fila eliminada, sin especificar los nombres de las columnas individuales
- Valores calculados basados en filas eliminadas

Ejemplo

Ejemplo: cláusula FROM OLD TABLE

Cuando utilice una instrucción SELECT FROM DELETE, debe utilizar la cláusula FROM OLD TABLE para recuperar los valores eliminados. La VIEJA TABLA consiste en las filas de la tabla o vista antes de que se produzca la eliminación. Por ejemplo, supongamos que una empresa está eliminando todos los puestos de operador y que la empresa quiere saber cuánto dinero ahorrará en salarios al eliminar estos puestos. Puede utilizar la siguiente instrucción SELECT FROM DELETE para eliminar operadores de la tabla EMP y recuperar la suma de los salarios de los operadores.

```
SELECT SUM(SALARY) INTO :salary FROM OLD TABLE  
  (DELETE FROM EMP  
    WHERE JOB = 'OPERATOR');
```

Ejemplo: recuperar la salida fila por fila de los datos eliminados

Para recuperar la salida fila por fila de los datos eliminados, utilice un cursor con una instrucción SELECT FROM DELETE. Por ejemplo, supongamos que una empresa está eliminando todos los puestos de analista y que la empresa quiere saber cuántos años de experiencia tiene cada analista en la empresa. Puede utilizar la siguiente instrucción SELECT FROM DELETE para eliminar analistas de la tabla EMP y recuperar la experiencia de cada analista.

```
DECLARE CS1 CURSOR FOR  
SELECT YEAR(CURRENT DATE - HIREDATE) FROM OLD TABLE  
  (DELETE FROM EMP  
    WHERE JOB = 'ANALYST');  
FETCH CS1 INTO :years_of_service;
```

Ejemplo: recuperación de una base de datos calculada a partir de una tabla eliminada

Si necesita recuperar datos calculados, basados en los datos que elimina, pero no añade esa columna a la tabla de destino. Por ejemplo, supongamos que necesita eliminar gerentes de la tabla EMP y que necesita recuperar el salario y los años de empleo de cada gerente. Puede utilizar la siguiente instrucción SELECT FROM DELETE para realizar la operación de eliminación y recuperar los datos necesarios.

```
DECLARE CS2 CURSOR FOR  
SELECT LASTNAME, SALARY, years_employed FROM OLD TABLE  
  (DELETE FROM EMP INCLUDE(years_employed INTEGER)  
    SET years_employed = YEAR(CURRENT DATE - HIREDATE)  
    WHERE JOB = 'MANAGER');
```

Referencia relacionada

[table-reference \(Db2 SQL\)](#)
[DELETE declaración \(Db2 SQL\)](#)

Acceso a datos en tablas desde programas de aplicación

El programa puede utilizar varias técnicas diferentes para leer los datos de cualquiera de las tablas de Db2 para las que dispone de acceso de lectura. La técnica más sencilla es utilizar sentencias SELECT básicas de SQL . No obstante, debe elegir la técnica que mejor funcione con su situación y tenga un comportamiento correcto.

Acerca de esta tarea

Consejo: Las herramientas de desarrollo de aplicaciones como [IBM Db2 for z/OS Developer Extension](#) pueden ayudarle con esta tarea.

Conceptos relacionados

[Investigación del rendimiento de SQL utilizando EXPLAIN \(Db2 Performance\)](#)

[Interpretación del acceso a datos mediante el uso de EXPLAIN \(Db2 Performance\)](#)

Tareas relacionadas

[Escritura eficiente de consultas SQL \(Db2 Performance\)](#)

Determinar a qué mesas tiene acceso

Puede solicitar a Db2 que le indique las mesas a las que tiene acceso un ID de autorización específico.

Acerca de esta tarea

El contenido de las tablas del catálogo Db2 puede ser una herramienta de referencia útil cuando empieza a desarrollar una instrucción SQL o un programa de aplicación.

La tabla de catálogos, SYSIBM.SYSTABAUTH, enumera los privilegios de tabla que se conceden a los ID de autorización. Para mostrar las tablas a las que tiene autorización para acceder (por privilegios otorgados a su ID de autorización o a PÚBLICO), puede ejecutar una instrucción SQL similar a la que se muestra en el siguiente ejemplo. Para ello, debe tener el privilegio SELECT en SYSIBM.SYSTABAUTH.

Procedimiento

Emitir una instrucción SELECT similar al siguiente ejemplo. Para ello, debe tener el privilegio SELECT en SYSIBM.SYSTABAUTH.

```
SELECT DISTINCT TCREATOR, TTNAME  
  FROM SYSIBM.SYSTABAUTH  
 WHERE GRANTEE IN (USER, 'PUBLIC', 'PUBLIC*') AND GRANTEETYPE = ' ';
```

En esta consulta, el predicado GRANTEETYPE = ' ' selecciona los ID de autorización.

Excepción: Si su subsistema de Db2 utiliza una rutina de salida para la autorización de control de acceso, no puede confiar en las consultas de catálogo para saber a qué tablas puede acceder. Cuando se instala una rutina de salida de este tipo, tanto RACF y Db2 controlan el acceso a la tabla.

Referencia relacionada

[Tabla de catálogo SYSTABAUTH \(Db2 SQL\)](#)

[Privilegios de tabla y vista explícitos \(Managing Security\)](#)

Mostrar información sobre las columnas de una tabla determinada

Puede solicitar a Db2 que enumere las columnas de una tabla concreta y cierta información sobre dichas columnas.

Acerca de esta tarea

La tabla de catálogo, SYSIBM.SYSCOLUMNS, describe cada columna de cada tabla.

Procedimiento

Consulta la tabla del catálogo de SYSIBM.SYSCOLUMNS.

ejemplos

Ejemplo

Supongamos que desea mostrar información sobre la tabla DSN8C10.DEPT. Si tiene el privilegio SELECT en SYSIBM.SYSCOLUMNS, puede utilizar la siguiente instrucción:

```
SELECT NAME, COLTYPE, SCALE, LENGTH
  FROM SYSIBM.SYSCOLUMNS
 WHERE TBNAME = 'DEPT'
   AND TBCREATOR = 'DSN8C10';
```

Ejemplo

Si muestra información de columna sobre una tabla que incluye columnas LOB o ROWID, el campo LENGTH de esas columnas contiene el número de bytes que ocupan esas columnas en la tabla base. El campo LENGTH no contiene la longitud de los datos LOB o ROWID.

Para determinar la longitud máxima de datos para una columna LOB o ROWID, incluya la columna "LENGTH2" en su consulta:

```
SELECT NAME, COLTYPE, LENGTH, LENGTH2
  FROM SYSIBM.SYSCOLUMNS
 WHERE TBNAME = 'EMP_PHOTO_RESUME'
   AND TBCREATOR = 'DSN8C10';
```

Referencia relacionada

[Tabla de catálogo SYSCOLUMNS \(Db2 SQL\)](#)

Recuperación de datos utilizando la sentencia SELECT

La forma más sencilla de recuperar datos es utilizar la sentencia SQL SELECT para especificar una tabla de resultados. Puede especificar las columnas y filas que desea recuperar.

Antes de empezar

Considere desarrollar sus propias sentencias SQL similares a los ejemplos de esta sección y, a continuación, ejecútelas dinámicamente utilizando SPUFI. Para ver un tutorial, visite [Lección 1.1: Consulta interactiva de datos \(Introducción a Db2 para z/OS\)](#).

También puede utilizar el Db2 command line processor o el Db2 Query Management Facility (QMF).

Procedimiento

Emitir una instrucción SELECT.

ejemplos

Ejemplo 1: Selección de todas las columnas con SELECT *

No es necesario que conozca los nombres de columna para seleccionar datos de Db2. Utilice un asterisco (*) en la cláusula SELECT para indicar que desea recuperar todas las columnas de cada fila seleccionada de la tabla nombrada. Las columnas implícitamente ocultas, como las columnas ROWID y las columnas de identificación de documentos XML, no se incluyen en el resultado de la instrucción SELECT *. Para ver los valores de estas columnas, debe especificar el nombre de la columna.

La siguiente instrucción SQL selecciona todas las columnas de la tabla de departamentos:

```
SELECT *
  FROM DSN8C10.DEPT;
```

La tabla de resultados tiene un aspecto similar al siguiente:

DEPTNO	DEPTNAME	MGRNO	ADMREDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICES DIV.	000010	A00	-----
B01	PLANNING	000020	A00	-----
C01	INFORMATION CENTER	000030	A00	-----
D01	DEVELOPMENT CENTER	-----	A00	-----
D11	MANUFACTURING CENTER	000060	D01	-----
D21	ADMINISTRATION SYSTEMS	000070	D01	-----
E01	SUPPORT SERVICES	000050	A00	-----
E11	OPERATIONS	000090	E01	-----
E21	SOFTWARE SUPPORT	000100	E01	-----
F22	BRANCH OFFICE F2	-----	E01	-----
G22	BRANCH OFFICE G2	-----	E01	-----
H22	BRANCH OFFICE H2	-----	E01	-----
I22	BRANCH OFFICE I2	-----	E01	-----
J22	BRANCH OFFICE J2	-----	E01	-----

Debido a que el ejemplo no especifica ninguna cláusula WHERE, la sentencia recupera datos de todas las filas.

Los guiones de MGRNO y LOCATION en la tabla de resultados indican valores nulos.

SELECT * se recomienda principalmente para su uso con SQL dinámico y definiciones de vistas. Puede utilizar SELECT * en SQL estático, pero no se recomienda hacerlo debido a posibles implicaciones de compatibilidad de variables de host y rendimiento. Supongamos que añade una columna a la tabla a la que se refiere SELECT *. Si no ha definido una variable de host receptora para esa columna, puede producirse un error o es posible que no se recuperen los datos de la columna añadida.

Si enumera los nombres de las columnas en una instrucción SELECT estática en lugar de utilizar un asterisco, puede evitar los problemas que podrían producirse con SELECT *. También puede ver la relación entre las variables de host receptor y las columnas de la tabla de resultados.

Ejemplo 2: seleccionar columnas específicas con SELECT *nombre-columna*

Seleccione la columna o columnas que desea recuperar nombrando cada columna. Con una sola instrucción SELECT, puede seleccionar datos de una columna o de hasta 750 columnas. Todas las columnas aparecen en el orden que especifique, no en el orden de la tabla.

Por ejemplo, la siguiente instrucción SQL recupera solo las columnas MGRNO y DEPTNO de la tabla de departamentos:

```
SELECT MGRNO, DEPTNO  
  FROM DSN8C10.DEPT;
```

La tabla de resultados tiene un aspecto similar al siguiente:

MGRNO	DEPTNO
000010	A00
000020	B01
000030	C01
-----	D01
000050	E01
000060	D11
000070	D21
000090	E11
000100	E21
-----	F22
-----	G22
-----	H22
-----	I22
-----	J22

Ejemplo 3: Selección de datos de columnas ocultas implícitamente

Para SELECCIONAR datos de columnas implícitamente ocultas, como ROWID e ID de documento XML, busque los nombres de las columnas en SYSIBM.SYSCOLUMNS y especifique estos nombres en la lista SELECT. Por ejemplo, supongamos que crea y rellena la siguiente tabla:

```
CREATE TABLE MEMBERS (MEMBERID INTEGER,  
                      BIO XML,
```

```
REPORT      XML,  
RECOMMENDATIONS      XML);
```

Db2 genera una columna de identificación de documento XML implícitamente oculta adicional. Para recuperar datos en todas las columnas, incluida la columna de ID del documento XML generado, primero busque el nombre de la columna generada en SYSIBM.SYSCOLUMNS. Supongamos que el nombre es DB2_GENERATED_DOCID_FOR_XML. A continuación, especifique la siguiente declaración:

```
SELECT DB2_GENERATED_DOCID_FOR_XML, MEMBERID, BIO,  
REPORT, RECOMMENDATIONS FROM MEMBERS
```

Conceptos relacionados

Variables de host

Utilice variables host para pasar un único elemento de datos entre Db2 y la aplicación.

Servidores remotos y datos distribuidos

Los datos distribuidos son datos que residen en un sistema de gestión de bases de datos (DBMS) distinto de su sistema local. El DBMS local es el sistema en el que se vincula el plan de aplicación. Los demás DBMS son remotos.

Predicados (Db2 SQL)

Referencia relacionada

[sentencia-select \(Db2 SQL\)](#)

Especificar las condiciones de búsqueda con una cláusula WHERE

Puede utilizar una cláusula WHERE para seleccionar las filas que cumplan determinadas condiciones. Una cláusula WHERE especifica una condición de búsqueda. *Una condición de búsqueda* consta de uno o más predicados. *Un predicado* especifica una prueba que desea que Db2 aplique a cada fila de la tabla.

Acerca de esta tarea

Una cláusula WHERE especifica una condición de búsqueda. *Una condición de búsqueda* consta de uno o más predicados. *Un predicado* especifica una prueba que desea que Db2 aplique a cada fila de la tabla.

Procedimiento

Especifique una cláusula WHERE con uno o más predicados.

Db2 evalúa un predicado para cada fila como verdadero, falso o desconocido. Los resultados son desconocidos solo si un operando es nulo.

Si una condición de búsqueda contiene una columna de un tipo distinto, el valor con el que se compara esa columna debe ser del mismo tipo distinto, o debe convertir el valor al tipo distinto.

La siguiente tabla enumera el tipo de comparación, los operadores de comparación y un ejemplo de cada tipo de comparación que puede utilizar en un predicado en una cláusula WHERE.

Tabla 62. Operadores de comparación utilizados en condiciones

Tipo de comparación	Operador de comparación	Ejemplo
Igual a	=	DEPTNO = 'X01'
No es igual que	<>	DEPTNO <> 'X01'
Menor que	<	AVG(SALARY) < 30000
Menor que o igual a	<=	EDAD <= 25
No menor que	>=	EDAD >= 21
Mayor que	>	SALARIO > 2000

Tabla 62. Operadores de comparación utilizados en condiciones (continuación)

Tipo de comparación	Operador de comparación	Ejemplo
Mayor que o igual a	\geq	SALARIO \geq 5000
No mayor que	\leq	SALARIO \leq 5000
Igual a nulo	IS NULL	NÚMERO DE TELÉFONO ES NULO
No es igual a otro valor o un valor es igual a nulo	IS DISTINCT FROM (es distinto de)	PHONENO ES DIFERENTE DE :PHONEHV
Similar a otro valor	LIKE	NAME LIKE 'o' STATUS LIKE 'N_'
Al menos una de dos condiciones	O	FECHA DE CONTRATACIÓN < '1965-01-01' O SALARIO < 16 000
Ambas condiciones	Y	FECHA DE CONTRATACIÓN < '1965-01-01' Y SALARIO < 16 000
Entre dos valores	BETWEEN	SALARY BETWEEN 20000 AND 40000
Igual a un valor de un conjunto	IN (X, Y, Z)	DEPTNO EN ('B01', 'C01', 'D01')

Nota: SALARIO ENTRE 20 000 Y 40 000 es equivalente a SALARIO \geq 20 000 Y SALARIO \leq 40 000.

También puede buscar filas que no cumplan una de las condiciones anteriores utilizando la palabra clave NOT antes de la condición especificada.

Puede buscar filas que no satisfagan el predicado IS DISTINCT FROM utilizando cualquiera de los siguientes predicados:

- $value\ 1\ IS\ NOT\ DISTINCT\ FROM\ value\ 2$
- NOT($value\ 1\ IS\ DISTINCT\ FROM\ value\ 2$)

Estas dos formas del predicado crean una expresión en la que un valor es igual a otro valor o ambos valores son iguales a nulo.

Conceptos relacionados

Subconsultas

Si necesita restringir la condición de búsqueda en función de la información de una tabla temporal, puede utilizar una subconsulta. Por ejemplo, puede que desee buscar todos los números de empleado de una tabla que también existan para un proyecto concreto en una segunda tabla.

Predicados (Db2 SQL)

Tareas relacionadas

Codificación de sentencias SQL para evitar procesos innecesarios (Db2 Performance)

Referencia relacionada

cláusula WHERE (Db2 SQL)

Manejo de valores nulos

Un valor nulo indica la ausencia de valor de columna en una fila. Un valor nulo es un valor desconocido; no es lo mismo que cero o todos los espacios en blanco.

Acerca de esta tarea

Los valores nulos pueden utilizarse como condición en las cláusulas WHERE y HAVING. Por ejemplo, una cláusula WHERE puede especificar una columna que, para algunas filas, contiene un valor nulo. Un predicado de comparación básico que utiliza una columna que contiene valores nulos no selecciona una fila que tiene un valor nulo para la columna. Esto se debe a que un valor nulo no es menor, igual o mayor

que el valor especificado en la condición. El predicado IS NULL se utiliza para comprobar si hay valores nulos.

ejemplos

Ejemplo 1: Selección de filas que contienen nulo en una columna

Para seleccionar los valores de todas las filas que contienen un valor nulo para el número de gerente, puede emitir la siguiente instrucción:

```
SELECT DEPTNO, DEPTNAME, ADMRDEPT  
      FROM DSN8C10.DEPT  
     WHERE MGRNO IS NULL
```

La siguiente tabla muestra el resultado.

DEPTNO	DEPTNAME	ADMRDEPT
D01	DEVELOPMENT CENTER	A00
F22	BRANCH OFFICE F2	E01
G22	BRANCH OFFICE G2	E01
H22	BRANCH OFFICE H2	E01
I22	BRANCH OFFICE I2	E01
J22	BRANCH OFFICE J2	E01

Ejemplo 2: Selección de filas que no contienen un valor nulo

Para obtener las filas que no tienen un valor nulo para el número de gerente, puede cambiar la cláusula WHERE en el ejemplo anterior de la siguiente manera:

```
WHERE MGRNO IS NOT NULL
```

Ejemplo 3: Comparación de valores que contienen el valor NULL

Otro predicado que resulta útil para comparar valores que pueden contener el valor NULL es el predicado DISTINCT. La comparación de dos columnas mediante una comparación normal de igualdad (COL1 = COL2) será verdadera si ambas columnas contienen un valor igual no nulo. Si ambas columnas son nulas, el resultado será falso porque nulo nunca es igual a ningún otro valor, ni siquiera a otro valor nulo. Al utilizar el predicado DISTINCT, los valores nulos se consideran iguales. Por lo tanto, « COL1 » NO ES DIFERENTE de « COL2 » será verdadero si ambas columnas contienen un valor no nulo igual y también cuando ambas columnas son el valor nulo.

Por ejemplo, supongamos que desea seleccionar información de dos tablas que contienen valores nulos. La primera tabla T1 tiene una columna C1 con los siguientes valores.

C1
2
1
nulo

La segunda tabla tiene la columna " C2 " con los siguientes valores.

C2
2
nulo

Suponga que emite la siguiente sentencia SELECT:

```
SELECT *  
  FROM T1, T2  
 WHERE C1 IS DISTINCT FROM C2
```

El resultado es el siguiente.

C1	C2
1	2
1	-
2	-
-	2

Conceptos relacionados

[Tipos de datos en Db2 for z/OS \(Db2 SQL\)](#)

[Valores nulos en columnas de tabla \(Introducción a Db2 para z/OS\)](#)

Cómo comprobar si hay valores nulos

Antes de recuperar un valor de columna, es posible que primero desee determinar si el valor de columna es nulo.

Las aplicaciones suelen necesitar comprobar dos valores para ver si son iguales o no. Puede utilizar un predicado básico para hacer una comparación de igualdad o desigualdad. Una comparación igual o una comparación no igual puede devolver verdadero, falso o desconocido. La regla normal en SQL, excepto para el predicado DISTINCT, es que un valor nulo nunca es igual a otro valor nulo. Si uno o ambos operandos de un predicado básico son el valor nulo, el resultado es desconocido.

Dependiendo de su aplicación, es posible que desee incluir o excluir filas que tengan un valor NULL en una columna. Para ello, puede utilizar el predicado NULL.

MY_EMP es una tabla que tiene una fila con el apellido y el número de teléfono de cada uno de los empleados de una empresa. En esta empresa, ningún empleado comparte un número de teléfono, pero es posible que algunos empleados no tengan número de teléfono. La columna APELLIDOS contiene el apellido de cada empleado. La columna NÚMERO DE TELEFONO contiene el número de teléfono de cada empleado. Si un empleado no tiene teléfono, el valor de la columna PHONENO es NULL. La tabla podría tener este aspecto:

LASTNAME	PHONENO
HAAS	-----
THOMPSON	3476
KWAN	4738
GEYER	6789
STERN	6423

Supongamos que quiere saber el apellido del empleado que no tiene número de teléfono. Usar una consulta como esta no funciona, porque si el valor de la columna PHONENO es NULL, la cláusula WHERE compara un valor de columna NULL con un valor de variable de host nulo. Se desconoce el resultado de esa comparación.

```
MOVE -1 TO PHONENO-IND.  
EXEC SQL  
  SELECT LASTNAME  
  INTO :LASTNAME-HV  
  FROM MY_EMP
```

```
WHERE PHONENO = :PHONENO-HV :PHONENO-IND  
END-EXEC.
```

Para encontrar al empleado con un valor NULL para el número de teléfono, debe utilizar un predicado NULL:

```
EXEC SQL  
  SELECT LASTNAME  
    INTO :LASTNAME-HV  
   FROM MY_EMP  
 WHERE PHONENO IS NULL  
END-EXEC.
```

La sentencia SELECT devuelve un valor LASTNAME de 'HAAS'.

Ahora suponga que desea seleccionar el apellido de un empleado cuyo número de teléfono coincide con un determinado valor o cuyo número de teléfono es NULL. Para ello, debe codificar dos condiciones de búsqueda: una para manejar el caso en el que el número de teléfono no es NULL, y otra para manejar el caso en el que el número de teléfono es NULL. La instrucción SELECT puede tener este aspecto:

```
EXEC SQL  
  SELECT LASTNAME  
    INTO :LASTNAME-HV  
   FROM MY_EMP  
 WHERE (PHONENO IS NOT NULL AND :PHONENO-HV :PHONENO-IND IS NOT NULL  
       AND PHONENO = :PHONENO-HV )  -- Search condition for non-NULL  
                                -- phone number  
     OR  
 (PHONENO IS NULL AND :PHONENO-HV :PHONENO-IND IS NULL)  
                                -- Search condition for NULL  
                                -- phone number  
END-EXEC.
```

Si establece :PHONENO-HV en '3476' y :PHONENO-IND en 0, la instrucción SELECT devuelve 'THOMPSON' porque se utiliza la condición de búsqueda para un número de teléfono no nulo. Si establece :PHONENO-HV en cualquier valor y :PHONENO-IND en -1, la instrucción SELECT devuelve 'HAAS' porque se utiliza la condición de búsqueda para un número de teléfono nulo.

Tareas relacionadas

[Declaración de variables host y variables de indicación](#)

Puede utilizar variables host y variables de indicación en sentencias SQL del programa para pasar datos entre Db2 y la aplicación.

Referencia relacionada

[Predicado DISTINCT \(Db2 SQL\)](#)

[Predicado NULL \(Db2 SQL\)](#)

Selección de columnas derivadas

En una sentencia SELECT, puede seleccionar columnas que no son columnas reales de una tabla. En su lugar, puede especificar "columnas" que se derivan de una constante, una expresión o una función.

Ejemplo: SELECT con una expresión

Esta instrucción SQL genera una tabla de resultados en la que la segunda columna es una columna derivada que se genera al sumar los valores de las columnas SALARY, BONUS y COMM.

```
SELECT EMPNO, (SALARY + BONUS + COMM)  
  FROM DSN8C10.EMP;
```

Las columnas derivadas en una tabla de resultados, como (SALARIO + BONIFICACIÓN + COM), no tienen nombre. Puede utilizar la cláusula AS para dar un nombre a una columna sin nombre de la tabla de resultados. Para obtener información sobre el uso de la cláusula AS, consulte ["Asignación de nombre a las columnas de resultados"](#) en la página 382.

Qué hacer a continuación

Para ordenar las filas de una tabla de resultados por los valores de una columna derivada, especifique un nombre para la columna utilizando la cláusula AS, y especifique ese nombre en la cláusula ORDER BY. Para obtener información sobre el uso de la cláusula ORDER BY, consulte “[Ordenación de las filas de la tabla de resultados](#)” en la página 384.

Selección de datos XML

Puede seleccionar todos los datos XML que estén almacenados en una columna determinada o sólo un subconjunto de datos de una columna XML.

Procedimiento

- Puede seleccionar todos los datos XML almacenados en una columna concreta especificando SELECT *nombre de columna* o SELECT *, tal como haría con columnas de cualquier otro tipo de datos.
- De forma alternativa, puede seleccionar sólo un subconjunto de datos de una columna XML utilizando una expresión XPath en una instrucción SELECT. Las expresiones XPath identifican nodos específicos en un documento XML.

Para seleccionar un subconjunto de datos en una columna XML, especifique la función XMLQUERY en su instrucción SELECT con los siguientes parámetros:

- Expresión XPath incrustada en una constante de cadena de caracteres. Especifique una expresión XPath que identifique qué datos XML se deben devolver.
- Cualquier valor adicional que se deba pasar a la expresión XPath, incluido el nombre de la columna XML. Especifique estos valores después de la palabra clave PASSING.

Ejemplo

Supongamos que almacena órdenes de compra como documentos XML en la columna POrder de la tabla PurchaseOrders. Debe encontrar en cada orden de compra los artículos cuyo nombre de producto sea igual a un nombre de la tabla de productos. Puede utilizar la siguiente declaración para encontrar estos valores:

```
SELECT XMLQUERY('//*[productName = $n]'  
    PASSING PO.POrder,  
    P.name AS "n")  
FROM PurchaseOrders PO, Product P;
```

Esta sentencia devuelve los elementos de artículo de la columna POrder que satisfacen los criterios de la expresión XPath.

Conceptos relacionados

[Visión general de XQuery \(Programación de Db2 para XML\)](#)

Referencia relacionada

[XMLQUERY función escalar \(Db2 SQL\)](#)

Formato de la tabla de resultados

Una sentencia SQL devuelve datos en una tabla denominada tabla de resultados. Puede especificar determinados atributos de la tabla de resultados como, por ejemplo, los nombres de columna, el orden de las filas y si las filas están numeradas.

Tablas de resultados

Los datos que se recuperan mediante una instrucción SQL siempre están en forma de tabla, que se denomina *tabla de resultados*. Como las tablas de las que se recuperan datos, una tabla de resultados tiene filas y columnas. Un programa obtiene estos datos una fila cada vez.

Ejemplo de tabla de resultados : Supongamos que emite la siguiente instrucción SELECT, que recupera el apellido, el nombre y el número de teléfono de los empleados del departamento de D11, de la tabla de empleados de muestra:

```
SELECT LASTNAME, FIRSTNME, PHONENO  
  FROM DSN8C10.EMP  
 WHERE WORKDEPT = 'D11'  
 ORDER BY LASTNAME;
```

La tabla de resultados tiene un aspecto similar al siguiente:

LASTNAME	FIRSTNME	PHONENO
ADAMSON	BRUCE	4510
BROWN	DAVID	4501
JOHN	REBA	0672
JONES	WILLIAM	0942
LUTZ	JENNIFER	0672
PIANKA	ELIZABETH	3782
SCOUTTEN	MARILYN	1682
STERN	IRVING	6432
WALKER	JAMES	2986
YAMAMOTO	KIYOSHI	2890
YOSHIMURA	MASATOSHI	2890

Exclusión de filas duplicadas de la tabla de resultados de una consulta

Puede solicitar a Db2 que excluya varias filas idénticas de una tabla de resultados de consulta. Por ejemplo, una consulta puede devolver varias filas para cada empleado cuando una fila por empleado es suficiente para su programa.

Procedimiento

Especifique la palabra clave DISTINCT en la consulta.

La palabra clave DISTINCT excluye las filas duplicadas de la tabla de resultados de la consulta, de modo que cada fila contiene datos únicos.

Restricción: No puede utilizar la palabra clave DISTINCT con columnas LOB o columnas XML.

Ejemplo

La siguiente instrucción SELECT enumera números de departamento únicos para departamentos administrativos:

```
SELECT DISTINCT ADMRDEPT  
  FROM DSN8C10.DEPT;
```

La tabla de resultados tiene un aspecto similar al siguiente:

ADMRDEPT
=====
A00
D01
E01

Tareas relacionadas

[Codificación de sentencias SQL para evitar procesos innecesarios \(Db2 Performance\)](#)

Referencia relacionada

[cláusula-select \(Db2 SQL\)](#)

Asignación de nombre a las columnas de resultados

Puede proporcionar sus propios nombres para las columnas de tabla de resultados para una sentencia SELECT. Esta capacidad es particularmente útil para una columna que se deriva de una expresión o función.

Procedimiento

Utilice la cláusula AS para nombrar columnas de resultados en una instrucción SELECT.

ejemplos

Los siguientes ejemplos muestran diferentes formas de utilizar la cláusula AS.

Ejemplo: SELECT con AS CLAUSE

El siguiente ejemplo de la sentencia SELECT da a la expresión SALARIO+PRIMA+COM el nombre TOTAL_SAL.

```
SELECT SALARY+BONUS+COMM AS TOTAL_SAL  
      FROM DSN8C10.EMP  
      ORDER BY TOTAL_SAL;
```

Ejemplo: CREAR VISTA con cláusula AS

Puede especificar nombres de columnas de resultados en la cláusula select de una sentencia CREATE VIEW. No es necesario que proporcione la lista de columnas de CREATE VIEW, porque la palabra clave AS nombra la columna derivada. Las columnas en la vista EMP_SAL son EMPNO y TOTAL_SAL.

```
CREATE VIEW EMP_SAL AS  
      SELECT EMPNO, SALARY+BONUS+COMM AS TOTAL_SAL  
            FROM DSN8C10.EMP;
```

Ejemplo: operador de conjunto con cláusula AS

Puede utilizar la cláusula AS con operadores de conjuntos, como UNION. En este ejemplo, la cláusula AS se utiliza para dar el mismo nombre a las columnas correspondientes de las tablas en una UNIÓN. La tercera columna de resultados de la unión de las dos tablas tiene el nombre TOTAL_VALUE, aunque contiene datos que se derivan de columnas con nombres diferentes:

```
SELECT 'On hand' AS STATUS, PARTNO, QOH * COST AS TOTAL_VALUE  
      FROM PART_ON_HAND  
UNION ALL  
SELECT 'Ordered' AS STATUS, PARTNO, QORDER * COST AS TOTAL_VALUE  
      FROM ORDER_PART  
      ORDER BY PARTNO, TOTAL_VALUE;
```

La columna STATUS y la columna derivada TOTAL_VALUE tienen el mismo nombre en la primera y la segunda tabla de resultados. Se combinan en la unión de las dos tablas de resultados, que es similar a la siguiente salida parcial:

STATUS	PARTNO	TOTAL_VALUE
=====	=====	=====
On hand	00557	345.60
Ordered	00557	150.50
.		
.		
.		

Ejemplo: columna derivada GROUP BY

Puede utilizar la cláusula AS en una cláusula FROM para asignar un nombre a una columna derivada a la que deseé hacer referencia en una cláusula GROUP BY. Esta instrucción SQL nombra HIREYEAR en la expresión de tabla anidada, lo que le permite utilizar el nombre de esa columna de resultados en la cláusula GROUP BY:

```
SELECT HIREYEAR, AVG(SALARY)  
      FROM (SELECT YEAR(HIREDATE) AS HIREYEAR, SALARY  
            FROM DSN8C10.EMP) AS NEWEMP  
      GROUP BY HIREYEAR;
```

No puede utilizar GROUP BY con un nombre que se defina con una cláusula AS para la columna derivada YEAR(HIREDATE) en el SELECT externo, porque ese nombre no existe cuando se ejecuta GROUP BY. Sin embargo, puede utilizar GROUP BY con un nombre que se defina con una cláusula AS en la expresión de tabla anidada, porque la expresión de tabla anidada se ejecuta antes del GROUP BY que hace referencia al nombre.

Tareas relacionadas

[Combinación de tablas de resultados desde varias sentencias SELECT](#)

Al combinar los resultados de varias sentencias SELECT, puede elegir qué incluir en la tabla de resultados. Puede incluir todas las filas, solo las filas que están en la tabla de resultados de ambas sentencias SELECT, o solo filas que son exclusivas para la tabla de resultados de la primera sentencia SELECT.

[Definición de una vista](#)

Una vista es una especificación con nombre de una tabla de resultados. Utilice las vistas para controlar qué usuarios tienen acceso a determinados datos o para simplificar la escritura de sentencias SQL.

[Resumen de valores de grupo](#)

Puede agrupar filas en la tabla de resultados por valores de una o más columnas o por los resultados de una expresión. A continuación, puede aplicar funciones de totales a cada grupo.

[Referencia relacionada](#)

[cláusula-select \(Db2 SQL\)](#)

Ordenación de las filas de la tabla de resultados

Si desea asegurarse de que las filas en la tabla de resultados se ordenen de un modo particular, deberá especificar el orden en la sentencia SELECT. De lo contrario, Db2 puede devolver las filas en cualquier orden.

Acerca de esta tarea

Usar ORDER BY es la única manera de garantizar que sus filas se ordenen como usted quiere.

Procedimiento

Para recuperar filas en un orden específico, utilice la cláusula ORDER BY

ejemplos

Ejemplo: Especificación de la clave de ordenación en la cláusula ORDER BY

El orden de las filas seleccionadas depende de las claves de ordenación que identifique en la cláusula ORDER BY. Una clave de ordenación puede ser un nombre de columna, un número entero que representa el número de una columna en la tabla de resultados o una expresión. Db2 ordena las filas por la primera clave de ordenación, seguida de la segunda clave de ordenación, y así sucesivamente.

Puede listar las filas en orden ascendente o descendente. Los valores nulos aparecen en último lugar en una clasificación ascendente y en primer lugar en una clasificación descendente.

Db2 ordena las cadenas en la secuencia de compaginación asociada al esquema de codificación de la tabla. Db2 clasifica números de forma algebraica y clasifica los valores de fecha y hora cronológicamente.

Restricción: No puede utilizar la cláusula ORDER BY con columnas LOB o XML.

Ejemplo: cláusula ORDER BY con un nombre de columna como clave de ordenación

Recuperar los números de empleado, apellidos y fechas de contratación de los empleados del departamento A00 en orden ascendente de fechas de contratación:

```
SELECT EMPNO, LASTNAME, HIREDATE
      FROM DSN8C10.EMP
     WHERE WORKDEPT = 'A00'
    ORDER BY HIREDATE ASC;
```

La tabla de resultados tiene un aspecto similar al siguiente:

EMPNO	LASTNAME	HIREDATE
000110	LUCCHESI	1958-05-16
000120	O'CONNELL	1963-12-05
000010	HAAS	1965-01-01

200010	HEMMINGER	1965-01-01
200120	ORLANDO	1972-05-05

Ejemplo: cláusula ORDER BY con una expresión como clave de ordenación

La siguiente subsecuencia recupera los números de empleado, salarios, comisiones y compensación total (salario más comisión) de los empleados con una compensación total superior a 40 000. Ordene los resultados por compensación total:

```
SELECT EMPNO, SALARY, COMM, SALARY+COMM AS "TOTAL COMP"
  FROM DSN8C10.EMP
 WHERE SALARY+COMM > 40000
 ORDER BY SALARY+COMM;
```

La tabla de resultados intermedios tiene un aspecto similar al siguiente resultado:

EMPNO	SALARY	COMM	TOTAL COMP
000030	38250.00	3060.00	41310.00
000050	40175.00	3214.00	43389.00
000020	41250.00	3300.00	44550.00
000110	46500.00	3720.00	50220.00
200010	46500.00	4220.00	50720.00
000010	52750.00	4220.00	56970.00

Referenciar columnas derivadas en la cláusula ORDER BY

Si utiliza la cláusula AS para nombrar una columna sin nombre en una instrucción SELECT, puede utilizar ese nombre en la cláusula ORDER BY. La siguiente instrucción SQL ordena la información seleccionada por salario total:

```
SELECT EMPNO, (SALARY + BONUS + COMM) AS TOTAL_SAL
  FROM DSN8C10.EMP
 ORDER BY TOTAL_SAL;
```

Referencia relacionada

[fullselect \(Db2 SQL\)](#)

Numeración de las filas en una tabla de resultados

Db2 no numera las filas de la tabla de resultados de una consulta a menos que usted solicite explícitamente que se numeren las filas.

Acerca de esta tarea

Para numerar las filas de una tabla de resultados, incluya la especificación ROW_NUMBER en su consulta. Si desea asegurarse de que las filas están en un orden determinado, incluya una cláusula ORDER BY después de la palabra clave OVER. De lo contrario, las filas se numeran en un orden arbitrario.

Ejemplo

Supongamos que desea una lista de empleados y salarios del departamento de recursos humanos (D11) en la tabla EMP de muestra. Puede devolver una lista numerada ordenada por apellido enviando la siguiente consulta:

```
SELECT ROW_NUMBER() OVER (ORDER BY LASTNAME) AS NUMBER,
       WORKDEPT, LASTNAME, SALARY
  FROM DSN8910.EMP
 WHERE WORKDEPT='D11'
```

Esta consulta devuelve el resultado siguiente:

NUMBER	WORKDEPT	LASTNAME	SALARY
1	D11	ADAMSON	25280.00
2	D11	BROWN	27740.00
3	D11	JOHN	29840.00
4	D11	JONES	18270.00
5	D11	LUTZ	29840.00

6	D11	PIANKA	22250 .00
7	D11	SCOUTTEN	21340 .00
8	D11	STERN	32250 .00
9	D11	WALKER	20450 .00
10	D11	YAMAMOTO	24680 .00
11	D11	YOSHIMURA	24680 .00

Referencia relacionada

Especificaciones OLAP (Db2 SQL)

Clasificación de las filas

Puede solicitar que Db2 calcule el rango ordinal de cada fila del conjunto de resultados en función de una columna concreta. Por ejemplo, puede clasificar los tiempos de finalización de un maratón para determinar los ganadores del primer, segundo y tercer puesto.

Procedimiento

Para clasificar filas, utilice una de las siguientes especificaciones de clasificación en una instrucción SQL:

- Utilice RANK para devolver un número de rango para cada valor de fila.

Utilice esta especificación si desea que se omitan los números de rango cuando existan valores de fila duplicados.

Por ejemplo, supongamos que los cinco primeros clasificados en un maratón tienen los siguientes tiempos:

- 2:31:57
- 2:34:52
- 2:34:52
- 2:37:26
- 2:38:01

Cuando se utiliza la especificación RANK, Db2 devuelve los siguientes números de rango:

Tabla 63. Ejemplo de valores devueltos cuando se especifica RANK

Valor	Número de rango
2:31:57	1
2:34:52	2
2:34:52	2
2:37:26	4
2:38:01	5

- Utilice DENSE_RANK para devolver un número de rango para cada valor de fila.

Utilice esta especificación si no desea que se omitan los números de rango cuando existan valores de fila duplicados.

Por ejemplo, cuando especifica DENSE_RANK con los mismos tiempos que se enumeran en la descripción de RANK, Db2 devuelve los siguientes números de rango:

Tabla 64. Ejemplo de valores devueltos cuando se especifica RANK

Valor	Número de rango
2:31:57	1
2:34:52	2
2:34:52	2
2:37:26	3

Tabla 64. Ejemplo de valores devueltos cuando se especifica RANK (continuación)

Valor	Número de rango
2:38:01	4

ejemplos

Supongamos que tiene los siguientes valores en la columna DATA de la tabla T1:

```
DATA
-----
100
35
23
8
8
6
```

Ejemplo: RANK

Supongamos que utiliza la siguiente especificación RANK:

```
SELECT DATA,
       RANK() OVER (ORDER BY DATA DESC) AS RANK_DATA
    FROM T1
   ORDER BY RANK_DATA;
```

Db2 devuelve los siguientes datos clasificados:

```
DATA      RANK_DATA
-----
100          1
35           2
23           3
8            4
8            4
6            6
```

Ejemplo: DENSE RANK

Supongamos que utiliza la siguiente especificación DENSE_RANK en los mismos datos:

```
SELECT DATA,
       DENSE_RANK() OVER (ORDER BY DATA DESC) AS RANK_DATA
    FROM T1
   ORDER BY RANK_DATA;
```

Db2 devuelve los siguientes datos clasificados:

```
DATA      RANK_DATA
-----
100          1
36           2
23           3
8            4
8            4
6            5
```

En el ejemplo con la especificación RANK, dos valores iguales se clasifican como 4. El siguiente número de rango es 6. Se omite el número 5.

En el ejemplo con la opción DENSE_RANK, esos dos valores iguales también se clasifican como 4. Sin embargo, el siguiente número de rango es 5. Con DENSE_RANK, no existen huecos en la numeración secuencial de rango.

Referencia relacionada

[Especificaciones OLAP \(Db2 SQL\)](#)

Acceso a parte de un conjunto de resultados en base a la posición de datos

La paginación basada en números o dependiente de datos se puede utilizar para recuperar un subconjunto de datos de un conjunto de resultados en base a la posición de los datos.

Acerca de esta tarea

Para recuperar un subconjunto de datos de un conjunto de resultados basado en la posición de los datos en el conjunto de resultados, puede utilizar la paginación dependiente de los datos o la paginación basada en números.

Procedimiento

- Para la *paginación dependiente de datos* : Utilice *expresiones de valor de fila* con los operadores de comparación <, <=, > o >= en una instrucción SELECT para recuperar solo una parte de un conjunto de resultados.

Cuando se utilizan con un predicado básico, las expresiones de valor de fila permiten que una aplicación acceda solo a una parte de una tabla de resultados de una consulta de selección (Db2) basada en un valor de clave lógica.

La siguiente instrucción SELECT devuelve información de la tabla en la que el valor de la columna "LASTNAME" es mayor o igual que "SMITH" y el valor de la columna "FIRSTNAME" es mayor que "JOHN":

```
SELECT EMPNO, LASTNAME, HIREDATE  
  FROM DSN8C10.EMP  
 WHERE (LASTNAME, FIRSTNAME) >= ('SMITH', 'JOHN')  
 ORDER BY HIREDATE ASC;
```

- Para la *paginación numérica* : Utilice la cláusula OFFSET (ya sea sola o con la cláusula FETCH) para omitir un número determinado de filas del conjunto de resultados.

Para acceder a una parte de un conjunto de resultados de un Db2, basado en una posición absoluta, la cláusula OFFSET puede especificarse como parte de la instrucción SELECT. La cláusula OFFSET especifica el número de filas que se deben omitir desde el principio de un conjunto de resultados, lo que puede ser una forma más eficiente de filtrar las filas innecesarias. La cláusula OFFSET puede utilizarse con la cláusula FETCH para limitar aún más el número de filas devueltas por el conjunto de resultados.

La siguiente instrucción SELECT omite las primeras 100 filas de la tabla "T1" antes de devolver las filas de la consulta:

```
SELECT * FROM T1  
OFFSET 100 ROWS;
```

El uso de la cláusula OFFSET con la cláusula FETCH especifica el número de filas que se deben omitir desde el principio de la tabla antes de devolver el número de filas especificado en la cláusula FETCH:

```
SELECT * FROM T1  
OFFSET 10 ROWS  
FETCH FIRST 10 ROWS ONLY;
```

Para devolver tres "páginas" de 10 filas cada una, puede utilizar sentencias similares a las siguientes sentencias SQL:

```
SELECT * FROM T1  
OFFSET 0 ROWS  
FETCH FIRST 10 ROWS ONLY;  
  
SELECT * FROM T1  
OFFSET 10 ROWS  
FETCH NEXT 10 ROWS ONLY;  
  
SELECT * FROM T1  
OFFSET 20 ROWS  
FETCH NEXT 10 ROWS ONLY;
```

Este ejemplo consta de tres sentencias SQL independientes, cada una con valores diferentes para la cláusula OFFSET. Cada sentencia SELECT se procesa como una nueva sentencia SQL.

Conceptos relacionados

[Soporte de paginación de SQL \(Novedades de Db2 para z/OS\)](#)

Referencia relacionada

[cláusula-offset \(de compensación\) \(Db2 SQL\)](#)

[Predicado básico \(Db2 SQL\)](#)

[cláusula-fetch \(Db2 SQL\)](#)

Combinación de tablas de resultados desde varias sentencias SELECT

Al combinar los resultados de varias sentencias SELECT, puede elegir qué incluir en la tabla de resultados. Puede incluir todas las filas, solo las filas que están en la tabla de resultados de ambas sentencias SELECT, o solo filas que son exclusivas para la tabla de resultados de la primera sentencia SELECT.

Acerca de esta tarea

Supongamos que desea combinar los resultados de dos sentencias SELECT que devuelven las siguientes tablas de resultados:

Ejemplo: Tabla de resultados de R1

COL1	COL2
a	a
a	b
a	c

Ejemplo: Tabla de resultados de R2

COL1	COL2
a	b
a	c
a	d

Puede utilizar *los operadores de conjunto* para combinar dos o más sentencias SELECT y formar una única tabla de resultados:

UNION

UNION devuelve todos los valores de la tabla de resultados de cada sentencia SELECT. Si desea que todas las filas duplicadas se repitan en la tabla de resultados, especifique UNION ALL. Si desea que se eliminen las filas duplicadas redundantes de la tabla de resultados, especifique UNION o UNION DISTINCT.

Por ejemplo, el siguiente ejemplo es el resultado de especificar UNION para R1 y R2.

COL1	COL2
a	a
a	b
a	c
a	d

EXCEPT

Devuelve todas las filas de la primera tabla de resultados (R1) que no están también en la segunda tabla de resultados (R2). Si desea que todas las filas duplicadas de R1 estén incluidas en la tabla de resultados, especifique EXCEPT ALL. Si desea que las filas duplicadas redundantes en R1 se eliminen de la tabla de resultados, especifique EXCEPT o EXCEPT DISTINCT.

El resultado de la operación EXCEPT depende de la sentencia SELECT que se incluya antes de la palabra clave EXCEPT en la sentencia SQL. Por ejemplo, si la sentencia SELECT que devuelve la tabla de resultados R1 aparece en primer lugar, el resultado es una sola fila:

COL1	COL2
a	a

Si la sentencia SELECT que devuelve la tabla de resultados R2 aparece en primer lugar, el resultado final es una fila diferente:

COL1	COL2
a	d

INTERSECT

Devuelve las filas que están en la tabla de resultados de ambas sentencias SELECT. Si desea que todas las filas duplicadas estén contenidas en la tabla de resultados, especifique INTERSECT ALL. Si desea que se eliminen las filas duplicadas redundantes de la tabla de resultados, especifique INTERSECT o INTERSECT DISTINCT.

Por ejemplo, el siguiente ejemplo es el resultado de especificar UNION para R1 y R2.

COL1	COL2
a	b
a	c

Cuando especifica uno de los operadores establecidos, Db2 procesa cada instrucción SELECT para formar una tabla de resultados provisional y, a continuación, combina la tabla de resultados provisional de cada instrucción. Si *la enésima columna* de la primera tabla de resultados (R1) y *la enésima columna* de la segunda tabla de resultados (R2) tienen el mismo nombre de columna de resultados, *la enésima columna* de la tabla de resultados tendrá ese mismo nombre de columna de resultados. Si *la enésima columna* de R1 y la *enésima columna* de R2 no tienen los mismos nombres, la columna de resultados no tiene nombre.

Procedimiento

- Para combinar dos o más sentencias SELECT para formar una sola tabla de resultados, utilice los operadores de conjuntos: UNION, EXCEPT o INTERSECT.

Por ejemplo, supongamos que tiene las siguientes tablas para gestionar el stock en dos librerías.

Tabla 65. STOCKA

ISBN	TÍTULO	AUTOR	PREMIO NOBEL
8778997709	Por quién doblan las campanas	Hemingway	N
4599877699	La buena tierra	Dólar	Y
9228736278	Una historia de dos ciudades	Dickens	N
1002387872	Amado	Morrison	Y
4599877699	La buena tierra	Dólar	Y
0087873532	El laberinto de la soledad	Paz	Y

Tabla 66. STOCKB

ISBN	TÍTULO	AUTOR	PREMIO NOBEL
6689038367	Las uvas de la ira	Steinbeck	Y
2909788445	El grito silencioso	OE	Y
1182983745	Luz en agosto	Faulkner	Y
9228736278	Una historia de dos ciudades	Dickens	N
1002387872	Amado	Morrison	Y

Ejemplo: cláusula UNIÓN

Supongamos que quiere una lista de libros cuyos autores hayan ganado el Premio Nobel y que estén en stock en cualquiera de las dos tiendas. La siguiente instrucción SQL devuelve estos libros ordenados por nombre de autor sin filas duplicadas redundantes:

```
SELECT TITLE, AUTHOR  
  FROM STOCKA  
 WHERE NOBELPRIZE = 'Y'  
UNION  
SELECT TITLE, AUTHOR  
  FROM STOCKB  
 WHERE NOBELPRIZE = 'Y'  
 ORDER BY AUTHOR
```

Esta declaración devuelve la siguiente tabla de resultados finales:

Tabla 67. Resultado de la UNIÓN

TÍTULO	AUTOR
La buena tierra	Dólar
Luz en agosto	Faulkner
Amado	Morrison
El grito silencioso	OE
El laberinto de la soledad	Paz
Las uvas de la ira	Steinbeck

Ejemplo: cláusula EXCEPT

Supongamos que desea una lista de libros que solo están en STOCKA. La siguiente instrucción SQL devuelve los nombres de los libros que están en STOCKA solo sin filas duplicadas redundantes:

```
SELECT TITLE  
  FROM STOCKA  
EXCEPT  
SELECT TITLE  
  FROM STOCKB  
 ORDER BY TITLE;
```

Esta declaración devuelve la siguiente tabla de resultados:

Tabla 68. Resultado de EXCEPTO

TÍTULO
Por quién doblan las campanas
La buena tierra
El laberinto de la soledad

Ejemplo: cláusula INTERSECT

Supongamos que desea una lista de libros que están tanto en STOCKA como en STOCKB. La siguiente sentencia devuelve una lista de todos los libros de ambas tablas con las filas duplicadas redundantes eliminadas.

```
SELECT TITLE  
  FROM STOCKA  
INTERSECT  
SELECT TITLE  
  FROM STOCKB  
 ORDER BY TITLE;
```

Esta declaración devuelve la siguiente tabla de resultados:

Tabla 69. Resultado de INTERSECT

TÍTULO

Una historia de dos ciudades

Amado

- Para mantener todas las filas duplicadas al combinar tablas de resultados, especifique la palabra clave ALL con la cláusula del operador de conjuntos.

Los siguientes ejemplos utilizan las tablas STOCKA y STOCK B del paso anterior.

Ejemplo: UNION ALL

La siguiente instrucción SQL devuelve una lista de libros que han ganado premios Nobel y están en stock en cualquiera de las tiendas, incluidos los duplicados.

```
SELECT TITLE, AUTHOR
  FROM STOCKA
 WHERE NOBELPRIZE = 'Y'
UNION ALL
SELECT TITLE, AUTHOR
  FROM STOCKB
 WHERE NOBELPRIZE = 'Y'
 ORDER BY AUTHOR
```

Esta declaración devuelve la siguiente tabla de resultados:

Tabla 70. Resultado de UNION ALL

TÍTULO	AUTOR
La buena tierra	Dólar
La buena tierra	Dólar
Luz en agosto	Faulkner
Amado	Morrison
Amado	Morrison
El grito silencioso	OE
El laberinto de la soledad	Paz
Las uvas de la ira	Steinbeck

Ejemplo: EXCEPT TODO

Supongamos que desea una lista de libros que solo están en STOCKA. La siguiente instrucción SQL devuelve los nombres de libros que están en STOCKA solo con todas las filas duplicadas:

```
SELECT TITLE
  FROM STOCKA
EXCEPT ALL
SELECT TITLE
  FROM STOCKB
 ORDER BY TITLE;
```

Esta declaración devuelve la siguiente tabla de resultados:

Tabla 71. Resultado de EXCEPT ALL

TÍTULO

Por quién doblan las campanas

La buena tierra

La buena tierra

Tabla 71. Resultado de EXCEPT ALL (continuación)

TÍTULO

El laberinto de la soledad

Ejemplo: INTERSECCIÓN TODO

Supongamos que desea una lista de libros que están tanto en STOCKA como en STOCKB, incluyendo cualquier coincidencia duplicada. La siguiente declaración devuelve una lista de títulos que están en ambos stocks, incluyendo coincidencias duplicadas. En este caso, existe una coincidencia para "Historia de dos ciudades" y una coincidencia para "Beloved"

```
SELECT TITLE  
      FROM STOCKA  
INTERSECT ALL  
SELECT TITLE  
      FROM STOCKB  
      ORDER BY TITLE;
```

Esta declaración devuelve la siguiente tabla de resultados:

Tabla 72. Resultado de INTERSECT ALL

TÍTULO

Una historia de dos ciudades

Amado

- Para eliminar filas duplicadas redundantes al combinar tablas de resultados, especifique una de las siguientes palabras clave:
 - UNIÓN o UNIÓN DISTINTA
 - EXCEPTO o EXCEPTO DISTINTO
 - INTERSECCIÓN o INTERSECCIÓN DISTINTA
- Para ordenar toda la tabla de resultados, especifique la cláusula ORDER BY al final.

Tareas relacionadas

[Ordenación de las filas de la tabla de resultados](#)

Si desea asegurarse de que las filas en la tabla de resultados se ordenen de un modo particular, deberá especificar el orden en la sentencia SELECT. De lo contrario, Db2 puede devolver las filas en cualquier orden.

Referencia relacionada

[fullselect \(Db2 SQL\)](#)

Resumen de valores de grupo

Puede agrupar filas en la tabla de resultados por valores de una o más columnas o por los resultados de una expresión. A continuación, puede aplicar funciones de totales a cada grupo.

Procedimiento

Utiliza la cláusula GROUP BY.

Cuando se utiliza, la cláusula GROUP BY sigue a la cláusula FROM y a cualquier cláusula WHERE, y precede a la cláusula ORDER BY.

Excepto las columnas indicadas en la cláusula GROUP BY, la sentencia SELECT debe especificar las columnas seleccionadas como un operando de una de las funciones de totales.

Si una columna que especifique en la cláusula GROUP BY contiene valores nulos, Db2 considera que esos valores nulos son iguales. Por lo tanto, todos los nulos forman un solo grupo.

ejemplos

Ejemplo: cláusula GROUP BY utilizando una columna

La siguiente declaración SQL enumera, para cada departamento, el nivel educativo más bajo y más alto dentro de ese departamento:

```
SELECT WORKDEPT, MIN(EDLEVEL), MAX(EDLEVEL)
  FROM DSN8C10.EMP
 GROUP BY WORKDEPT;
```

Ejemplo: cláusula GROUP BY que utiliza más de una columna

Puede agrupar las filas por los valores de más de una columna. Por ejemplo, la siguiente declaración encuentra el salario medio de hombres y mujeres en los departamentos de A00 y C01:

```
SELECT WORKDEPT, SEX, AVG(SALARY) AS AVG_SALARY
  FROM DSN8C10.EMP
 WHERE WORKDEPT IN ('A00', 'C01')
 GROUP BY WORKDEPT, SEX;
```

La tabla de resultados tiene un aspecto similar al siguiente:

WORKDEPT	SEX	AVG_SALARY
A00	F	49625.00000000
A00	M	35000.00000000
C01	F	29722.50000000

Db2 agrupa las filas primero por número de departamento y luego (dentro de cada departamento) por sexo antes de obtener el valor medio de SALARIO para cada grupo.

Ejemplo: cláusula GROUP BY que utiliza una expresión

También puede agrupar las filas según los resultados de una expresión. Por ejemplo, la siguiente declaración agrupa los departamentos por sus personajes principales y enumera el nivel de educación más bajo y más alto para cada grupo:

```
SELECT SUBSTR(WORKDEPT,1,1), MIN(EDLEVEL), MAX(EDLEVEL)
  FROM DSN8C10.EMP
 GROUP BY SUBSTR(WORKDEPT,1,1);
```

Referencia relacionada

[cláusula-group-by \(Db2 SQL\)](#)

Grupos de filtrado

Si agrupa filas en la tabla de resultados, también puede especificar una condición de búsqueda que cada grupo recuperado debe satisfacer. La condición de búsqueda comprueba las propiedades de cada grupo en lugar de las propiedades de las filas individuales del grupo.

Procedimiento

Utilice la cláusula HAVING para especificar una condición de búsqueda.

La cláusula HAVING actúa como una cláusula WHERE para grupos y contiene el mismo tipo de condiciones de búsqueda que se especifican en una cláusula WHERE.

Ejemplo

Ejemplo: cláusula HAVING

La siguiente instrucción SQL incluye una cláusula HAVING que especifica una condición de búsqueda para grupos de departamentos de trabajo en la tabla de empleados:

```
SELECT WORKDEPT, AVG(SALARY) AS AVG_SALARY
  FROM DSN8C10.EMP
 GROUP BY WORKDEPT
 HAVING COUNT(*) > 1
 ORDER BY WORKDEPT;
```

La tabla de resultados tiene un aspecto similar al siguiente:

WORKDEPT	AVG_SALARY
A00	40850.0000000
C01	29722.5000000
D11	25147.27272727
D21	25668.57142857
E11	21020.0000000
E21	24086.6666666

Compare el ejemplo anterior con el segundo ejemplo que se muestra en “Resumen de valores de grupo” en la página 393. La cláusula HAVING COUNT(*) > 1 garantiza que solo se muestren los departamentos con más de un miembro. En este caso, los departamentos B01 y E01 no se muestran porque la cláusula HAVING comprueba una propiedad del grupo.

Ejemplo: cláusula HAVING utilizada con una cláusula GROUP BY

Utilice la cláusula HAVING para recuperar el salario medio y el nivel mínimo de educación de las mujeres en cada departamento para el que todas las empleadas tengan un nivel de educación superior o igual a 16. Suponiendo que solo desea resultados de los departamentos A00 y D11, la siguiente instrucción SQL comprueba la propiedad de grupo, MIN(EDLEVEL):

```
SELECT WORKDEPT, AVG(SALARY) AS AVG_SALARY,
       MIN(EDLEVEL) AS MIN_EDLEVEL
  FROM DSN8C10.EMP
 WHERE SEX = 'F' AND WORKDEPT IN ('A00', 'D11')
 GROUP BY WORKDEPT
 HAVING MIN(EDLEVEL) >= 16;
```

La tabla de resultados tiene un aspecto similar al siguiente:

WORKDEPT	AVG_SALARY	MIN_EDLEVEL
A00	49625.0000000	18
D11	25817.5000000	17

Cuando especifique tanto GROUP BY como HAVING, la cláusula HAVING debe seguir a la cláusula GROUP BY. Una función en una cláusula HAVING puede incluir DISTINCT si no ha utilizado DISTINCT en ningún otro lugar de la misma instrucción SELECT. También puede conectar varios predicados en una cláusula HAVING con AND u OR, y puede usar NOT para cualquier predicado de una condición de búsqueda.

Referencia relacionada

[cláusula-where \(Db2 SQL\)](#)

[cláusula-having \(Db2 SQL\)](#)

Buscar filas que se modificaron dentro de un período de tiempo específico

Puede filtrar las filas en función de la hora en que se actualizaron. Por ejemplo, es posible que desee encontrar todas las filas de una tabla concreta que se hayan modificado en los últimos 7 días.

Procedimiento

Especifique la expresión ROW CHANGE TIMESTAMP en el predicado de su sentencia SQL.

Recomendación: Asegúrese de que la tabla tiene una columna ROW CHANGE TIMESTAMP que se definió antes del período de tiempo que desea consultar. Esta columna garantiza que Db2 devuelva solo las filas que se actualizaron en el período de tiempo dado.

Si la tabla no tiene una columna ROW CHANGE TIMESTAMP, Db2 devuelve todas las filas de cada página que hayan sufrido algún cambio dentro del período de tiempo dado. En este caso, su conjunto de resultados puede contener filas que no se han actualizado en el período de tiempo dado, si se han actualizado o insertado otras filas en esa página.

ejemplos

Ejemplo

Supongamos que la tabla TAB tiene una columna ROW CHANGE TIMESTAMP y que desea devolver todos los registros que han cambiado en los últimos 30 días. La siguiente consulta devuelve todas esas filas.

```
SELECT * FROM TAB  
WHERE ROW CHANGE TIMESTAMP FOR TAB <= CURRENT TIMESTAMP AND  
ROW CHANGE TIMESTAMP FOR TAB >= CURRENT TIMESTAMP - 30 days;
```

Ejemplo

Supongamos que desea devolver todos los registros que han cambiado desde las 9:00 a. m. del 1 de enero de 2004. La siguiente consulta devuelve todas esas filas.

```
SELECT * FROM TAB  
WHERE ROW CHANGE TIMESTAMP FOR TAB >= '2004-01-01-09.00.00';
```

Referencia relacionada

- [Expresión ROW CHANGE \(Db2 SQL\)](#)
- [CREATE TABLE declaración \(Db2 SQL\)](#)
- [cláusula-where \(Db2 SQL\)](#)

Unión de datos desde más de una tabla

Algunas veces la información que desea ver no está en una sola tabla. Para formar una fila de la tabla de resultados es posible que desee recuperar algunos valores de columnas de una tabla y algunos valores de columna de otra tabla.

Acerca de esta tarea

Puede utilizar una instrucción SELECT para recuperar y unir valores de columna de dos o más tablas en una sola fila.

Una operación de unión normalmente empareja una fila de una tabla con una fila de otra en base a una condición de unión. Db2 admite los siguientes tipos de uniones: unión interna, unión externa izquierda, unión externa derecha y unión externa completa. Se pueden especificar uniones en la cláusula FROM de una consulta.

Db2 admite uniones internas, uniones externas, que incluyen uniones externas izquierdas, uniones externas derechas, uniones externas completas y uniones cruzadas.

Unión interna

Un resultado de *unión interna* es el producto cruzado de las tablas, pero solo conserva las filas en las que la condición de unión es verdadera. El resultado de un T1 INNER JOIN T2 o consiste en sus filas emparejadas. Si no se especifica un operador de unión, INNER es el predeterminado. El orden en el que se realiza una UNIÓN EXTERIOR IZQUIERDA o una UNIÓN EXTERIOR DERECHA puede afectar al resultado. Para más información, consulte [“Uniones internas” en la página 401](#).

Unión externa

Un resultado de *unión externa* incluye las filas que se producen por la unión interna, más las filas que faltan, dependiendo de si se utiliza una unión externa izquierda, externa completa, externa derecha o externa completa. Para obtener más información, consulte [“Uniones externas” en la página 403](#).

Unión externa izquierda

Un resultado de *unión externa izquierda* incluye las filas de la tabla izquierda que faltaban en la unión interna. El resultado de T1 LEFT OUTER JOIN T2 consiste en sus filas emparejadas y, para cada fila no emparejada de T1, la concatenación de esa fila con la fila nula de T2. Todas las columnas derivadas de T2 permiten valores nulos. Para obtener más información, consulte [“Unión externa izquierda” en la página 404](#).

Unión externa derecha

Un resultado de *unión externa derecha* incluye las filas de la tabla derecha que faltaban en la unión interna. El resultado de T1 RIGHT OUTER JOIN T2 consiste en sus filas emparejadas y, para cada fila no emparejada de T2, la concatenación de esa fila con la fila nula de T1. Todas las columnas derivadas de T1 permiten valores nulos. Para obtener más información, consulte “[Unión externa derecha](#)” en la página 405.

Unión externa completa

Un resultado de *unión externa completa* incluye las filas de ambas tablas que faltaban en la unión interna. El resultado de T1 FULL OUTER JOIN T2 consiste en sus filas emparejadas y, para cada fila no emparejada de T1, la concatenación de esa fila con la fila nula de T2, y para cada fila no emparejada de T2, la concatenación de esa fila con la fila nula en T1. Todas las columnas de la tabla de resultados permiten valores nulos. Para obtener más información, consulte “[Unión externa completa](#)” en la página 407.

Unión cruzada

Un resultado de *combinación cruzada* incluye el producto cruzado de las tablas, donde cada fila de la tabla izquierda se combina con cada fila de la tabla derecha. Una combinación cruzada también se conoce como *producto cartesiano*. El resultado de un producto cartesiano (T1 CROSS JOIN T2) consiste en cada fila de un par de datos (T1) emparejada con cada fila de un par de datos (T2). También se puede especificar una combinación cruzada sin la sintaxis CROSS JOIN, enumerando las dos tablas en la cláusula FROM separadas por comas sin utilizar una cláusula WHERE para proporcionar criterios de combinación.

ejemplos

Expresiones de tabla anidadas y funciones de tabla definidas por el usuario en uniones

Un operando de una unión puede ser más complejo que el nombre de una sola tabla. Puede especificar uno de los siguientes elementos como operando de unión:

expresión de tabla anidada

Una selección completa que está entre paréntesis y seguida de un nombre de correlación. El nombre de la correlación le permite referirse al resultado de esa expresión.

El uso de una expresión de tabla anidada en una combinación puede ser útil cuando se desea crear una tabla temporal para utilizarla en una combinación. Puede especificar la expresión de tabla anidada como operando derecho o izquierdo de una unión, dependiendo de las filas no coincidentes que deseé incluir.

función de tabla definida por el usuario

Función definida por el usuario que devuelve una tabla.

El uso de una expresión de tabla anidada en una combinación puede ser útil cuando se desea realizar alguna operación en los valores de una tabla antes de combinarlos con otra tabla.

Ejemplo: Uso de referencias correlacionadas

En la siguiente instrucción SELECT, el nombre de correlación que se utiliza para la expresión de tabla anidada es CHEAP_PARTS. Puede utilizar este nombre de correlación para referirse a las columnas que devuelve la expresión. En este caso, esas referencias correlacionadas son CHEAP_PARTS.PROD # y CHEAP_PARTS.PRODUCT.

```
SELECT CHEAP_PARTS.PROD#, CHEAP_PARTS.PRODUCT
  FROM (SELECT PROD#, PRODUCT
         FROM PRODUCTS
        WHERE PRICE < 10) AS CHEAP_PARTS;
```

La tabla de resultados tiene un aspecto similar al siguiente:

PROD#	PRODUCT
=====	=====
505	SCREWDRIVER
30	RELAY

Las referencias correlacionadas son válidas porque no aparecen en la expresión de la tabla donde se define CHEAP_PARTS. Las referencias correlacionadas proceden de una especificación de tabla de un nivel superior en la jerarquía de subconsultas.

Ejemplo: Uso de una expresión de tabla anidada como operando derecho de una combinación

La siguiente consulta contiene un fullselect (en negrita) como operando derecho de una unión externa izquierda con la tabla PROYECTOS. El nombre de la correlación es TEMP. En este caso, se incluyen las filas no coincidentes de la tabla PROYECTOS, pero no las filas no coincidentes de la expresión de tabla anidada.

```
SELECT PROJECT, COALESCE(PROJECTS.PROD#, PRODNUM) AS PRODNUM,
       PRODUCT, PART, UNITS
  FROM PROJECTS LEFT JOIN
    (SELECT PART,
           COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM,
           PRODUCTS.PRODUCT
      FROM PARTS FULL OUTER JOIN PRODUCTS
        ON PARTS.PROD# = PRODUCTS.PROD#) AS TEMP
     ON PROJECTS.PROD# = PRODNUM;
```

Ejemplo: Uso de una expresión de tabla anidada como operando izquierdo de una combinación

La siguiente consulta contiene un fullselect como operando izquierdo de una unión externa izquierda con la tabla PRODUCTS. El nombre de la correlación es PARTX. En este caso, se incluyen las filas no coincidentes de la expresión de tabla anidada, pero no las filas no coincidentes de la tabla PRODUCTS.

```
SELECT PART, SUPPLIER, PRODNUM, PRODUCT
  FROM (SELECT PART, PROD# AS PRODNUM, SUPPLIER
         FROM PARTS
        WHERE PROD# < '200') AS PARTX
  LEFT OUTER JOIN PRODUCTS
    ON PRODNUM = PROD#;
```

La tabla de resultados tiene un aspecto similar al siguiente:

PART	SUPPLIER	PRODNUM	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
OIL	WESTERN_CHEM	160	-----

Dado que PROD# es un campo de caracteres, Db2 realiza una comparación de caracteres para determinar el conjunto de filas en el resultado. Por lo tanto, como los caracteres «30» son mayores que «200», la fila en la que PROD# es igual a «30» no aparece en el resultado.

Ejemplo: Uso de una función de tabla como operando de una unión

Supongamos que CVTPRICE es una función de tabla que convierte los precios de la tabla PRODUCTS a la moneda que usted especifique y devuelve la tabla PRODUCTS con los precios en esas unidades. Puede obtener una tabla de piezas, proveedores y precios de productos con los precios en la moneda que elija ejecutando una consulta similar a la siguiente:

```
SELECT PART, SUPPLIER, PARTS.PROD#, Z.PRODUCT, Z.PRICE
  FROM PARTS, TABLE(CVTPRICE(:CURRENCY)) AS Z
 WHERE PARTS.PROD# = Z.PROD#;
```

Referencias correlacionadas en especificaciones de tablas en uniones

Utilice nombres de correlación para referirse a los resultados de una expresión de tabla anidada. Después de especificar el nombre de correlación para una expresión, cualquier referencia posterior a este nombre de correlación se denomina *referencia correlacionada*.

Puede incluir referencias correlacionadas en expresiones de tabla anidadas o como argumentos de funciones de tabla. La regla básica que se aplica a ambos casos es que la referencia correlacionada debe proceder de una especificación de tabla de un nivel superior en la jerarquía de subconsultas. También puede utilizar una referencia correlacionada y la especificación de la tabla a la que se refiere en la misma cláusula FROM si la especificación de la tabla aparece a la izquierda de la referencia correlacionada y la referencia correlacionada está en una de las siguientes cláusulas:

- Una expresión de tabla anidada que va precedida de la palabra clave TABLE
- El argumento de una función de tabla

Para obtener más información sobre referencias correlacionadas, consulte “[Nombres de correlación en referencias](#)” en la página 418.

Una función de tabla o una expresión de tabla que contenga referencias correlacionadas a otras tablas en la misma cláusula FROM no puede participar en una combinación externa completa o en una combinación externa derecha. Los siguientes ejemplos ilustran usos válidos de referencias correlacionadas en especificaciones de tablas.

En este ejemplo, la referencia correlacionada T.C2 es válida porque la especificación de la tabla, a la que se refiere, T, está a su izquierda.

```
SELECT T.C1, Z.C5
  FROM T, TABLE(TF3(T.C2)) AS Z
 WHERE T.C3 = Z.C4;
```

Si especifica la unión en el orden opuesto, con T después de TABLE(TF3(T.C2) , T.C2 no es válido.

En este ejemplo, la referencia correlacionada D.DEPTNO es válida porque la expresión de tabla anidada dentro de la cual aparece está precedida por TABLE, y la especificación de tabla D aparece a la izquierda de la expresión de tabla anidada en la cláusula FROM.

```
SELECT D.DEPTNO, D.DEPTNAME,
       EMPINFO.AVGSAL, EMPINFO.EMPCOUNT
    FROM DEPT D,
         TABLE(SELECT AVG(E.SALARY) AS AVGSAL,
                COUNT(*) AS EMPCOUNT
               FROM EMP E
              WHERE E.WORKDEPT=D.DEPTNO) AS EMPINFO;
```

Si elimina la palabra clave TABLE, D.DEPTNO no será válido.

Referencia relacionada

[cláusula-from \(Db2 SQL\)](#)

[tabla-unida \(Db2 SQL\)](#)

Unirse a más de dos mesas

Las uniones no están limitadas a dos tablas. Puede unirse a más de dos tablas en una sola instrucción SQL.

Procedimiento

Especifique las condiciones de unión que incluyen columnas de todas las tablas relevantes.

Ejemplo

Ejemplo: Unir tres tablas

Supongamos que desea una tabla de resultados que muestre los empleados que tienen proyectos de los que son responsables, sus proyectos y los nombres de sus departamentos. Debe unirse a tres mesas para obtener toda la información. Puede utilizar la siguiente instrucción SELECT:

```
SELECT EMPNO, LASTNAME, DEPTNAME, PROJNO
  FROM DSN8C10.EMP, DSN8C10.PROJ, DSN8C10.DEPT
 WHERE EMPNO = RESPEMP
   AND WORKDEPT = DSN8C10.DEPT.DEPTNO;
```

La tabla de resultados tiene un aspecto similar al siguiente:

EMPNO	LASTNAME	DEPTNAME	PROJNO
=====	=====	=====	=====
000010	HAAS	SPIFFY COMPUTER SERVICE DIV	AD3100
000010	HAAS	SPIFFY COMPUTER SERVICE DIV	MA2100
000020	THOMPSON	PLANNING	PL2100
000030	KWAN	INFORMATION CENTER	IF1000

000030	KWAN	INFORMATION CENTER	IF2000
000050	GEYER	SUPPORT SERVICES	OP1000
000050	GEYER	SUPPORT SERVICES	OP2000
000060	STERN	MANUFACTURING SYSTEMS	MA2110
000070	PULASKI	ADMINISTRATION SYSTEMS	AD3110
000090	HENDERSON	OPERATIONS	OP1010
000100	SPENSER	SOFTWARE SUPPORT	OP2010
000150	ADAMSON	MANUFACTURING SYSTEMS	MA2112
000160	PIANKA	MANUFACTURING SYSTEMS	MA2113
000220	LUTZ	MANUFACTURING SYSTEMS	MA2111
000230	JEFFERSON	ADMINISTRATION SYSTEMS	AD3111
000250	SMITH	ADMINISTRATION SYSTEMS	AD3112
000270	PEREZ	ADMINISTRATION SYSTEMS	AD3113
000320	MEHTA	SOFTWARE SUPPORT	OP2011
000330	LEE	SOFTWARE SUPPORT	OP2012
000340	GOUNOT	SOFTWARE SUPPORT	OP2013

Db2 determina los resultados intermedios y finales de la consulta anterior mediante los siguientes pasos lógicos:

1. Une las tablas de empleados y proyectos en el número de empleado, eliminando las filas sin número de empleado coincidente en la tabla de proyectos.
2. Une la tabla de resultados intermedios con la tabla de departamentos en los números de departamento coincidentes.
3. Procese la lista de selección en la tabla de resultados finales, dejando solo cuatro columnas.

Ejemplo: Unir más de dos tablas utilizando más de un tipo de unión

Al unir más de dos tablas, no es necesario que utilice el mismo tipo de unión para cada unión.

Para unir tablas utilizando más de un tipo de unión, especifique los tipos de unión en la cláusula FROM.

Supongamos que desea una tabla de resultados que muestre los siguientes elementos:

- Empleados cuyo apellido comienza con «S» o una letra que viene después de «S» en el alfabeto
- Los nombres de los departamentos para estos empleados
- Cualquier proyecto del que estos empleados sean responsables

Puede utilizar la siguiente instrucción SELECT:

```
SELECT EMPNO, LASTNAME, DEPTNAME, PROJNO
  FROM DSN8C10.EMP INNER JOIN DSN8C10.DEPT
    ON WORKDEPT = DSN8C10.DEPT.DEPTNO
   LEFT OUTER JOIN DSN8C10.PROJ
     ON EMPNO = RESPEMP
    WHERE LASTNAME > 'S';
```

La tabla de resultados se parece a la siguiente salida:

EMPNO	LASTNAME	DEPTNAME	PROJNO
000020	THOMPSON	PLANNING	PL2100
000060	STERN	MANUFACTURING SYSTEMS	MA2110
000100	SPENSER	SOFTWARE SUPPORT	OP2010
000170	YOSHIMURA	MANUFACTURING SYSTEMS	-----
000180	SCOUTTEN	MANUFACTURING SYSTEMS	-----
000190	WALKER	MANUFACTURING SYSTEMS	-----
000250	SMITH	ADMINISTRATION SYSTEMS	AD3112
000280	SCHNEIDER	OPERATIONS	-----
000300	SMITH	OPERATIONS	-----
000310	SETRIGHT	OPERATIONS	-----
200170	YAMAMOTO	MANUFACTURING SYSTEMS	-----
200280	SCHWARTZ	OPERATIONS	-----
200310	SPRINGER	OPERATIONS	-----
200330	WONG	SOFTWARE SUPPORT	-----

Db2

determina los resultados intermedios y finales de la consulta anterior mediante los siguientes pasos lógicos:

1. Une las tablas de empleados y departamentos haciendo coincidir los números de departamento y eliminando las filas en las que el apellido comienza con una letra anterior a la «S» del alfabeto.
2. Une la tabla de resultados intermedios con la tabla de proyectos en el número de empleado, manteniendo las filas para las que no existe ningún número de empleado coincidente en la tabla de proyectos.
3. Procese la lista de selección en la tabla de resultados finales, dejando solo cuatro columnas.

Referencia relacionada

[cláusula-from \(Db2 SQL\)](#)

Uniones internas

Una *unión interna* es un método para combinar dos tablas que descarta las filas de cualquiera de las tablas que no coincidan con ninguna fila de la otra tabla. El cotejo se basa en la condición de unión.

Para solicitar una unión interna, ejecute una instrucción SELECT en la que especifique las tablas que desea unir en la cláusula FROM, y especifique una cláusula WHERE o una cláusula ON para indicar la condición de unión. La condición de unión puede ser cualquier condición de búsqueda simple o compuesta que no contenga ninguna referencia de subconsulta.

En el tipo más simple de unión interna, la condición de unión es *columna1 = columna2*.

Ejemplo de unión interna

Para este ejemplo, supongamos que las tablas PARTES y PRODUCTOS contienen las siguientes filas:

PARTS table			PRODUCTS table		
PART	PROD#	SUPPLIER	PROD#	PRODUCT	PRICE
WIRE	10	ACWF	505	SCREWDRIVER	3.70
OIL	160	WESTERN_CHEM	30	RELAY	7.55
MAGNETS	10	BATEMAN	205	SAW	18.90
PLASTIC	30	PLASTIK_CORP	10	GENERATOR	45.75
BLADES	205	ACE_STEEL			

Para unir las tablas de PIEZAS y PRODUCTOS en la columna PROD# y obtener una tabla de piezas con sus proveedores y los productos que utilizan las piezas, puede utilizar una de las siguientes instrucciones SELECT:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS, PRODUCTS
 WHERE PARTS.PROD# = PRODUCTS.PROD#;
```

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS INNER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#;
```

La tabla de resultados tiene el siguiente aspecto:

PART	SUPPLIER	PROD#	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_CORP	30	RELAY
BLADES	ACE_STEEL	205	SAW

Tres cosas sobre este ejemplo:

- Una parte de la tabla de partes (OIL) tiene un producto (n.º 160), que no está en la tabla de productos. Un producto (DESTORNILLADOR, n.º 505) no tiene piezas enumeradas en la tabla de piezas. Ni ACEITE ni DESATORNILLADOR aparecen en el resultado de la unión.

Por el contrario, *una unión externa* incluye filas en las que los valores de las columnas unidas no coinciden.

- Puede especificar explícitamente que esta combinación es una combinación interna (no externa). Utilice INNER JOIN en la cláusula FROM en lugar de la coma, y utilice ON para especificar la condición de unión (en lugar de WHERE) cuando se unen tablas explícitamente en la cláusula FROM.
- Si no especifica una cláusula WHERE en la primera forma de la consulta, la tabla de resultados contiene todas las combinaciones posibles de filas para las tablas que se identifican en la cláusula FROM. Puede obtener el mismo resultado especificando una condición de unión que sea siempre verdadera en la segunda forma de la consulta, como en la siguiente declaración:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS INNER JOIN PRODUCTS
    ON 1=1;
```

Independientemente de si omite la cláusula WHERE o especifica una condición de unión que siempre es verdadera, el número de filas de la tabla de resultados es el producto del número de filas de cada tabla.

Puede especificar condiciones de unión más complicadas para obtener diferentes conjuntos de resultados. Por ejemplo, para eliminar de la tabla de piezas, proveedores, números de producto y productos a los proveedores que empiezan por **la letra A**, escriba una consulta como la siguiente:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS INNER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#
   AND SUPPLIER NOT LIKE 'A%';
```

El resultado de la consulta son todas las filas que no tienen un proveedor que comience por A. La tabla de resultados tiene el siguiente aspecto:

PART	SUPPLIER	PROD#	PRODUCT
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_Corp	30	RELAY

Ejemplo de unión de una tabla consigo misma mediante una unión interna

Unir una tabla consigo misma es útil para mostrar relaciones entre filas. El siguiente ejemplo devuelve una lista de los principales proyectos de la tabla PROJ y los proyectos que forman parte de esos principales proyectos.

En este ejemplo, **A** indica la primera instancia de la tabla DSN8C10.PROJ, y **B** indica la segunda instancia de esta tabla. La condición de unión es tal que el valor en la columna PROJNO en la tabla DSN8C10.PROJ A debe ser igual a un valor en la columna MAJPROJ en la tabla DSN8C10.PROJ B.

La siguiente instrucción SQL une la tabla DSN8C10.PROJ a sí mismo y devuelve el número y el nombre de cada proyecto principal, seguidos del número y el nombre del proyecto que forma parte de él:

```
SELECT A.PROJNO, A.PROJNAME, B.PROJNO, B.PROJNAME
  FROM DSN8C10.PROJ A, DSN8C10.PROJ B
 WHERE A.PROJNO = B.MAJPROJ;
```

La tabla de resultados tiene un aspecto similar al siguiente:

PROJNO	PROJNAME	PROJNO	PROJNAME
AD3100	ADMIN SERVICES	AD3110	GENERAL AD SYSTEMS
AD3110	GENERAL AD SYSTEMS	AD3111	PAYROLL PROGRAMMING
AD3110	GENERAL AD SYSTEMS	AD3112	PERSONNEL PROGRAMMG
OP2010	SYSTEMS SUPPORT	OP2013	DB/DC SUPPORT

En este ejemplo, la coma en la cláusula FROM especifica implícitamente una unión interna y actúa del mismo modo que si se hubieran utilizado las palabras clave INNER JOIN. Cuando utilice la coma para una unión interna, debe especificar la condición de unión en la cláusula WHERE. Cuando utilice las palabras clave INNER JOIN, debe especificar la condición de unión en la cláusula ON.

Conceptos relacionados

Uniones externas

Una unión externa es un método para combinar dos o más tablas de modo que el resultado incluya filas no coincidentes de una de las tablas, o de ambas. El cotejo se basa en la condición de unión.

Referencia relacionada

cláusula-from (Db2 SQL)

tabla-unida (Db2 SQL)

Uniones externas

Una unión externa es un método para combinar dos o más tablas de modo que el resultado incluya filas no coincidentes de una de las tablas, o de ambas. El cotejo se basa en la condición de unión.

Db2 admite tres tipos de uniones externas:

Unión externa izquierda

Un resultado de *unión externa izquierda* incluye las filas de la tabla izquierda que faltaban en la unión interna. El resultado de T1 LEFT OUTER JOIN T2 consiste en sus filas emparejadas y, para cada fila no emparejada de T1, la concatenación de esa fila con la fila nula de T2. Todas las columnas derivadas de T2 permiten valores nulos. Para obtener más información, consulte “[Unión externa izquierda](#)” en la página 404.

Unión externa derecha

Un resultado de *unión externa derecha* incluye las filas de la tabla derecha que faltaban en la unión interna. El resultado de T1 RIGHT OUTER JOIN T2 consiste en sus filas emparejadas y, para cada fila no emparejada de T2, la concatenación de esa fila con la fila nula de T1. Todas las columnas derivadas de T1 permiten valores nulos. Para obtener más información, consulte “[Unión externa derecha](#)” en la página 405.

Unión externa completa

Un resultado de *unión externa completa* incluye las filas de ambas tablas que faltaban en la unión interna. El resultado de T1 FULL OUTER JOIN T2 consiste en sus filas emparejadas y, para cada fila no emparejada de T1, la concatenación de esa fila con la fila nula de T2, y para cada fila no emparejada de T2, la concatenación de esa fila con la fila nula en T1. Todas las columnas de la tabla de resultados permiten valores nulos. Para obtener más información, consulte “[Unión externa completa](#)” en la página 407.

Ejemplos de uniones exteriores

Los siguientes ejemplos utilizan dos tablas: la tabla de piezas (PARTS) y la tabla de productos (PRODUCTS), que consisten en suministros de hardware.

La figura siguiente muestra que cada fila de la tabla PARTS contiene datos de un único componente: el nombre de componente, el número de componente y el proveedor del componente.

PARTS

PART	Prod#	Proveedor
Cables	10	ACWF
petróleo	160	WESTERN_CHEM
IMÁGENES	10	BATEMAN
Plástico	30	PLASTK_CORP
Blades	205	ACE_STEEL

Figura 18. Tabla PARTS de ejemplo

La figura siguiente muestra que cada fila de la tabla PRODUCTS contiene datos para un único producto: el número, el nombre y el precio del producto.

Productos		
Prod#	PRODUCTO	PRECIO
505	DESTORNILLADOR	3.70
30	Retransmitir	7.55
205	SAW	18.90
10	generador	45.75

Figura 19. Tabla PRODUCTS de ejemplo

La figura siguiente muestra las distintas formas de combinar las tablas PARTS y PRODUCTS utilizando funciones de unión externa. La ilustración se basa en un subconjunto de columnas de cada tabla.

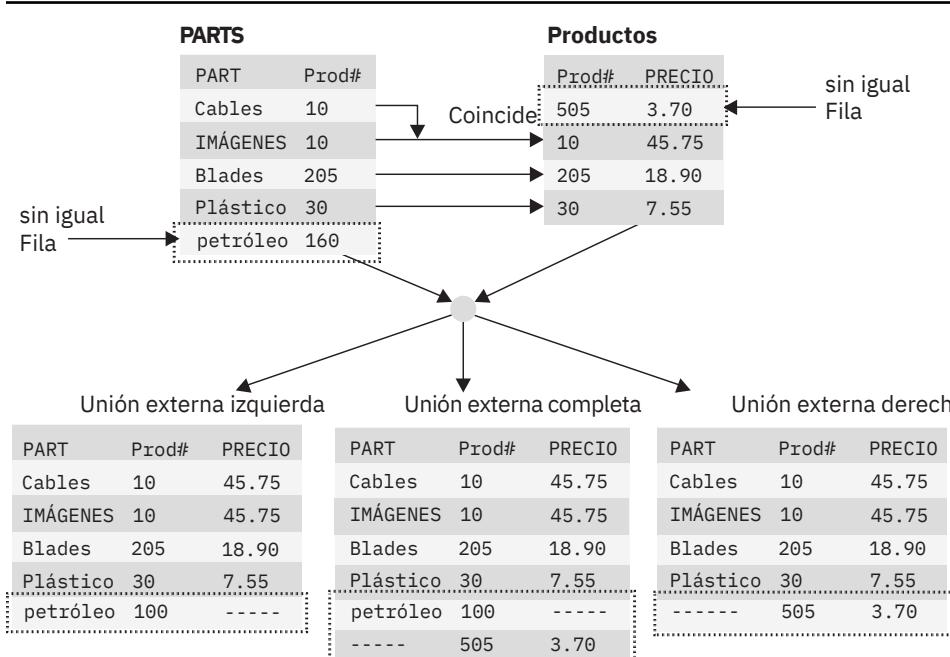


Figura 20. Uniones externas de dos tablas

Una unión interna consta de filas formadas a partir de las tablas PARTS y PRODUCTS, basándose en la coincidencia de igualdad de valores de columna entre la columna PROD# de la tabla PARTS y la columna PROD# de la tabla PRODUCTS. La unión interna no contiene ninguna fila formada a partir de columnas no coincidentes cuando las columnas PROD# no son iguales.

Se pueden especificar uniones en la cláusula FROM de una consulta. Se unen los datos de las filas que cumplen las condiciones de búsqueda de todas las tablas para formar la tabla de resultados.

Las columnas resultantes de una unión tienen nombres si la lista SELECT más exterior hace referencia a columnas base. Sin embargo, si se utiliza una función (por ejemplo, COALESCE) para crear una columna del resultado, dicha columna no tiene ningún nombre a menos que se utilice la cláusula AS en la lista SELECT.

Conceptos relacionados

Uniones internas

Una *unión interna* es un método para combinar dos tablas que descarta las filas de cualquiera de las tablas que no coincidan con ninguna fila de la otra tabla. El cotejo se basa en la condición de unión.

Referencia relacionada

[tabla-unida \(Db2 SQL\)](#)

Unión externa izquierda

Una *unión externa izquierda* es un método para combinar tablas. El resultado incluye sólo las filas no coincidentes de la tabla que se ha especificado antes de la cláusula LEFT OUTER JOIN.

Si se une a dos tablas y desea que el conjunto de resultados incluya filas no coincidentes de una sola tabla, utilice una cláusula LEFT OUTER JOIN o una cláusula RIGHT OUTER JOIN. El cotejo se basa en la condición de unión.

La cláusula LEFT OUTER JOIN incluye filas de la tabla especificada antes de LEFT OUTER JOIN que no tienen valores coincidentes en la tabla especificada después de LEFT OUTER JOIN.

Al igual que en una unión interna, la condición de unión puede ser cualquier condición de búsqueda simple o compuesta que no contenga una referencia de subconsulta.

Ejemplo de unión externa izquierda

Para este ejemplo, supongamos que las tablas PARTES y PRODUCTOS contienen las siguientes filas:

PARTS table			PRODUCTS table		
PART	PROD#	SUPPLIER	PROD#	PRODUCT	PRICE
WIRE	10	ACWF	505	SCREWDRIVER	3.70
OIL	160	WESTERN_CHEM	30	RELAY	7.55
MAGNETS	10	BATEMAN	205	SAW	18.90
PLASTIC	30	PLASTIK_Corp	10	GENERATOR	45.75
BLADES	205	ACE_STEEL			

Para incluir filas de la tabla PARTS que no tienen valores coincidentes en la tabla PRODUCTS, y para incluir precios que exceden 10.00, ejecute la siguiente consulta:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT, PRICE
  FROM PARTS LEFT OUTER JOIN PRODUCTS
    ON PARTS.PROD#=PRODUCTS.PROD#
   AND PRODUCTS.PRICE>10.00;
```

La tabla de resultados tiene un aspecto similar al siguiente:

PART	SUPPLIER	PROD#	PRODUCT	PRICE
WIRE	ACWF	10	GENERATOR	45.75
MAGNETS	BATEMAN	10	GENERATOR	45.75
PLASTIC	PLASTIK_Corp	30	-----	-----
BLADES	ACE_STEEL	205	SAW	18.90
OIL	WESTERN_CHEM	160	-----	-----

Una fila de la tabla PRODUCTOS solo aparece en la tabla de resultados si su número de producto coincide con el número de producto de una fila de la tabla PIEZAS y el precio es superior a 10.00 para esa fila. Las filas en las que el valor PRICE no supera el valor de 10.00 se incluyen en el resultado de la unión, pero el valor PRICE se establece en nulo.

En esta tabla de resultados, la fila del PROD# 30 tiene valores nulos en las dos columnas de la derecha porque el precio del PROD# 30 es inferior a 10.00. El PROD# 160 tiene valores nulos en las dos columnas de la derecha porque el PROD# 160 no coincide con otro número de producto.

Conceptos relacionados

Unión externa derecha

Una unión externa derecha es un método para combinar tablas. El resultado incluye sólo las filas no coincidentes de la tabla que se ha especificado después de la cláusula RIGHT OUTER JOIN.

Unión externa completa

Una unión externa completa es un método para combinar tablas de modo que el resultado incluya filas no coincidentes de ambas tablas.

Referencia relacionada

[tabla-unida \(Db2 SQL\)](#)

Unión externa derecha

Una unión externa derecha es un método para combinar tablas. El resultado incluye sólo las filas no coincidentes de la tabla que se ha especificado después de la cláusula RIGHT OUTER JOIN.

Si se une a dos tablas y desea que el conjunto de resultados incluya filas no coincidentes de una sola tabla, utilice una cláusula LEFT OUTER JOIN o una cláusula RIGHT OUTER JOIN. El cotejo se basa en la condición de unión.

La cláusula RIGHT OUTER JOIN incluye filas de la tabla especificada después de RIGHT OUTER JOIN que no tienen valores coincidentes en la tabla especificada antes de RIGHT OUTER JOIN.

Al igual que en una unión interna, la condición de unión puede ser cualquier condición de búsqueda simple o compuesta que no contenga una referencia de subconsulta.

Ejemplo de unión externa derecha

Para este ejemplo, supongamos que las tablas PARTES y PRODUCTOS contienen las siguientes filas:

PARTS table			PRODUCTS table		
PART	PROD#	SUPPLIER	PROD#	PRODUCT	PRICE
WIRE	10	ACWF	505	SCREWDRIVER	3.70
OIL	160	WESTERN_CHEM	30	RELAY	7.55
MAGNETS	10	BATEMAN	205	SAW	18.90
PLASTIC	30	PLASTIK_Corp	10	GENERATOR	45.75
BLADES	205	ACE_STEEL			

Para incluir filas de la tabla PRODUCTS que no tienen filas correspondientes en la tabla PARTS, ejecute esta consulta:

```
SELECT PART, SUPPLIER, PRODUCTS.PROD#, PRODUCT, PRICE
  FROM PARTS RIGHT OUTER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#
   AND PRODUCTS.PRICE > 10.00;
```

La tabla de resultados tiene un aspecto similar al siguiente:

PART	SUPPLIER	PROD#	PRODUCT	PRICE
WIRE	ACWF	10	GENERATOR	45.75
MAGNETS	BATEMAN	10	GENERATOR	45.75
BLADES	ACE_STEEL	205	SAW	18.90
		30	RELAY	7.55
		505	SCREWDRIVER	3.70

Una fila de la tabla PARTES solo aparece en la tabla de resultados si su número de producto coincide con el número de producto de una fila de la tabla PRODUCTOS y el precio es mayor que el de esa fila (10.00).

Debido a que la tabla PRODUCTOS puede tener filas con números de producto no coincidentes en la tabla de resultados, y la columna PRECIO está en la tabla PRODUCTOS, se incluyen en el resultado las filas en las que el PRECIO es menor o igual a 10.00. Las columnas PARTES contienen valores nulos para estas filas en la tabla de resultados.

Conceptos relacionados

Uniones externas

Una unión externa es un método para combinar dos o más tablas de modo que el resultado incluya filas no coincidentes de una de las tablas, o de ambas. El cotejo se basa en la condición de unión.

Unión externa izquierda

Una unión externa izquierda es un método para combinar tablas. El resultado incluye sólo las filas no coincidentes de la tabla que se ha especificado antes de la cláusula LEFT OUTER JOIN.

Unión externa completa

Una unión externa completa es un método para combinar tablas de modo que el resultado incluya filas no coincidentes de ambas tablas.

Referencia relacionada

[tabla-unida \(Db2 SQL\)](#)

Unión externa completa

Una *unión externa completa* es un método para combinar tablas de modo que el resultado incluya filas no coincidentes de ambas tablas.

Si se une a dos tablas y desea que el conjunto de resultados incluya filas no coincidentes de ambas tablas, utilice una cláusula FULL OUTER JOIN. El cotejo se basa en la condición de unión. Si alguna columna de la tabla de resultados no tiene un valor, esa columna tiene el valor nulo en la tabla de resultados.

La condición de unión para una unión externa completa debe ser una condición de búsqueda simple que compare dos columnas o una invocación de una función de conversión que tenga un nombre de columna como argumento.

Ejemplos de uniones exteriores completas

Para este ejemplo, supongamos que las tablas PARTES y PRODUCTOS contienen las siguientes filas:

PARTS table			PRODUCTS table		
PART	PROD#	SUPPLIER	PROD#	PRODUCT	PRICE
WIRE	10	ACWF	505	SCREWDRIVER	3.70
OIL	160	WESTERN_CHEM	30	RELAY	7.55
MAGNETS	10	BATEMAN	205	SAW	18.90
PLASTIC	30	PLASTIK_Corp	10	GENERATOR	45.75
BLADES	205	ACE_STEEL			

La siguiente consulta realiza una unión externa completa de las tablas PARTES y PRODUCTOS:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS FULL OUTER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#;
```

La tabla de resultados de la consulta tiene un aspecto similar al siguiente:

PART	SUPPLIER	PROD#	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_Corp	30	RELAY
BLADES	ACE_STEEL	205	SAW
OIL	WESTERN_CHEM	160	-----
-----	-----	---	SCREWDRIVER

Ejemplo de unión externa completa utilizando COALESCE o VALUE

COALESCE es la palabra clave que especifica el estándar SQL como sinónimo de la función VALUE. Esta función, con cualquiera de los dos nombres, puede ser especialmente útil en operaciones de unión externa completa porque devuelve el primer valor no nulo del par de columnas de unión.

El número de producto en el resultado del ejemplo para “[Unión externa completa](#)” en la página 407 es nulo para SCREWDRIVER, aunque la tabla PRODUCTS contiene un número de producto para SCREWDRIVER. Si por el contrario selecciona PRODUCTS.PROD#, PROD# es nulo para OIL. Si selecciona tanto PRODUCTS.PROD # como PARTS.PROD #, el resultado contiene dos columnas, ambas con algunos valores nulos. Puede fusionar datos de ambas columnas en una única columna, eliminando los valores nulos, mediante la función COALESCE.

Con las mismas tablas de PARTES y PRODUCTOS, el siguiente ejemplo combina los datos no nulos de las columnas PROD#:

```
SELECT PART, SUPPLIER,
  COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM, PRODUCT
  FROM PARTS FULL OUTER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#;
```

La tabla de resultados tiene un aspecto similar al siguiente:

PART	SUPPLIER	PRODNUM	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_Corp	30	RELAY
BLADES	ACE_STEEL	205	SAW
OIL	WESTERN_CHEM	160	-----
		505	SCREWDRIVER

La cláusula AS (AS PRODNUM) proporciona un nombre para el resultado de la función COALESCE.

Conceptos relacionados

Uniones externas

Una unión externa es un método para combinar dos o más tablas de modo que el resultado incluya filas no coincidentes de una de las tablas, o de ambas. El cotejo se basa en la condición de unión.

Unión externa izquierda

Una unión externa izquierda es un método para combinar tablas. El resultado incluye sólo las filas no coincidentes de la tabla que se ha especificado antes de la cláusula LEFT OUTER JOIN.

Unión externa derecha

Una unión externa derecha es un método para combinar tablas. El resultado incluye sólo las filas no coincidentes de la tabla que se ha especificado después de la cláusula RIGHT OUTER JOIN.

Referencia relacionada

[tabla-unida \(Db2 SQL\)](#)

Reglas SQL para sentencias que contienen operaciones de unión

Por lo general, Db2 realiza primero una operación de unión, antes de evaluar las demás cláusulas de la instrucción SELECT.

Las reglas SQL dictan que el resultado de una instrucción SELECT debe parecer como si las cláusulas se hubieran evaluado en este orden:

- FROM
- WHERE
- GROUP BY
- HAVING
- SELECT

Una operación de unión forma parte de una cláusula FROM; por lo tanto, con el fin de predecir qué filas se devolverán de una sentencia SELECT que contiene una operación de unión, asuma que la operación de unión se realiza primero.

Ejemplo : Supongamos que desea obtener una lista de nombres de piezas, nombres de proveedores, números de productos y nombres de productos de las tablas PARTES y PRODUCTOS. Desea incluir filas de cualquiera de las tablas en las que el valor PROD# no coincide con un valor PROD# de la otra tabla, lo que significa que necesita hacer una unión externa completa. También desea excluir las filas del producto número 10. Considere la sentencia SELECT siguiente:

```
SELECT PART, SUPPLIER,
       VALUE(PARTS.PROD#,PRODUCTS.PROD#) AS PRODNUM, PRODUCT
  FROM PARTS FULL OUTER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#
   WHERE PARTS.PROD# <> '10' AND PRODUCTS.PROD# <> '10';
```

El siguiente resultado **no** es el que deseaba:

PART	SUPPLIER	PRODNUM	PRODUCT
PLASTIC	PLASTIK_Corp	30	RELAY
BLADES	ACE_STEEL	205	SAW

Db2 realiza primero la operación de unión. El resultado de la operación de unión incluye filas de una tabla que no tienen filas correspondientes de la otra tabla. Sin embargo, la cláusula WHERE excluye las filas de ambas tablas que tienen valores nulos para la columna PROD#.

La siguiente sentencia es una sentencia SELECT correcta para producir la lista:

```
SELECT PART, SUPPLIER,
       VALUE(X.PROD#, Y.PROD#) AS PRODNUM, PRODUCT
  FROM
    (SELECT PART, SUPPLIER, PROD# FROM PARTS WHERE PROD# <> '10') X
  FULL OUTER JOIN
    (SELECT PROD#, PRODUCT FROM PRODUCTS WHERE PROD# <> '10') Y
      ON X.PROD# = Y.PROD#;
```

Para esta sentencia, Db2 aplica la cláusula WHERE a cada tabla por separado. Db2 luego realiza la operación de unión externa completa, que incluye filas de una tabla que no tienen una fila correspondiente en la otra tabla. El resultado final incluye filas con el valor nulo para la columna PROD# y tiene un aspecto similar al siguiente resultado:

PART	SUPPLIER	PRODNUM	PRODUCT
OIL	WESTERN_CHEM	160	
BLADES	ACE_STEEL	205	SAW
PLASTIC	PLASTIK_CORP	30	RELAY
		505	SCREWDRIVER

Optimización de la recuperación para un conjunto de filas pequeño

Cuando solo necesita que unos pocos miles de filas satisfagan una consulta, puede decirle a Db2 que optimice su proceso de recuperación para devolver únicamente un número de filas especificado.

Acerca de esta tarea

Pregunta : ¿Cómo puedo indicar a Db2 que solo quiero algunas de las miles de filas que satisfacen una consulta?

Respuesta : Utilice la cláusula optimize o la cláusula fetch de la sentencia SELECT.

Db2 normalmente optimiza las consultas para recuperar todas las filas que cumplen los criterios. Pero a veces quieras recuperar unas pocas filas. Por ejemplo, para recuperar la primera fila que sea mayor o igual que un valor conocido, codifique la instrucción SELECT de la siguiente manera:

```
SELECT column list FROM table
  WHERE key >= value
  ORDER BY key ASC
```

Incluso con la cláusula ORDER BY, Db2 podría recuperar todos los datos primero y ordenarlos después de la recuperación, lo que podría afectar al rendimiento. En su lugar, puede escribir la consulta de una de las siguientes maneras:

```
SELECT * FROM table
  WHERE key >= value
  ORDER BY key ASC
  OPTIMIZE FOR 1 ROW
```

```
SELECT * FROM table
  WHERE key >= value
  ORDER BY key ASC
  FETCH FIRST n ROWS ONLY
```

Utilice la cláusula OPTIMIZE FOR 1 ROW para influir en la ruta de acceso. OPTIMIZE FOR 1 ROW indica a Db2 que seleccione una vía de acceso que devuelve la primera fila cualificada.

Utilice la cláusula FETCH FIRST n ROWS ONLY para limitar el número de filas de la tabla de resultados a n filas. OBTENER PRIMERO n ÚNICAMENTE FILAS tiene los siguientes beneficios:

- Cuando se utilizan sentencias FETCH para recuperar datos de una tabla de resultados, la cláusula fetch hace que Db2 recupere solo el número de filas que se necesitan. Esto puede tener beneficios de rendimiento, especialmente en aplicaciones distribuidas. Si intenta ejecutar una sentencia FETCH para recuperar *la fila n+1*, Db2 devuelve un SQLCODE +100.
- Cuando se utiliza la cláusula fetch en una sentencia SELECT INTO, nunca se recupera más de una fila. El uso de la cláusula fetch en una instrucción SELECT INTO puede evitar errores SQL causados por la selección inadvertida de más de un valor en una variable de host.

Cuando se especifica la cláusula fetch pero no la cláusula optimize, la cláusula optimize es implícita. Cuando se especifica FETCH FIRST *n* ROWS ONLY y OPTIMIZE FOR *m* ROWS y *m* es menor que *n*, Db2 optimiza la consulta para *m* filas. Si *m* es mayor que *n*, Db2 optimiza la consulta para *n* filas.

Tareas relacionadas

[Captación de un número limitado de filas \(Db2 Performance\)](#)

Referencia relacionada

[cláusula-optimize \(Db2 SQL\)](#)

[cláusula-fetch \(Db2 SQL\)](#)

Creación de SQL recursivo mediante expresiones de tabla comunes

Las consultas que utilizan la recursividad son útiles en aplicaciones como las de listas de materiales, planificación de redes y sistemas de reservas.

Acerca de esta tarea

Puede utilizar expresiones de tabla comunes para crear SQL recursivo. Si un fullselect de una expresión de tabla común contiene una referencia a sí misma en una cláusula FROM, la expresión de tabla común es *una expresión de tabla común recursiva*.

Las expresiones de tabla comunes recursivas deben seguir estas reglas:

- La primera fullselect de la primera unión (la fullselect de inicialización) no debe incluir una referencia a la expresión de tabla común.
- Cada fullselect que forma parte del ciclo de recursión debe:
 - Empiece con SELECCIONAR o SELECCIONAR TODO. SELECT DISTINCT no está permitido.
 - Incluya solo una referencia a la expresión de tabla común que forma parte del ciclo de recursión en su cláusula FROM.
 - No incluir funciones agregadas, una cláusula GROUP BY o una cláusula HAVING.
- Los nombres de las columnas deben especificarse después del nombre de la tabla de la expresión de tabla común.
- El tipo de datos, la longitud y el CCSID de cada columna de la expresión de tabla común deben coincidir con el tipo de datos, la longitud y el CCSID de cada columna correspondiente en el fullselect iterativo.
- Si utiliza la palabra clave UNION, especifique UNION ALL en lugar de UNION.
- No puede especificar INTERSECT o EXCEPT.
- Las uniones externas no deben formar parte de ningún ciclo de recursión.
- Una subconsulta no debe formar parte de ningún ciclo de recursividad.

Importante: Debe tener cuidado para evitar un bucle infinito cuando utilice una expresión de tabla común recursiva. Db2 emite una advertencia si uno de los siguientes elementos **no se encuentra** en la selección completa iterativa de una expresión de tabla común recursiva:

- Una columna de enteros que se incrementa en una constante
- Un predicado en la cláusula WHERE en forma de *counter_column < constant* o *counter_column < :host variable*

Visite “[Ejemplos de expresiones de tablas comunes recursivas](#)” en la página 156 para ver ejemplos de aplicaciones de listas de materiales que utilizan expresiones de tabla comunes recursivas.

Actualización de datos a medida que se recuperan de la base de datos

A medida que recupera filas, puede actualizarlas al mismo tiempo.

Acerca de esta tarea

Pregunta : ¿Cómo puedo actualizar las filas de datos a medida que las recupero?

Respuesta : En la sentencia SELECT, utilice la cláusula FOR UPDATE sin una lista de columnas, o la cláusula FOR UPDATE OF con una lista de columnas. Para que el programa sea más eficiente, especifique una lista de columnas con solo aquellas que deseé actualizar. A continuación, utilice la instrucción UPDATE posicionada. La cláusula WHERE CURRENT OF identifica el cursor que apunta a la fila que desea actualizar.

Evitar errores aritméticos decimales

Cuando solicite que Db2 realice una operación decimal, pueden producirse errores si Db2 no utiliza la precisión y la escala adecuadas.

Acerca de esta tarea

Para las sentencias SQL estáticas, la forma más sencilla de evitar un error de división es anular las reglas de " DEC31 " especificando la opción de precompilador DEC(15). En algunos casos, puede evitar un error de división especificando D31.s, donde s es un número en el rango de 1 a 9 y representa la escala mínima que se utilizará para las operaciones de división. Esta especificación reduce la probabilidad de errores en las sentencias que están integradas en el programa.

Si las sentencias SQL dinámicas tienen un comportamiento de enlace, definición o invocación y el valor de la opción de instalación USE FOR DYNAMICRULES en el panel DSNTIP4 es NO, puede utilizar la opción de precompilador DEC(15), DEC15 o D15.s para anular las reglas de DEC31, donde s es un número en el rango de 1 a 9.

Para una declaración dinámica, o para una sola declaración estática, utilice la función escalar DECIMAL para especificar valores de la precisión y la escala para un resultado que no cause errores.

Antes de ejecutar una sentencia dinámica, establezca el valor del registro especial CURRENT PRECISION en un DEC15 o D15.s, donde s es un número entre 1 y 9

Incluso si utiliza reglas de e DEC31, las operaciones de multiplicación pueden provocar a veces un desbordamiento porque la precisión del producto es superior a 31. Para evitar el desbordamiento de la multiplicación de números grandes, utilice la función incorporada MULTIPLY_ALT en lugar del operador de multiplicación.

Precisión para operaciones con números decimales

Db2 acepta dos conjuntos de reglas para determinar la precisión y escala del resultado de una operación con números decimales.

- DEC15 permiten una precisión máxima de 15 dígitos en el resultado de una operación. DEC15 las reglas están en vigor cuando ambos operandos tienen una precisión de 15 o menos, o a menos que se apliquen las reglas de la e DEC31.
- DEC31 permiten una precisión máxima de 31 dígitos en el resultado. DEC31 las reglas están en vigor si se cumple alguna de las siguientes condiciones:
 - Cualquiera de los operandos de la operación tiene una precisión superior a 15 dígitos.
 - La operación se encuentra en una instrucción SQL dinámica y se cumple cualquiera de las siguientes condiciones:
 - El valor actual del registro especial CURRENT PRECISION es DEC31 o D31.s, donde s es un número en el rango 1-9 y representa la escala mínima que se utilizará para las operaciones de división.
 - La opción de instalación para ARITMÉTICA DECIMAL en el panel DSNTIP4 es DEC31, 31 o D31.s, donde s es un número en el rango de 1 a 9; la opción de instalación para USE FOR DYNAMICRULES

en el panel DSNTIP4 es SÍ; y el valor de PRECISIÓN ACTUAL no ha sido establecido por la aplicación.

- La instrucción SQL tiene comportamiento de enlace, definición o invocación; la instrucción está en una aplicación precompilada con la opción DEC(31); la opción de instalación para USE FOR DYNAMICRULES en el panel DSNTIP4 es NO; y el valor de CURRENT PRECISION no ha sido establecido por la aplicación. Visite [“Opciones de reglas dinámicas para sentencias de SQL dinámico”](#) en la página 944 para obtener una explicación del comportamiento de vincular, definir e invocar.
- La operación se encuentra en una instrucción SQL incrustada (estática) que usted precompiló con la opción DEC(31), DEC31, o D31.s, o con el valor predeterminado para esa opción cuando la opción de instalación ARITMÉTICA DECIMAL es DEC31 o 31. s es un número en el rango 1-9 y representa la escala mínima que se utilizará para las operaciones de división. Visite [“Procesamiento de instrucciones SQL para la preparación de programas”](#) en la página 896 para obtener información sobre la precompilación y una lista de todas las opciones de precompilación.

Recomendación: Para reducir la posibilidad de desbordamiento, o cuando se trata de una precisión superior a 15 dígitos, elija DEC31 o D31.s, donde s es un número en el rango 1-9 y representa la escala mínima que se utilizará para las operaciones de división.

Controlar cómo un Db2 a los números decimales de punto flotante

Puede especificar un modo de redondeo predeterminado que Db2 utilizará para todos los valores DECFLOAT.

Procedimiento

Establezca el registro especial CURRENT DECFLOAT ROUNDING MODE.

Referencia relacionada

[MODO DE REDONDEADO DECFLOAT ACTUAL registro especial \(Db2 SQL\)](#)

[SET CURRENT DECFLOAT ROUNDING MODE declaración \(Db2 SQL\)](#)

Implicaciones del uso de SELECT *

En general, solo debe utilizar SELECT * cuando desee seleccionar todas las columnas, excepto las columnas ocultas. De lo contrario, especifique las columnas específicas que desea ver.

Pregunta : ¿Cuáles son las implicaciones de usar SELECT *?

Respuesta : Por lo general, debe seleccionar solo las columnas que necesita, ya que Db2 es sensible al número de columnas seleccionadas. Utilice SELECCIONAR * solo cuando esté seguro de que desea seleccionar todas las columnas, excepto las ocultas. (Las columnas ocultas no se devuelven cuando se especifica SELECT *) Una alternativa a seleccionar todas las columnas es utilizar vistas definidas solo con las columnas necesarias y utilizar SELECT * para acceder a las vistas. Evite SELECT * si todas las columnas seleccionadas participan en una operación de clasificación (SELECT DISTINCT y SELECT.UNIÓN, por ejemplo).

Subconsultas

Si necesita restringir la condición de búsqueda en función de la información de una tabla temporal, puede utilizar una subconsulta. Por ejemplo, puede que desee buscar todos los números de empleado de una tabla que también existan para un proyecto concreto en una segunda tabla.

Descripción general conceptual de las subconsultas

Supongamos que desea una lista de los números de empleado, nombres y comisiones de todos los empleados que trabajan en un proyecto concreto, cuyo número de proyecto es MA2111. La primera parte de la sentencia SELECT es fácil de escribir:

```
SELECT EMPNO, LASTNAME, COMM  
  FROM DSN8C10.EMP
```

```
WHERE EMPNO  
:
```

Sin embargo, no puede continuar porque el DSN8C10.EMP la tabla no incluye datos de número de proyecto. No sabe qué empleados están trabajando en el proyecto MA2111 sin emitir otra instrucción SELECT contra el DSN8C10.EMPPROJECT tabla.

Puede utilizar una subconsulta para resolver este problema. *Una subconsulta* es una subselección o una selección completa en una cláusula WHERE. La sentencia SELECT que rodea la subconsulta se denomina *SELECT externo*.

```
SELECT EMPNO, LASTNAME, COMM  
  FROM DSN8C10.EMP  
 WHERE EMPNO IN  
       (SELECT EMPNO  
        FROM DSN8C10.EMPPROJECT  
        WHERE PROJNO = 'MA2111');
```

Para comprender mejor los resultados de esta instrucción SQL, imagine que Db2 sigue el siguiente proceso:

1. Db2 evalúa la subconsulta para obtener una lista de valores de EMPNO:

```
(SELECT EMPNO  
  FROM DSN8C10.EMPPROJECT  
 WHERE PROJNO = 'MA2111');
```

El resultado se encuentra en una tabla de resultados provisionales, similar a la del siguiente resultado:

```
from EMPNO  
=====  
200  
200  
200  
220
```

2. A continuación, la tabla de resultados intermedios sirve como lista en la condición de búsqueda de la sentencia SELECT externa. Db2 , de hecho, cumple esta afirmación:

```
SELECT EMPNO, LASTNAME, COMM  
  FROM DSN8C10.EMP  
 WHERE EMPNO IN  
 ('000200', '000220');
```

Como consecuencia, la tabla de resultados se parece a la siguiente salida:

EMPNO	LASTNAME	COMM
=====	=====	====
000200	BROWN	2217
000220	LUTZ	2387

Subconsultas correlacionadas y no correlacionadas

Las subconsultas proporcionan información necesaria para calificar una fila (en una cláusula WHERE) o un grupo de filas (en una cláusula HAVING). La subconsulta produce una tabla de resultados que se utiliza para calificar la fila o el grupo de filas seleccionadas.

Una subconsulta se ejecuta solo una vez, si la subconsulta es la misma para cada fila o grupo. Este tipo de subconsulta *no* está correlacionada, lo que significa que se ejecuta solo una vez. Por ejemplo, en la siguiente declaración, el contenido de la subconsulta es el mismo para cada fila de la tabla DSN8C10.EMP:

```
SELECT EMPNO, LASTNAME, COMM  
  FROM DSN8C10.EMP  
 WHERE EMPNO IN  
       (SELECT EMPNO  
        FROM DSN8C10.EMPPROJECT  
        WHERE PROJNO = 'MA2111');
```

Las subconsultas que varían en contenido de una fila a otra o de un grupo a otro son subconsultas *correlacionadas*. Para obtener información sobre subconsultas correlacionadas, consulte “[Subconsultas correlacionadas](#)” en la página 417.

Subconsultas y predicados

Un *predicado* es un elemento de una condición de búsqueda que especifica una condición que es verdadera, falsa o desconocida sobre una fila o grupo dado. Una subconsulta, que es una instrucción SELECT dentro de la cláusula WHERE o HAVING de otra instrucción SQL, siempre forma parte de un predicado. El predicado tiene la forma:

```
operand operator (subquery)
```

Una cláusula WHERE o HAVING puede incluir predicados que contengan subconsultas. Un predicado que contiene una subconsulta, como cualquier otro predicado de búsqueda, puede ir entre paréntesis, puede ir precedido de la palabra clave NOT y puede vincularse a otros predicados mediante las palabras clave AND y OR. Por ejemplo, la cláusula WHERE de una consulta puede parecerse a la siguiente cláusula:

```
WHERE X IN (subquery1) AND (Y > SOME (subquery2) OR Z IS NULL)
```

Las subconsultas también pueden aparecer en los predicados de otras subconsultas. Estas subconsultas son subconsultas anidadas en algún nivel de anidación. Por ejemplo, una subconsulta dentro de una subconsulta dentro de una SELECT externa tiene un nivel de anidamiento de 2. Db2 permite el anidamiento hasta un nivel de 15, pero pocas consultas requieren un nivel de anidamiento superior a 1.

La relación de una subconsulta con su SELECT externo es la misma que la relación de una subconsulta anidada con una subconsulta, y se aplican las mismas reglas, salvo que se indique lo contrario.

La tabla de resultados de la subconsulta

Una subconsulta debe producir una tabla de resultados que tenga el mismo número de columnas que el número de columnas del lado izquierdo del operador de comparación. Por ejemplo, las dos sentencias SELECT siguientes son aceptables:

```
SELECT EMPNO, LASTNAME  
  FROM DSN8C10.EMP  
 WHERE SALARY =  
       (SELECT AVG(SALARY)  
        FROM DSN8C10.EMP);
```

```
SELECT EMPNO, LASTNAME  
  FROM DSN8C10.EMP  
 WHERE (SALARY, BONUS) IN  
       (SELECT AVG(SALARY), AVG(BONUS)  
        FROM DSN8C10.EMP);
```

Excepto en el caso de una subconsulta de un predicado básico, la tabla de resultados puede contener más de una fila. Para obtener más información, consulte “[Lugares donde puede incluir una subconsulta](#)” en la página 415.

Conceptos relacionados

[Acceso a subconsultas \(Db2 Performance\)](#)

[Predicados \(Db2 SQL\)](#)

Tareas relacionadas

[Escritura de subconsultas eficientes \(Db2 Performance\)](#)

Referencia relacionada

[cláusula-where \(Db2 SQL\)](#)

[cláusula-having \(Db2 SQL\)](#)

Lugares donde puede incluir una subconsulta

Puede especificar una subconsulta en una cláusula WHERE o en una cláusula HAVING.

Puede especificar una subconsulta en una cláusula WHERE o HAVING utilizando uno de los siguientes elementos:

Ejemplo: predicado básico en una subconsulta

Puede utilizar una subconsulta inmediatamente después de cualquiera de los operadores de comparación. Si lo hace, la subconsulta puede devolver como máximo un valor. Db2 compara ese valor con el valor a la izquierda del operador de comparación.

La siguiente instrucción SQL devuelve los números de empleado, nombres y salarios de los empleados cuyo nivel de educación es superior al nivel de educación medio de toda la empresa.

```
SELECT EMPNO, LASTNAME, SALARY
  FROM DSN8C10.EMP
 WHERE EDLEVEL >
    (SELECT AVG(EDLEVEL)
      FROM DSN8C10.EMP);
```

Ejemplo: predicado cuantificado en una subconsulta: ALL, ANY o SOME

Puede utilizar una subconsulta después de un operador de comparación, seguido de la palabra clave ALL, ANY o SOME. El número de columnas y filas que la subconsulta puede devolver para un predicado cuantificado depende del tipo de predicado cuantificado:

- Para = ALGUNOS, = CUALQUIERA o <> TODOS, la subconsulta puede devolver una o varias filas y una o varias columnas. El número de columnas de la tabla de resultados debe coincidir con el número de columnas del lado izquierdo del operador.
- Para todos los demás predicados cuantificados, la subconsulta puede devolver una o varias filas, pero no más de una columna.

Consulte la información sobre predicados cuantificados, incluido qué hacer si una subconsulta que devuelve uno o más valores nulos le da resultados inesperados.

Ejemplo: TODOS los predicados

Utilice ALL para indicar que los operandos del lado izquierdo de la comparación deben compararse de la misma manera con **todos los valores** que devuelve la subconsulta. Por ejemplo, supongamos que utiliza el operador de comparación mayor que con ALL:

```
WHERE column > ALL (subquery)
```

Para satisfacer esta cláusula WHERE, el valor de la columna debe ser mayor que todos los valores que devuelve la subconsulta. Una subconsulta que devuelve una tabla de resultados vacía satisface el predicado.

Ahora suponga que utiliza el operador <> con ALL en una cláusula WHERE como esta:

```
WHERE (column1, column2, ... column) <> ALL (subquery)
```

Para satisfacer esta cláusula WHERE, cada valor de columna debe ser distinto de todos los valores de la columna correspondiente de la tabla de resultados que devuelve la subconsulta. Una subconsulta que devuelve una tabla de resultados vacía satisface el predicado.

Ejemplo: CUALQUIER o ALGUNO predicado

Utilice ANY o SOME para indicar que los valores del lado izquierdo del operador deben compararse de la manera indicada con **al menos uno** de los valores que devuelve la subconsulta. Por ejemplo, supongamos que utiliza el operador de comparación mayor que con ANY:

```
WHERE expression > ANY (subquery)
```

Para satisfacer esta cláusula WHERE, el valor de la expresión debe ser mayor que al menos uno de los valores (es decir, mayor que el valor más bajo) que devuelve la subconsulta. Una subconsulta que devuelve una tabla de resultados vacía no satisface el predicado.

Ahora suponga que utiliza el operador = con SOME en una cláusula WHERE como esta:

```
WHERE (column1, column1, ... columnn) = SOME (subquery)
```

Para satisfacer esta cláusula WHERE, cada valor de columna debe ser igual a al menos uno de los valores de la columna correspondiente de la tabla de resultados que devuelve la subconsulta. Una subconsulta que devuelve una tabla de resultados vacía no satisface el predicado.

Ejemplo: IN predicado en una subconsulta

Puede utilizar IN para indicar que el valor o los valores del lado izquierdo del operador IN deben estar entre los valores que devuelve la subconsulta. Usar IN equivale a usar = ANY o = SOME.

La siguiente consulta devuelve los nombres de los directores de departamento:

```
SELECT EMPNO, LASTNAME
  FROM DSN8C10.EMP
 WHERE EMPNO IN
      (SELECT DISTINCT MGRNO
        FROM DSN8C10.DEPT);
```

EXISTS predicado en una subconsulta

Cuando se utiliza la palabra clave EXISTS, Db2 comprueba si la subconsulta devuelve una o más filas. Devolver una o más filas cumple la condición; no devolver ninguna fila no cumple la condición.

La condición de búsqueda en la siguiente consulta se cumple si cualquier proyecto que esté representado en la tabla de proyectos tiene una fecha de inicio estimada posterior al 1 de enero de 2005:

```
SELECT EMPNO, LASTNAME
  FROM DSN8C10.EMP
 WHERE EXISTS
      (SELECT *
        FROM DSN8C10.PROJ
       WHERE PRSTDATE > '2005-01-01');
```

El resultado de la subconsulta es siempre el mismo para cada fila que se examina para la SELECT externa. Por lo tanto, o bien aparece cada fila en el resultado del SELECT externo o no aparece ninguna. Una subconsulta correlacionada es más potente que la subconsulta no correlacionada que se utiliza en este ejemplo porque el resultado de una subconsulta correlacionada se evalúa para cada fila del SELECT externo.

Como se muestra en el ejemplo, no es necesario especificar nombres de columna en la subconsulta de una cláusula EXISTS. En su lugar, puede codificar SELECT *. También puede utilizar la palabra clave EXISTS con la palabra clave NOT para seleccionar filas cuando los datos o la condición que especifique no existan; es decir, puede codificar la siguiente cláusula:

```
WHERE NOT EXISTS (SELECT ...);
```

Tareas relacionadas

[Escritura de subconsultas eficientes \(Db2 Performance\)](#)

Referencia relacionada

[Predicado cuantificado \(Db2 SQL\)](#)

[cláusula-having \(Db2 SQL\)](#)

[cláusula-where \(Db2 SQL\)](#)

[Predicado EXISTS \(Db2 SQL\)](#)

[Predicado IN \(Db2 SQL\)](#)

Subconsultas correlacionadas

Una subconsulta correlacionada es una subconsulta que Db2 revalúa cuando examina una nueva fila (en una cláusula WHERE) o un grupo de filas (en una cláusula HAVING) mientras ejecuta la sentencia SELECT externa.

En una subconsulta no correlacionada, Db2 ejecuta la subconsulta una vez, sustituye el resultado de la subconsulta en el lado derecho de la condición de búsqueda y evalúa el SELECT externo en función del valor de la condición de búsqueda.

Funciones definidas por el usuario en subconsultas correlacionadas

Tenga cuidado cuando invoque una función definida por el usuario en una subconsulta correlacionada, y esa función definida por el usuario utilice un bloc de notas. Db2 no actualiza el bloc de notas entre invocaciones de la subconsulta. Esto puede causar resultados no deseados porque el bloc de notas mantiene los valores a través de las invocaciones de la subconsulta.

Ejemplo de subconsulta correlacionada

Supongamos que desea una lista de todos los empleados cuyos niveles de educación son superiores a los niveles de educación medios en sus respectivos departamentos. Para obtener esta información, Db2 debe buscar en el DSN8C10.EMP tabla. Db2 , para cada empleado de la tabla, debe comparar el nivel de educación del empleado con el nivel de educación promedio del departamento de ese empleado.

Para este ejemplo, debe utilizar una subconsulta correlacionada, que difiere de una subconsulta no correlacionada. Una subconsulta no correlacionada compara el nivel de educación del empleado con el promedio de toda la empresa, lo que requiere examinar toda la tabla. Una subconsulta correlacionada evalúa solo el departamento que corresponde al empleado en particular.

En la subconsulta, le indicas a Db2 que calcule el nivel educativo promedio para el número de departamento en la fila actual. La siguiente consulta realiza esta acción:

```
SELECT EMPNO, LASTNAME, WORKDEPT, EDLEVEL
  FROM DSN8C10.EMP X
 WHERE EDLEVEL >
    (SELECT AVG(EDLEVEL)
      FROM DSN8C10.EMP
     WHERE WORKDEPT = X.WORKDEPT);
```

Una subconsulta correlacionada se parece a una no correlacionada, excepto por la presencia de una o más referencias correlacionadas. En el ejemplo, la única referencia correlacionada es la aparición de X.WORKDEPT en la cláusula WHERE de la subselección. En esta cláusula, el calificador X es el nombre de la correlación que se define en la cláusula FROM de la sentencia SELECT externa. X designa filas de la primera instancia de DSN8C10.EMP. En cualquier momento durante la ejecución de la consulta, X designa la fila de DSN8C10.EMP a la que se está aplicando la cláusula WHERE.

Considera lo que ocurre cuando la subconsulta se ejecuta para una fila determinada de DSN8C10.EMP. Antes de ejecutarse, X.WORKDEPT recibe el valor de la columna WORKDEPT para esa fila. Supongamos, por ejemplo, que la fila es para Christine Haas. Su departamento de trabajo es A00, que es el valor de WORKDEPT para esa fila. Por lo tanto, la siguiente es la subconsulta que se ejecuta para esa fila:

```
(SELECT AVG(EDLEVEL)
  FROM DSN8C10.EMP
 WHERE WORKDEPT = 'A00');
```

La subconsulta produce el nivel de educación promedio del departamento de Christine. El SELECT externo compara entonces este promedio con el nivel de estudios de Christine. Para cualquier otra fila para la que WORKDEPT tenga un valor diferente, ese valor aparece en la subconsulta en lugar de A00. Por ejemplo, en la fila de Michael L Thompson, este valor es B01, y la subconsulta de su fila ofrece el nivel medio de educación del departamento B01.

La tabla de resultados que se genera a partir de la consulta es similar a la siguiente salida:

EMPNO	LASTNAME	WORKDEPT	EDLEVEL
000010	HASS	A00	18
000030	KWAN	C01	20
000070	PULASKI	D21	16
000090	HENDERSON	E11	16

Conceptos relacionados

[Subconsultas correlacionadas y no correlacionadas \(Db2 Performance\)](#)

Referencia relacionada

[cláusula-having \(Db2 SQL\)](#)

[cláusula-where \(Db2 SQL\)](#)

Nombres de correlación en referencias

Un nombre de correlación es un nombre que se especifica para una tabla, vista, expresión de tabla anidada o función de tabla. Este nombre es válido solo dentro del contexto en el que se define. Utilice nombres de correlación para evitar ambigüedades, para establecer referencias correlacionadas, o utilizar nombres más cortos para tablas o vistas.

Una referencia correlacionada puede aparecer en una subconsulta, en una expresión de tabla anidada o como argumento de una función de tabla definida por el usuario. Para obtener información sobre referencias correlacionadas en expresiones de tablas anidadas y funciones de tabla, consulte “[Unión de datos desde más de una tabla](#)” en la página 396. En una subconsulta, la referencia debe tener la forma X.C, donde X es un nombre de correlación y C es el nombre de una columna de la tabla que X representa.

En una subconsulta puede aparecer cualquier número de referencias correlacionadas, sin restricciones de variedad. Por ejemplo, puede utilizar una referencia correlacionada en el SELECT externo y otra en una subconsulta anidada.

Cuando se utiliza una referencia correlacionada en una subconsulta, el nombre de la correlación puede definirse en la SELECT externa o en cualquiera de las subconsultas que contengan la referencia. Supongamos, por ejemplo, que una consulta contiene subconsultas A, B y C, y que A contiene B y B contiene C. La subconsulta C puede utilizar una referencia de correlación que se define en B, A o en la SELECT externa.

Puede definir un nombre de correlación para cada nombre de tabla en una cláusula FROM. Especifique el nombre de la correlación después del nombre de la tabla. Deje uno o más espacios en blanco entre el nombre de una tabla y su nombre de correlación. Puede incluir la palabra AS entre el nombre de la tabla y el nombre de la correlación para aumentar la legibilidad de la instrucción SQL.

El siguiente ejemplo muestra el uso de una referencia correlacionada en la condición de búsqueda de una subconsulta:

```
SELECT EMPNO, LASTNAME, WORKDEPT, EDLEVEL
  FROM DSN8C10.EMP AS X
 WHERE EDLEVEL >
    (SELECT AVG(EDLEVEL)
      FROM DSN8C10.EMP
     WHERE WORKDEPT = X.WORKDEPT);
```

El siguiente ejemplo muestra el uso de una referencia correlacionada en la lista de selección de una subconsulta:

```
UPDATE BP1TBL T1
  SET (KEY1, CHAR1, VCHAR1) =
    (SELECT VALUE(T2.KEY1,T1.KEY1), VALUE(T2.CHAR1,T1.CHAR1),
           VALUE(T2.VCHAR1,T1.VCHAR1)
```

```

    FROM BP2TBL T2
    WHERE (T2.KEY1 = T1.KEY1)
  WHERE KEY1 IN
    (SELECT KEY1
     FROM BP2TBL T3
     WHERE KEY2 > 0);

```

Uso de subconsultas correlacionadas en una instrucción UPDATE:

Utilice nombres de correlación en una instrucción UPDATE para referirse a las filas que está actualizando. La subconsulta para la que especificó un nombre de correlación se denomina *subconsulta correlacionada*.

Por ejemplo, cuando todas las actividades de un proyecto deben completarse antes de septiembre de 2006, su departamento considera que ese proyecto es un proyecto prioritario. Supongamos que ha añadido la columna PRIORITY a DSN8C10.PROJ. Puede utilizar la siguiente instrucción SQL para evaluar los proyectos en el DSN8C10.PROJ y escriba un 1 (una bandera para indicar PRIORIDAD) en la columna PRIORIDAD para cada proyecto prioritario:

```

UPDATE DSN8C10.PROJ X
SET PRIORITY = 1
WHERE DATE('2006-09-01') >
  (SELECT MAX(ACENDATE)
   FROM DSN8C10.PROJECT
   WHERE PROJNO = X.PROJNO);

```

Db2 , examina cada fila en el DSN8C10.PROJ table, determina la fecha de finalización máxima de la actividad (la columna ACENDATE) para todas las actividades del proyecto (desde la fecha de inicio de la actividad (DSN8C10).PROJECT tabla). Si la fecha de finalización de cada actividad asociada al proyecto es anterior a septiembre de 2006, la fila actual en el DSN8C10.PROJ table califica, y Db2 la actualiza.

Uso de subconsultas correlacionadas en una instrucción DELETE:

Utilice nombres de correlación en una instrucción DELETE para referirse a las filas que está eliminando. La subconsulta para la que especificó un nombre de correlación se denomina *subconsulta correlacionada*. Db2 evalúa la subconsulta correlacionada una vez por cada fila de la tabla que se nombra en la instrucción DELETE para decidir si se elimina la fila.

Uso de tablas sin restricciones referenciales:

Supongamos que un departamento considera que un proyecto está completo cuando la cantidad total de tiempo que se le dedica actualmente es menor o igual a la mitad del tiempo de una persona. A continuación, el departamento elimina las filas de ese proyecto de la base de datos de proyectos (DSN8C10).PROJ tabla. En los ejemplos de este tema, PROJ y PROJECT son tablas independientes; es decir, son tablas separadas sin restricciones referenciales definidas en ellas.

```

DELETE FROM DSN8C10.PROJ X
  WHERE .5 >
    (SELECT SUM(ACSTAFF)
     FROM DSN8C10.PROJECT
     WHERE PROJNO = X.PROJNO);

```

Para procesar esta declaración, Db2 determina para cada proyecto (representado por una fila en el archivo DSN8C10.PROJ tabla) si la plantilla combinada para ese proyecto es inferior a 0.5. Si es así, Db2 elimina esa fila de la base de datos DSN8C10.PROJ tabla.

Para continuar con este ejemplo, supongamos que Db2 elimina una fila en el archivo DSN8C10.PROJ tabla. También debe eliminar las filas que estén relacionadas con el proyecto eliminado en el DSN8C10.PROJECT tabla. Para ello, utilice una declaración similar a esta:

```

DELETE FROM DSN8C10.PROJECT X
  WHERE NOT EXISTS
    (SELECT *
     FROM DSN8C10.PROJ
     WHERE PROJNO = X.PROJNO);

```

Db2 determina, para cada fila en el DSN8C10.PROJECT tabla, si existe una fila con el mismo número de proyecto en el DSN8C10.PROJ tabla. Si no, Db2 elimina la fila de DSN8C10.PROJECT.

Uso de una sola tabla:

Una subconsulta de una sentencia DELETE buscada (una sentencia DELETE que no utiliza un cursor) puede hacer referencia a la misma tabla de la que se eliminan las filas. En la siguiente sentencia, que elimina al empleado con el salario más alto de cada departamento, la tabla de empleados aparece en el DELETE externo y en la subselección:

```
DELETE FROM YEMP X  
WHERE SALARY = (SELECT MAX(SALARY) FROM YEMP Y  
WHERE X.WORKDEPT = Y.WORKDEPT);
```

Este ejemplo utiliza una copia de la tabla de empleados para la subconsulta.

La siguiente declaración, sin una subconsulta correlacionada, produce resultados equivalentes:

```
DELETE FROM YEMP  
WHERE (SALARY, WORKDEPT) IN (SELECT MAX(SALARY), WORKDEPT  
FROM YEMP  
GROUP BY WORKDEPT);
```

Uso de tablas con restricciones referenciales:

Db2 restringe las operaciones de eliminación para tablas dependientes que están involucradas en restricciones referenciales. Si una instrucción DELETE tiene una subconsulta que hace referencia a una tabla que está implicada en la eliminación, haga que la última regla de eliminación en la ruta a esa tabla sea RESTRICT o NO ACTION. Esta acción garantiza que el resultado de la subconsulta no se materialice antes de que se produzca la eliminación. Sin embargo, si el resultado de la subconsulta se materializa antes de la eliminación, la regla de eliminación también puede ser CASCADE o SET NULL.

Ejemplo : Sin restricciones referenciales, la siguiente instrucción elimina de la tabla de departamentos aquellos cuyos responsables no figuran correctamente en la tabla de empleados:

```
DELETE FROM DSN8C10.DEPT THIS  
WHERE NOT DEPTNO =  
(SELECT WORKDEPT  
FROM DSN8C10.EMP  
WHERE EMPNO = THIS.MGRNO);
```

Con las restricciones referenciales que se definen para las tablas de muestra, esta sentencia provoca un error porque la tabla de resultados de la subconsulta no se materializa antes de que se produzca la eliminación. Debido a que DSN8C10.EMP es una tabla dependiente de DSN8C10.DEPT, la eliminación implica la tabla a la que se hace referencia en la subconsulta, y la última regla de eliminación en la ruta a EMP es SET NULL, no RESTRICT o NO ACTION. Si la sentencia pudiera ejecutarse, sus resultados dependerían del orden en que Db2 o accede a las filas. Por lo tanto, Db2 prohíbe la eliminación.

Restricciones de la utilización de tipos diferenciados con UNION, EXCEPT e INTERSECT

Db2 impone la tipificación firme de tipos distinct con UNION, EXCEPT y INTERSECT. Cuando utiliza estas palabras clave para combinar los valores de columna desde varias tablas, las columnas combinadas deben ser de los mismos tipos. Si una columna tiene un tipo diferenciado, la columna correspondiente debe ser del mismo tipo diferenciado.

Ejemplo : Supongamos que crea una vista que combina los valores de las tablas US_SALES, EUROPEAN_SALES y JAPAN_SALES. Las columnas TOTAL de las tres tablas son de diferentes tipos distintos. Antes de combinar los valores de la tabla, debe convertir los tipos de dos de las columnas TOTAL al tipo de la tercera columna TOTAL. Supongamos que se ha elegido el tipo US_DOLLAR como tipo común distinto. Debido a que Db2 no genera funciones de conversión para pasar de un tipo distinto a otro, deben existir dos funciones definidas por el usuario:

- Una función llamada EURO_TO_US que convierte valores de tipo EURO a tipo US_DOLLAR
- Una función llamada YEN_TO_US que convierte valores de tipo JAPANESE_YEN a tipo US_DOLLAR

A continuación, puede ejecutar una consulta como esta para mostrar una tabla de ventas combinadas:

```
SELECT PRODUCT_ITEM, MONTH, YEAR, TOTAL  
FROM US_SALES  
UNION  
SELECT PRODUCT_ITEM, MONTH, YEAR, EURO_TO_US(TOTAL)  
FROM EUROPEAN_SALES  
UNION  
SELECT PRODUCT_ITEM, MONTH, YEAR, YEN_TO_US(TOTAL)  
FROM JAPAN_SALES;
```

Dado que el tipo de resultado tanto de la función YEN_TO_US como de la función EURO_TO_US es US_DOLLAR, se cumple el requisito de que los tipos distintos de las columnas combinadas sean iguales.

Comparación de tipos distintos

Solo puede comparar un objeto con un tipo distinto con otro objeto que tenga exactamente el mismo tipo distinto. No se pueden comparar datos de un tipo distinto directamente con datos de su tipo de origen. Sin embargo, puede comparar un tipo distinto con su tipo de origen utilizando una función de conversión.

La regla básica para las comparaciones es que los tipos de datos de los operandos deben ser compatibles. La regla de compatibilidad define, por ejemplo, que todos los tipos numéricos (SMALLINT, INTEGER, FLOAT y DECIMAL) son compatibles. Es decir, puede comparar un valor INTEGER con un valor de tipo FLOAT. Sin embargo, no se puede comparar un objeto de un tipo distinto con un objeto de un tipo diferente. Solo puede comparar un objeto con un tipo distinto con otro objeto que tenga exactamente el mismo tipo distinto.

Por ejemplo, supongamos que desea saber qué productos se vendieron por más de 100 \$ 000.00 en EE. UU. en el mes de julio de 2003 (7/03). Debido a que no se pueden comparar datos del tipo US_DOLLAR con instancias de datos del tipo de origen US_DOLLAR (DECIMAL) directamente, debe utilizar una función de conversión para convertir datos de DECIMAL a US_DOLLAR o de US_DOLLAR a DECIMAL. Db2, siempre que se crea un tipo distinto, crea dos funciones de conversión, una para convertir del tipo de origen al tipo distinto y la otra para convertir del tipo distinto al tipo de origen. Para el tipo distinto US_DOLLAR, Db2 crea una función de conversión llamada DECIMAL y una función de conversión llamada US_DOLLAR. Cuando se compara un objeto de tipo US_DOLLAR con un objeto de tipo DECIMAL, se puede utilizar una de esas funciones de conversión para que los tipos de datos sean idénticos para la comparación. Supongamos que la tabla US_SALES se define así:

```
CREATE TABLE US_SALES  
(PRODUCT_ITEM  INTEGER,  
 MONTH        INTEGER CHECK (MONTH BETWEEN 1 AND 12),  
 YEAR         INTEGER CHECK (YEAR > 1990),  
 TOTAL        US_DOLLAR);
```

Entonces, puede convertir datos DECIMALES a US_DOLLAR de esta manera:

```
SELECT PRODUCT_ITEM  
  FROM US_SALES  
 WHERE TOTAL > US_DOLLAR(100000.00)  
   AND MONTH = 7  
   AND YEAR  = 2003;
```

La conversión cumple el requisito de que los tipos de datos comparados sean idénticos.

No puede utilizar variables de host en sentencias que prepare para ejecución dinámica. Como se explica en “[Ejecución dinámica de una sentencia SQL utilizando PREPARE y EXECUTE](#)” en la página 549, puede sustituir los marcadores de parámetros por variables de host cuando prepare una sentencia y, a continuación, utilizar variables de host cuando ejecute la sentencia.

Si utiliza un marcador de parámetro en un predicado de una consulta y la columna con la que compara el valor representado por el marcador de parámetro es de un tipo distinto, debe convertir el marcador de parámetro al tipo distinto o convertir la columna a su tipo de origen.

Por ejemplo, supongamos que el tipo distinto CNUM se define así:

```
CREATE DISTINCT TYPE CNUM AS INTEGER;
```

La tabla CLIENTE se define de la siguiente manera:

```
CREATE TABLE CUSTOMER
(CUST_NUM      CNUM NOT NULL,
 FIRST_NAME    CHAR(30) NOT NULL,
 LAST_NAME     CHAR(30) NOT NULL,
 PHONE_NUM     CHAR(20) WITH DEFAULT,
 PRIMARY KEY   (CUST_NUM));
```

En un programa de aplicación, usted prepara una instrucción SELECT que compara la columna CUST_NUM con un marcador de parámetro. Debido a que CUST_NUM es de un tipo distinto, debe convertir el tipo distinto a su tipo de origen:

```
SELECT FIRST_NAME, LAST_NAME, PHONE_NUM FROM CUSTOMER
WHERE CAST(CUST_NUM AS INTEGER) = ?
```

Alternativamente, puede convertir el marcador de parámetro al tipo distinto:

```
SELECT FIRST_NAME, LAST_NAME, PHONE_NUM FROM CUSTOMER
WHERE CUST_NUM = CAST (? AS CNUM)
```

Instrucciones SQL anidadas

Una instrucción SQL puede invocar explícitamente funciones definidas por el usuario o procedimientos almacenados, o puede activar implícitamente desencadenadores que invocan funciones definidas por el usuario o procedimientos almacenados. Esta situación se conoce como *sentencias SQL anidadas*.

Db2 admite hasta 64 niveles de sentencias SQL anidadadas.

Restricciones para sentencias SQL anidadadas

Tenga en cuenta las siguientes restricciones de Db2 en sentencias SQL anidadadas:

- Restricciones para las sentencias SELECT:

Cuando ejecutas una instrucción SELECT en una tabla, no puedes ejecutar instrucciones INSERT, UPDATE, MERGE o DELETE en la misma tabla en un nivel inferior de anidamiento.

Por ejemplo, supongamos que ejecuta esta instrucción SQL en el nivel 1 de anidamiento:

```
SELECT UDF1(C1) FROM T1;
```

No puede ejecutar esta instrucción SQL en un nivel de anidamiento inferior:

```
INSERT INTO T1 VALUES(...);
```

- Restricciones para las sentencias SELECT FROM FINAL TABLE que especifican sentencias INSERT, UPDATE o DELETE para cambiar datos:

Cuando ejecutas este tipo de declaración, se produce un error si se dan las dos condiciones siguientes:

- La sentencia SELECT que modifica datos (especificando INSERT, UPDATE o DELETE) activa un AFTER TRIGGER.
- El AFTER TRIGGER da como resultado operaciones SQL anidadas adicionales que modifican la tabla que es el objetivo de la instrucción SELECT original que modifica los datos.

- Restricciones para las instrucciones INSERT, UPDATE, MERGE y DELETE:

Cuando ejecutas una instrucción INSERT, UPDATE, MERGE o DELETE en una tabla, no puedes acceder a esa tabla desde una función definida por el usuario o un procedimiento almacenado que se encuentre en un nivel inferior de anidamiento.

Por ejemplo, supongamos que ejecuta esta instrucción SQL en el nivel 1 de anidamiento:

```
DELETE FROM T1 WHERE UDF3(T1.C1) = 3;
```

No puede ejecutar esta instrucción SELECT en un nivel de anidamiento inferior:

```
SELECT * FROM T1;
```

Anidación de sentencias para desencadenadores AFTER

Si el desencadenador AFTER no se activa mediante una instrucción de cambio de datos INSERT, UPDATE o DELETE que se especifica en una referencia de tabla de cambio de datos SELECT FROM FINAL TABLE, la lista anterior de restricciones no se aplica a las instrucciones SQL que se ejecutan en un nivel inferior de anidamiento como resultado de un desencadenador after. Por ejemplo, supongamos que una sentencia UPDATE en el nivel de anidamiento 1 activa un desencadenador after update, que llama a un procedimiento almacenado. El procedimiento almacenado ejecuta dos sentencias SQL que hacen referencia a la tabla desencadenante: una sentencia SELECT y una sentencia INSERT. En esta situación, se pueden ejecutar tanto la instrucción SELECT como la instrucción INSERT, aunque se encuentren en el nivel de anidamiento 3.

Aunque las activaciones de desencadenadores cuentan en los niveles de anidamiento de sentencias SQL, las restricciones anteriores sobre sentencias SQL no se aplican a las sentencias SQL que se ejecutan en el cuerpo del desencadenador.

Por ejemplo, supongamos que el desencadenador TR1 está definido en la tabla T1:

```
CREATE TRIGGER TR1
AFTER INSERT ON T1
FOR EACH STATEMENT MODE DB2SQL
BEGIN ATOMIC
  UPDATE T1 SET C1=1;
END
```

Ahora suponga que ejecuta esta instrucción SQL en el nivel 1 de anidamiento:

```
INSERT INTO T1 VALUES(...);
```

Aunque la sentencia UPDATE en el cuerpo desencadenante está en el nivel 2 de anidamiento y modifica la misma tabla que la sentencia desencadenante actualiza, Db2 puede ejecutar la sentencia INSERT correctamente.

Recuperación de un conjunto de filas utilizando un cursor

En un programa de aplicación, puede recuperar un conjunto de filas de una tabla o una tabla de resultados que devuelve el procedimiento almacenado. Puede recuperar una o más filas a la vez.

Acerca de esta tarea

Utilice cualquiera de los siguientes tipos de cursos para recuperar filas de una tabla de resultados:

- Un cursor posicionado en fila recupera como máximo una sola fila a la vez de la tabla de resultados en variables de host. En cualquier momento, el cursor se posiciona como máximo en una sola fila. Para obtener información sobre cómo utilizar un cursor de posición de fila, consulte [“Acceso a datos mediante un cursor posicionado en una fila” en la página 428](#).
- Un cursor posicionado en un conjunto de filas recupera cero, una o más filas a la vez, como un conjunto de filas, de la tabla de resultados en matrices de variables de host. En cualquier momento, el cursor puede colocarse en un conjunto de filas. Puede hacer referencia a todas las filas del conjunto de filas, o solo a una fila del conjunto de filas, cuando utiliza una instrucción DELETE o UPDATE posicionada. Para obtener información sobre cómo utilizar un cursor posicionado en una fila, consulte [“Acceso a datos utilizando un cursor colocado por conjunto de filas” en la página 433](#).

Cursos

Un cursor es un mecanismo que señala una o más filas en un conjunto de filas. Las filas se recuperan de una tabla o de un conjunto de resultados devuelto por un procedimiento almacenado. El programa de aplicación puede utilizar un cursor para recuperar filas de una tabla.

Acerca de esta tarea

Los cursos vinculados con estabilidad de cursor que se utilizan en operaciones de obtención de bloques son especialmente vulnerables a la lectura de datos que ya han cambiado. En una obtención en bloque, el acceso a la base de datos obtiene previamente filas antes de la recuperación de filas controlada por la aplicación. Durante ese tiempo, el cursor podría cerrarse y los bloqueos podrían liberarse antes de que la aplicación reciba los datos. Por lo tanto, es posible que la aplicación recupere una fila de valores que ya no existe o que no encuentre una fila insertada recientemente. En muchos casos, eso es aceptable; un caso en el que **no es aceptable** se dice que requiere *la actualización de los datos*.

Si su aplicación requiere la actualización de datos para un cursor, debe evitar la obtención de bloques para los datos a los que apunta. Para evitar la obtención de bloques para un cursor distribuido, declare el cursor con la cláusula FOR UPDATE.

Tipos de cursos

Puede declarar cursos colocados por fila o por conjunto de filas de varias formas. Estos cursos pueden ser desplazables o no desplazables, retenidos o no retenidos, o retornables o no retornables.

Además, puede declarar un cursor retornable en un procedimiento almacenado incluyendo la cláusula WITH RETURN; el cursor puede devolver conjuntos de resultados a un llamador del procedimiento almacenado.

Cursos desplazables y no desplazables:

Cuando declara un cursor, le indica a Db2 si desea que el cursor se pueda desplazar o no, incluyendo u omitiendo la cláusula SCROLL. Esta cláusula determina si el cursor se mueve secuencialmente hacia adelante a través de la tabla de resultados o si puede moverse aleatoriamente a través de la tabla de resultados.

Uso de un cursor no desplazable:

El tipo más simple de cursor es un cursor no desplazable. Un cursor no desplazable puede estar posicionado en una fila o en un conjunto de filas. Un cursor no desplazable colocado en fila avanza por su tabla de resultados una fila cada vez. De manera similar, un cursor no desplazable posicionado en un conjunto de filas se mueve hacia adelante a través de su tabla de resultados un conjunto de filas a la vez.

Un cursor no desplazable siempre se mueve secuencialmente hacia adelante en la tabla de resultados. Cuando la aplicación abre el cursor, este se coloca antes de la primera fila (o primer conjunto de filas) de la tabla de resultados. Cuando la aplicación ejecuta el primer FETCH, el cursor se posiciona en la primera fila (o primer conjunto de filas). Cuando la aplicación ejecuta las siguientes sentencias FETCH, el cursor avanza una fila (o un conjunto de filas) por cada FETCH. Despues de cada sentencia FETCH, el cursor se posiciona en la fila (o conjunto de filas) que se ha obtenido.

Después de que la aplicación ejecute una instrucción UPDATE o DELETE posicionada, el cursor permanece en la fila actual (o conjunto de filas) de la tabla de resultados. No puede recuperar filas (o conjuntos de filas) hacia atrás ni moverse a una posición específica en una tabla de resultados con un cursor no desplazable.

Uso de un cursor desplazable:

Para que un cursor se pueda desplazar, debe declararlo como desplazable. Un cursor desplazable puede posicionarse en una fila o en un conjunto de filas. Para utilizar un cursor desplazable, ejecute sentencias FETCH que indiquen dónde desea colocar el cursor.

Si desea ordenar las filas del conjunto de resultados del cursor y también desea que el cursor se pueda actualizar, debe declarar el cursor como desplazable, incluso si lo utiliza solo para recuperar filas (o conjuntos de filas) de forma secuencial. Puede utilizar la cláusula ORDER BY en la declaración de un cursor actualizable solo si declara el cursor como desplazable.

Declaración de un cursor desplazable:

Para indicar que un cursor es de desplazamiento bidireccional, déclárelo con la palabra clave SCROLL. Los siguientes ejemplos muestran una característica de los cursores desplazables: *la sensibilidad*.

La siguiente figura muestra una declaración para un cursor desplazable insensible.

```
EXEC SQL DECLARE C1 INSENSITIVE SCROLL CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO
  FROM DSN8C10.DEPT
  ORDER BY DEPTNO
END-EXEC.
```

Declarar un cursor desplazable con la palabra clave INSENSITIVE tiene los siguientes efectos:

- El tamaño, el orden de las filas y los valores de cada fila de la tabla de resultados no cambian después de que la aplicación abra el cursor.

Las filas que se insertan en la tabla subyacente no se añaden a la tabla de resultados.

- La tabla de resultados es de solo lectura. Por lo tanto, no puede declarar el cursor con la cláusula FOR UPDATE, y no puede utilizar el cursor para operaciones de actualización o eliminación posicionadas.

La siguiente figura muestra una declaración para un cursor sensible estático desplazable.

```
EXEC SQL DECLARE C2 SENSITIVE STATIC SCROLL CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO
  FROM DSN8C10.DEPT
  ORDER BY DEPTNO
END-EXEC.
```

Declarar un cursor como ESTÁTICO SENSIBLE tiene los siguientes efectos:

- El tamaño de la tabla de resultados no aumenta después de que la aplicación abre el cursor.

Las filas que se insertan en la tabla subyacente no se añaden a la tabla de resultados.

- El orden de las filas no cambia después de que la aplicación abra el cursor.

Si la declaración del cursor contiene una cláusula ORDER BY y las columnas que están en la cláusula ORDER BY se actualizan después de que se abra el cursor, el orden de las filas en la tabla de resultados no cambia.

- Cuando la aplicación ejecuta instrucciones UPDATE y DELETE posicionadas con el cursor, esos cambios son visibles en la tabla de resultados.
- Cuando el valor actual de una fila ya no satisface la instrucción SELECT que se utilizó en la declaración del cursor, esa fila ya no es visible en la tabla de resultados.
- Cuando se elimina una fila de la tabla de resultados de la tabla subyacente, esa fila ya no está visible en la tabla de resultados.
- Los cambios que otros cursores u otros procesos de aplicación realizan en la tabla subyacente pueden ser visibles en la tabla de resultados, dependiendo de si las sentencias FETCH que utiliza con el cursor son sentencias FETCH INSENSITIVE o FETCH SENSITIVE.

La siguiente figura muestra una declaración para un cursor sensible y dinámico.

```
EXEC SQL DECLARE C2 SENSITIVE DYNAMIC SCROLL CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO
  FROM DSN8C10.DEPT
  ORDER BY DEPTNO
END-EXEC.
```

Declarar un cursor como SENSITIVE DYNAMIC tiene los siguientes efectos:

- El tamaño y el contenido de la tabla de resultados pueden cambiar con cada consulta.

La tabla base puede cambiar mientras el cursor se desplaza sobre ella. Si otro proceso de solicitud cambia los datos, el cursor ve los datos recién modificados cuando se confirma. Si el proceso de aplicación del cursor cambia los datos, el cursor ve los datos recién cambiados inmediatamente.

- El orden de las filas puede cambiar después de que la aplicación abra el cursor.

Si la sentencia SELECT de la declaración del cursor contiene una cláusula ORDER BY, y las columnas que están en la cláusula ORDER BY se actualizan después de que se abra el cursor, el orden de las filas en la tabla de resultados cambia.

- Cuando la aplicación ejecuta instrucciones UPDATE y DELETE posicionadas con el cursor, esos cambios son visibles. Además, cuando la aplicación ejecuta operaciones de inserción, actualización o eliminación (dentro de la aplicación pero fuera del cursor), esos cambios son visibles.
- Todas las inserciones, actualizaciones y eliminaciones comprometidas por otros procesos de aplicación son visibles.
- Debido a que la sentencia FETCH se ejecuta contra la tabla base, el cursor no necesita una tabla de resultados temporal. Cuando se define un cursor como SENSITIVE DYNAMIC, no se puede especificar la palabra clave INSENSITIVE en una sentencia FETCH para ese cursor.

Visibilidad de los cambios en una tabla de resultados:

Que un cursor pueda ver sus propios cambios o los cambios que se hacen en los datos por otros procesos o curosres depende de cómo se declare el cursor y de la capacidad de actualización del cursor. La visibilidad también depende del tipo de operación de búsqueda que se ejecute con el cursor. La siguiente tabla resume la visibilidad de los cambios en una tabla de resultados para cada tipo de cursor.

Tipo de cursor declarado	¿El cursor es actualizable o de solo lectura?	¿Los cambios realizados con el cursor son visibles en la tabla de resultados? ^{“3” en la página 427}	¿Los cambios realizados por otros curosres o procesos son visibles en la tabla de resultados?
SIN DESPLAZAMIENTO (la tabla de resultados está materializada)	Solo lectura ^{“1” en la página 427}	No aplicable	Nee
SIN DESPLAZAMIENTO (la tabla de resultados no se materializa)	Actualizable ^{“2” en la página 427}	Sí	Sí
INSENSITIVE SCROLL	Solo lectura ^{“4” en la página 427}	No aplicable	Nee
SENSITIVE STATIC SCROLL	Actualizable ^{“2” en la página 427, “6” en la página 427}	Sí	Depende de la sensibilidad explícita o implícita especificada en la cláusula FETCH ^{“5” en la página 427}
DESPLAZAMIENTO DINÁMICO SENSIBLE	Actualizable ^{“2” en la página 427}	Sí	Si ^{“7” en la página 427}

Tipo de cursor declarado	¿El cursor es actualizable o de solo lectura?	¿Los cambios realizados con el cursor son visibles en la tabla de resultados? ^{“3” en la página 427}	¿Los cambios realizados por otros cursos o procesos son visibles en la tabla de resultados?
---------------------------------	--	--	--

Notas:

1. El contenido de la instrucción SELECT del cursor hace que el cursor sea implícitamente de solo lectura.
2. El cursor solo se puede actualizar si NO SE ESPECIFICA PARA SOLO LECTURA o PARA SOLO RECUPERACIÓN como parte de la instrucción SELECT del cursor, y no hay nada en el contenido de la instrucción SELECT que haga que el cursor sea implícitamente de solo lectura.
3. Si se especifica INSENSITIVE en FETCH, solo serán visibles los cambios realizados por el mismo cursor, suponiendo que las filas que se están recuperando no hayan sido ya leídas por un SENSITIVE FETCH en el mismo cursor.
4. Un cursor INSENSIBLE es de solo lectura si no se especifica una cláusula de actualizabilidad.
5. La cláusula de sensibilidad en una sentencia FETCH afecta a la visibilidad de los cambios de otros de la siguiente manera:
 - Para FETCH INSENSITIVE: Solo son visibles las actualizaciones y eliminaciones realizadas con el mismo cursor.
 - Para FETCH SENSITIVE: Todas las actualizaciones y eliminaciones son visibles.
6. Las actualizaciones y eliminaciones posicionadas no están permitidas si los valores de las columnas seleccionadas no coinciden con los valores actuales de las columnas de la tabla base, incluso si la fila satisface el predicado de la instrucción SELECT del cursor.
7. Todas las actualizaciones y eliminaciones que se realizan con este cursor, y los cambios confirmados que se realizan con otros procesos, son visibles en las siguientes sentencias FETCH. Los insertos que se realizan mediante este proceso también son visibles a medida que se desplaza la tabla de resultados. Las inserciones por otros procesos en las tablas base subyacentes a la tabla de resultados son visibles después de que se confirmen.

Conceptos relacionados

[Interacción de la sentencia FETCH entre el posicionamiento de fila y conjunto de filas](#)

Cuando declara un cursor con la cláusula WITH ROWSET POSITIONING, puede entremezclar sentencias FETCH colocadas por fila con sentencias FETCH colocadas por conjunto de filas.

[Comparación de cursos desplazables](#)

Si un cursor desplazable puede ver los cambios que realizan otros procesadores o cursos en los datos depende de cómo se declare el cursor. También depende del tipo de operación de captación que se ejecuta.

Cursos retenidos y no retenidos

Un cursor retenido no se cierra después de una operación de confirmación. Un cursor no retenido se cierra después de una operación de confirmación. Especifica si desea que un cursor esté retenido o no incluyendo u omitiendo la cláusula WITH HOLD cuando declara el cursor.

Después de una operación de confirmación, la posición de un cursor retenido depende de su tipo:

- Un cursor no desplazable que se mantiene se coloca después de la última fila recuperada y antes de la siguiente fila lógica. La siguiente fila se puede devolver desde la tabla de resultados con una instrucción FETCH NEXT.
- Un cursor estático desplazable que se mantiene se coloca en la última fila recuperada. La última fila recuperada puede devolverse desde la tabla de resultados con una sentencia FETCH CURRENT.
- Un cursor dinámico desplazable que se mantiene se coloca después de la última fila recuperada y antes de la siguiente fila lógica. Utilice una sentencia FETCH para reposicionar el cursor y recuperar la fila o

conjunto de filas deseado. Db2 devuelve SQLCODE +231 para una sentencia FETCH que especifica la palabra clave CURRENT para una obtención de una sola fila.

Un cursor detenido puede cerrarse cuando:

- Usted emite una instrucción CLOSE cursor, ROLLBACK o CONNECT
- Usted emite una llamada a la función CAF CLOSE o una llamada a la función RRSAF TERMINATE THREAD
- El programa de aplicación finaliza.

Si el programa se cierra de forma anormal, se pierde la posición del cursor. Para prepararse para el reinicio, el programa debe reposicionar el cursor.

Las siguientes restricciones se aplican a los cursosres que se declaran CON RETENCIÓN:

- No utilice DECLARE CURSOR WITH HOLD con el nuevo inicio de sesión de usuario desde un centro de archivos adjuntos de Db2 , porque todos los cursosres abiertos se cierran.
- No declare un cursor CON RETENCIÓN en un hilo que pueda quedar inactivo. Si lo hace, sus bloqueos se mantienen indefinidamente.

IMS

No puede utilizar DECLARE CURSOR...CON RETENCIÓN en los programas de procesamiento de mensajes (MPP) y en el procesamiento de mensajes por lotes (BMP) basado en mensajes. Cada mensaje es un nuevo usuario para Db2; tanto si los declara o no con WITH HOLD, no continúan los cursosres para los nuevos usuarios. Puede utilizar WITH HOLD en programas por lotes BMP y DL/I no basados en mensajes.

CICS

En CICS aplicaciones, puede utilizar DECLARE CURSOR...CON RETENCIÓN para indicar que un cursor no debe cerrarse en un punto de confirmación o sincronización. Sin embargo, SYNCPOINT ROLLBACK cierra todos los cursosres, y el fin de tarea (EOT) cierra todos los cursosres antes de que Db2 reutilice o finalice el hilo. Debido a que las transacciones pseudoconversacionales suelen tener múltiples sentencias EXEC CICS RETURN y, por lo tanto, abarcan múltiples EOT, el alcance de un cursor retenido es limitado. En todos los EOT, debe volver a abrir y reposicionar un cursor declarado CON RETENCIÓN, como si no hubiera especificado CON RETENCIÓN.

Siempre debe cerrar los cursosres que ya no necesite. Si deja que Db2 cierre un CICS, es posible que el cursor no se cierre hasta que el CICS función de adjuntar reutiliza o finaliza el hilo.

Si la CICS aplicación utiliza un subprocesso de entrada protegido, este subprocesso seguirá conteniendo recursos, incluso cuando finalice la tarea que ha utilizado estos recursos. Estos recursos no se publicarán hasta que finalice el hilo protegido.

La siguiente declaración de cursor hace que el cursor mantenga su posición en la tabla DSN8C10.EMP después de un punto de confirmación:

```
EXEC SQL
  DECLARE EMPLUPDT CURSOR WITH HOLD FOR
    SELECT EMPNO, LASTNAME, PHONENO, JOB, SALARY, WORKDEPT
      FROM DSN8C10.EMP
        WHERE WORKDEPT < 'D11'
          ORDER BY EMPNO
  END-EXEC.
```

Acceso a datos mediante un cursor posicionado en una fila

Un cursor de posición de fila es un cursor que apunta a una sola fila y recupera como máximo una sola fila a la vez de la tabla de resultados. Puede especificar una solicitud de recuperación para indicar qué filas recuperar, en relación con la posición actual del cursor.

Procedimiento

Para acceder a los datos utilizando un cursor posicionado en fila:

1. Ejecutar una instrucción DECLARE CURSOR para definir la tabla de resultados en la que opera el cursor. Consulte “[Declarar un cursor de fila](#)” en la página 429.
2. Ejecutar un CURSOR ABIERTO para que el cursor esté disponible para la aplicación. Consulte “[Apertura de un cursor de fila](#)” en la página 431.
3. Especifique lo que el programa debe hacer cuando se hayan recuperado todas las filas. Consulte “[Especificación de la acción que debe realizar el cursor de fila cuando llegue al final de los datos](#)” en la página 431.
4. Ejecutar varias sentencias SQL para recuperar datos de la tabla o modificar filas seleccionadas de la tabla. Consulte “[Ejecución de sentencias SQL utilizando un cursor de fila](#)” en la página 431.
5. Ejecutar una instrucción CLOSE CURSOR para que el cursor no esté disponible para la aplicación. Consulte “[Cerrar un cursor de fila](#)” en la página 433.

Resultados

Su programa puede tener varios cursos, cada uno de los cuales realiza los pasos anteriores.

Declarar un cursor de fila

Antes de poder utilizar un cursor posicionado en fila para recuperar filas, debe declarar el cursor. Cuando declara un cursor, identifica un conjunto de filas a las que se debe acceder con el cursor.

Procedimiento

Para declarar un cursor de fila, emita una instrucción DECLARE CURSOR.

La sentencia DECLARE CURSOR nombría un cursor y especifica una sentencia SELECT. La sentencia SELECT define los criterios para las filas que van a componer la tabla de resultados.

El siguiente ejemplo muestra una forma sencilla de la instrucción DECLARE CURSOR:

```
EXEC SQL
DECLARE C1 CURSOR FOR
  SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY
    FROM DSN8C10.EMP
  END-EXEC.
```

Puede utilizar este cursor para enumerar información seleccionada sobre los empleados.

Los cursos más complicados pueden incluir cláusulas WHERE o uniones de varias tablas. Por ejemplo, supongamos que desea utilizar un cursor para enumerar a los empleados que trabajan en un determinado proyecto. Declare un cursor como este para identificar a esos empleados:

```
EXEC SQL
DECLARE C2 CURSOR FOR
  SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY
    FROM DSN8C10.EMP X
    WHERE EXISTS
      (SELECT *
        FROM DSN8C10.PROJ Y
        WHERE X.EMPNO=Y.RESPEMP
        AND Y.PROJNO=:GOODPROJ);
```

Declaración de cursos para tablas que utilizan seguridad multinivel

Puede declarar un cursor que recupere filas de una tabla que utilice seguridad multinivel con granularidad a nivel de fila. Sin embargo, la tabla de resultados para el cursor contiene solo aquellas filas que tienen un valor de etiqueta de seguridad que es equivalente o está dominado por el valor de etiqueta de seguridad de su ID.

Actualizar una columna

Puede actualizar las columnas en las filas que recupera. La actualización de una fila después de utilizar un cursor para recuperarla se denomina actualización *posicionada*. Si tiene la intención de

realizar actualizaciones posicionadas en la tabla identificada, incluya la cláusula FOR UPDATE. La cláusula PARA ACTUALIZAR tiene dos formas:

- El primer formulario es PARA ACTUALIZAR *LA LISTA DE COLUMNAS*. Utilice este formulario cuando sepa de antemano qué columnas necesita actualizar.
- El segundo formulario es PARA ACTUALIZAR, sin lista de columnas. Utilice este formulario cuando pueda usar el cursor para actualizar cualquiera de las columnas de la tabla.

Por ejemplo, puede utilizar este cursor para actualizar solo la columna SALARIO de la tabla de empleados:

```
EXEC SQL
DECLARE C1 CURSOR FOR
  SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY
    FROM DSN8C10.EMP X
   WHERE EXISTS
     (SELECT *
       FROM DSN8C10.PROJ Y
      WHERE X.EMPNO=Y.RESPEMP
        AND Y.PROJNO=:GOODPROJ)
FOR UPDATE OF SALARY;
```

Si puede utilizar el cursor para actualizar cualquier columna de la tabla de empleados, defina el cursor de la siguiente manera:

```
EXEC SQL
DECLARE C1 CURSOR FOR
  SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY
    FROM DSN8C10.EMP X
   WHERE EXISTS
     (SELECT *
       FROM DSN8C10.PROJ Y
      WHERE X.EMPNO=Y.RESPEMP
        AND Y.PROJNO=:GOODPROJ)
FOR UPDATE;
```

Db2 debe procesar más cuando se utiliza la cláusula FOR UPDATE sin una lista de columnas que cuando se utiliza la cláusula FOR UPDATE con una lista de columnas. Por lo tanto, si solo desea actualizar unas pocas columnas de una tabla, su programa puede funcionar de manera más eficiente si incluye una lista de columnas.

Las opciones de precompilador NOFOR y STDSQL afectan al uso de la cláusula FOR UPDATE en sentencias SQL estáticas. Si no especifica la cláusula FOR UPDATE en una instrucción DECLARE CURSOR, y no especifica la opción STDSQL(YES) o las opciones del precompilador NOFOR, recibirá un error si ejecuta una instrucción UPDATE posicionada.

Puede actualizar una columna de la tabla identificada aunque no forme parte de la tabla de resultados. En este caso, no es necesario nombrar la columna en la instrucción SELECT. Cuando el cursor recupera una fila (mediante FETCH) que contiene un valor de columna que desea actualizar, puede utilizar UPDATE. WHERE CURRENT OF para identificar la fila que se va a actualizar.

Tabla de resultados de solo lectura

Algunas tablas de resultados no se pueden actualizar, por ejemplo, el resultado de unir dos o más tablas.

Conceptos relacionados

[Seguridad multinivel \(Gestión de la seguridad\)](#)

Referencia relacionada

[Descripciones de opciones de proceso de SQL](#)

Puede especificar cualquier opción de proceso de SQL tanto si utiliza el precompilador de Db2 como si utiliza el coprocesador de Db2. Sin embargo, es probable que el coprocesador de Db2 omita ciertas opciones ya que existen opciones del compilador del lenguaje principal que proporcionan la misma información.

[DECLARE CURSOR declaración \(Db2 SQL\)](#)

[sentencia-select \(Db2 SQL\)](#)

Apertura de un cursor de fila

Después de declarar un cursor de fila, debe decirle a Db2 que está preparado para procesar la primera fila de la tabla de resultados. Esta acción se invoca abriendo el cursor.

Acerca de esta tarea

Para abrir un cursor de fila, ejecute la instrucción OPEN en su programa. Db2 y luego utiliza la instrucción SELECT dentro de DECLARE CURSOR para identificar un conjunto de filas. Si utiliza variables de host en la condición de búsqueda de esa instrucción SELECT, Db2 utiliza el **valor actual** de las variables para seleccionar las filas. La tabla de resultados que satisface la condición de búsqueda puede contener cero, una o varias filas. Un ejemplo de una sentencia OPEN es:

```
EXEC SQL  
  OPEN C1  
END-EXEC.
```

Si utiliza los registros especiales CURRENT DATE, CURRENT TIME o CURRENT TIMESTAMP en un cursor, Db2 determina los valores de esos registros especiales solo cuando abre el cursor. Db2 utiliza los valores que obtuvo en el momento de la apertura (OPEN) para todas las declaraciones FETCH posteriores.

Dos factores que influyen en la cantidad de tiempo que requiere Db2 para procesar la declaración OPEN son:

- Si Db2 debe realizar algún tipo de operación antes de poder recuperar filas
- Si Db2 utiliza paralelismo para procesar la instrucción SELECT del cursor

Especificar la acción que debe realizar el cursor de fila cuando llegue al final de los datos

Su programa debe estar codificado para reconocer y manejar una condición de fin de datos siempre que utilice un cursor de fila para recuperar una fila.

Acerca de esta tarea

Para determinar si el programa ha recuperado la última fila de datos, compruebe si el campo SQLCODE tiene un valor de 100 o el campo SQLSTATE un valor de '02000'. Estos códigos se producen cuando una sentencia FETCH ha recuperado la última fila de la tabla de resultados y su programa emite un FETCH posterior. Por ejemplo:

```
IF SQLCODE = 100 GO TO DATA-NOT-FOUND.
```

Una alternativa a esta técnica es codificar la sentencia WHENEVER NOT FOUND. La sentencia WHENEVER NOT FOUND hace que su programa se ramifique a otra parte que luego emite una sentencia CLOSE. Por ejemplo, para bifurcarse a la etiqueta DATA-NOT-FOUND cuando la sentencia FETCH no devuelve una fila, utilice esta sentencia:

```
EXEC SQL  
  WHENEVER NOT FOUND GO TO DATA-NOT-FOUND  
END-EXEC.
```

Para más información sobre la declaración WHENEVER NOT FOUND, consulte ["Comprobación de la ejecución de sentencias SQL"](#) en la página 556.

Ejecución de sentencias SQL utilizando un cursor de fila

Puede utilizar cursores de fila para ejecutar sentencias FETCH, sentencias UPDATE de posición y sentencias DELETE de posición.

Acerca de esta tarea

Ejecutar una sentencia FETCH para uno de los siguientes propósitos:

- Copiar datos de una fila de la tabla de resultados a una o más variables del host
- Para colocar el cursor antes de realizar una operación de actualización o eliminación posicionada

El siguiente ejemplo muestra una sentencia FETCH que recupera columnas seleccionadas de la tabla de empleados:

```
EXEC SQL
  FETCH C1 INTO
    :HV-EMPNO, :HV-FIRSTNAME, :HV-MIDINIT, :HV-LASTNAME, :HV-SALARY :IND-SALARY
END-EXEC.
```

La sentencia SELECT dentro de la sentencia DECLARE CURSOR identifica la tabla de resultados de la que se obtienen filas, pero Db2 no recupera ningún dato hasta que el programa de aplicación ejecuta una sentencia FETCH.

Cuando su programa ejecuta la sentencia FETCH, Db2 posiciona el cursor en una fila de la tabla de resultados. Esa fila se llama *la fila actual*. Db2 y luego copia el contenido de la fila actual en las variables de host del programa que especifique en la cláusula INTO de FETCH. Esta secuencia se repite cada vez que emite FETCH, hasta que procesa todas las filas de la tabla de resultados.

La fila a la que apunta Db2 cuando ejecutas una sentencia FETCH depende de si el cursor se declara como desplazable o no desplazable.

Cuando consultes un subsistema remoto con FETCH, considera usar la obtención de bloques para un mejor rendimiento. Bloquear los procesos de obtención de filas anteriores a la fila actual. No puede utilizar una obtención en bloque cuando realiza una operación de actualización o eliminación posicionada.

Después de que su programa haya ejecutado una sentencia FETCH para recuperar la fila actual, puede utilizar una sentencia UPDATE posicionada para modificar los datos de esa fila. Un ejemplo de una instrucción UPDATE posicionada es:

```
EXEC SQL
  UPDATE DSN8C10.EMP
    SET SALARY = 50000
    WHERE CURRENT OF C1
END-EXEC.
```

Una instrucción UPDATE posicionada actualiza la fila en la que se encuentra el cursor.

Una instrucción UPDATE posicionada está sujeta a estas restricciones:

- No puede actualizar una fila si su actualización infringe alguna restricción única, de verificación o referencial.
- No puede utilizar una instrucción UPDATE para modificar las filas de una tabla temporal creada. Sin embargo, puede utilizar una instrucción UPDATE para modificar las filas de una tabla temporal declarada.
- Si el lado derecho de la cláusula SET en la sentencia UPDATE contiene un fullselect, ese fullselect no puede incluir un nombre correlacionado para una tabla que se está actualizando.
- No se puede utilizar una instrucción SQL de modificación de datos en la cláusula FROM de una instrucción SELECT que defina un cursor que se utilice en una instrucción UPDATE posicionada.
- Una sentencia UPDATE posicionada fallará si el valor de la columna de etiqueta de seguridad de la fila donde está posicionado el cursor no es equivalente al valor de etiqueta de seguridad de su ID de usuario. Si su identificación de usuario tiene privilegios de escritura, una instrucción UPDATE posicionada fallará si el valor de la columna de etiqueta de seguridad de la fila donde está posicionado el cursor no domina el valor de etiqueta de seguridad de su identificación de usuario.

Después de que su programa haya ejecutado una sentencia FETCH para recuperar la fila actual, puede utilizar una sentencia DELETE posicionada para eliminar esa fila. Un ejemplo de una instrucción DELETE posicionada tiene este aspecto:

```
EXEC SQL
  DELETE FROM DSN8C10.EMP
```

```
WHERE CURRENT OF C1  
END-EXEC.
```

Una instrucción DELETE posicionada elimina la fila en la que está posicionado el cursor.

Una instrucción DELETE posicionada está sujeta a estas restricciones:

- No se puede utilizar una instrucción DELETE con un cursor para eliminar filas de una tabla temporal creada. Sin embargo, puede utilizar una instrucción DELETE con un cursor para eliminar filas de una tabla temporal declarada.
- Después de haber eliminado una fila, no podrá actualizar o eliminar otra fila utilizando ese cursor hasta que ejecute una instrucción FETCH para posicionar el cursor en otra fila.
- No puede eliminar una fila si al hacerlo se infringen restricciones referenciales.
- No se puede utilizar una instrucción SQL de modificación de datos en la cláusula FROM de una instrucción SELECT que defina un cursor que se utilice en una instrucción DELETE posicionada.
- Una instrucción DELETE posicionada fallará si el valor de la columna de etiqueta de seguridad de la fila donde está posicionado el cursor no es equivalente al valor de etiqueta de seguridad de su ID de usuario. Si su identificación de usuario tiene privilegios de escritura, una instrucción DELETE posicionada fallará si el valor de la columna de etiqueta de seguridad de la fila donde está posicionado el cursor no domina el valor de etiqueta de seguridad de su identificación de usuario.

Cerrar un cursor de fila

Cierre un cursor de fila cuando termine de procesar filas si desea liberar los recursos o si desea volver a utilizar el cursor. De lo contrario, puede dejar que Db2 cierre automáticamente el cursor cuando finalice la transacción actual o cuando finalice su programa.

Acerca de esta tarea

Para liberar los recursos que están retenidos por el cursor, cierre el cursor explícitamente emitiendo la instrucción CLOSE.

Si desea volver a utilizar el cursor de conjunto de filas, vuelva a abrirlo.

Procedimiento

Emitir una declaración de CIERRE.

Un ejemplo de una instrucción CLOSE tiene este aspecto:

```
EXEC SQL  
CLOSE C1  
END-EXEC.
```

Acceso a datos utilizando un cursor colocado por conjunto de filas

Un cursor colocado por conjunto de filas es un cursor que puede devolver una o varias filas para una única operación de captación. El cursor se coloca en el conjunto de filas que se van a captar.

Procedimiento

Para acceder a los datos utilizando un cursor posicionado en fila:

1. Ejecutar una instrucción DECLARE CURSOR para definir la tabla de resultados en la que opera el cursor. Consulte “[Declarar un cursor de fila](#)” en la página 434.
2. Ejecutar un CURSOR ABIERTO para que el cursor esté disponible para la aplicación. Consulte “[Abrir un cursor de fila](#)” en la página 434.
3. Especifique lo que el programa debe hacer cuando se hayan recuperado todas las filas. Consulte “[Especificar la acción que debe realizar el cursor de conjunto de filas cuando llegue al final de los datos](#)” en la página 434.

4. Ejecutar varias sentencias SQL para recuperar datos de la tabla o modificar filas seleccionadas de la tabla. Consulte “[Ejecución de sentencias SQL utilizando un cursor de conjunto de filas](#)” en la página 435.
5. Ejecutar una instrucción CLOSE CURSOR para que el cursor no esté disponible para la aplicación. Consulte “[Cerrar un cursor de fila](#)” en la página 438.

Resultados

Su programa puede tener varios cursores, cada uno de los cuales realiza los pasos anteriores.

Declarar un cursor de fila

Antes de poder utilizar un cursor posicionado en un conjunto de filas para recuperar filas, debe declarar un cursor que esté habilitado para recuperar conjuntos de filas. Cuando declara un cursor, identifica un conjunto de filas a las que se debe acceder con el cursor.

Acerca de esta tarea

Para conocer las restricciones que se aplican a los cursores posicionados en fila y a los cursores posicionados en fila, consulte “[Declarar un cursor de fila](#)” en la página 429.

Procedimiento

Utilice la cláusula WITH ROWSET POSITIONING en la instrucción DECLARE CURSOR. El siguiente ejemplo muestra cómo declarar un cursor de conjunto de filas:

```
EXEC SQL
  DECLARE C1 CURSOR WITH ROWSET POSITIONING FOR
    SELECT EMPNO, LASTNAME, SALARY
      FROM DSN8C10.EMP
  END-EXEC.
```

Abrir un cursor de fila

Después de declarar un cursor de conjunto de filas, debe indicar a Db2 que está listo para procesar el primer conjunto de filas de la tabla de resultados. Esta acción se invoca abriendo el cursor.

Acerca de esta tarea

Para abrir un cursor de conjunto de filas, ejecute la instrucción OPEN en su programa. Db2 y luego utiliza la instrucción SELECT dentro de DECLARE CURSOR para identificar las filas en la tabla de resultados. Para obtener más información sobre el proceso de CURSOR ABIERTO, consulte “[Apertura de un cursor de fila](#)” en la página 431.

Especificar la acción que debe realizar el cursor de conjunto de filas cuando llegue al final de los datos

Su programa debe estar codificado para reconocer y manejar una condición de fin de datos siempre que utilice un cursor de conjunto de filas para recuperar filas.

Acerca de esta tarea

Para determinar si el programa ha recuperado la última fila de datos de la tabla de resultados, compruebe si el campo SQLCODE tiene un valor de +100 o el campo SQLSTATE un valor de '02000'. Con un cursor de fila, estos códigos aparecen cuando una sentencia FETCH recupera la última fila de la tabla de resultados. Sin embargo, cuando se ha recuperado la última fila, el programa debe seguir procesando las filas del último conjunto de filas hasta esa última fila. Para ver un ejemplo de procesamiento de fin de datos para un cursor de conjunto de filas, consulte “[Ejemplos de capturas de filas con cursores](#)” en la página 452.

Para determinar el número de filas recuperadas, utilice uno de los siguientes valores:

- El contenido del campo SQLERRD(3) en el archivo SQLCA
- El contenido del elemento ROW_COUNT de GET DIAGNOSTICS

Para obtener información sobre GET DIAGNOSTICS, consulte “[Comprobación de la ejecución de sentencias SQL utilizando la instrucción GET DIAGNOSTICS](#)” en la página 563.

Si declara el cursor como desplazable dinámicamente y SQLCODE tiene el valor +100, puede continuar con una instrucción FETCH hasta que no se recuperen más filas. Las recuperaciones adicionales podrían recuperar más filas porque un cursor desplazable dinámico es sensible a las actualizaciones de otros procesos de la aplicación. Para obtener información sobre los cursores dinámicos, consulte “[Tipos de cursores](#)” en la página 424.

Ejecución de sentencias SQL utilizando un cursor de conjunto de filas

Puede utilizar cursores de conjunto de filas para ejecutar sentencias FETCH de varias filas, sentencias UPDATE de posición y sentencias DELETE de posición.

Acerca de esta tarea

Puede ejecutar estas sentencias SQL estáticas cuando utilice un cursor de conjunto de filas:

- Una sentencia FETCH de varias filas que copia un conjunto de filas de valores de columna en cualquiera de las siguientes áreas de datos:
 - Matrices de variables de host que se declaran en su programa
 - Matrices asignadas dinámicamente cuyas direcciones de almacenamiento se colocan en un área de descriptor SQL (SQLDA), junto con los atributos de las columnas que se van a recuperar
- Después de cualquiera de las dos formas de la sentencia FETCH de varias filas, puede emitir:
 - Una instrucción UPDATE posicionada en el conjunto de filas actual
 - Una instrucción DELETE posicionada en el conjunto de filas actual

Debe utilizar la cláusula WITH ROWSET POSITIONING de la sentencia DECLARE CURSOR si tiene previsto utilizar una sentencia FETCH con posicionamiento de conjuntos de filas.

El siguiente ejemplo muestra una sentencia FETCH que recupera 20 filas en matrices de variables de host que se declaran en su programa:

```
EXEC SQL
  FETCH NEXT ROWSET FROM C1
  FOR 20 ROWS
  INTO :HVA-EMPNO, :HVA-LASTNAME, :HVA-SALARY :INDA-SALARY
END-EXEC.
```

Cuando su programa ejecuta una sentencia FETCH con la palabra clave ROWSET, el cursor se posiciona en un conjunto de filas en la tabla de resultados. Ese conjunto de filas se denomina *conjunto de filas actual*. La dimensión de cada una de las matrices de variables de host debe ser mayor o igual que el número de filas que se van a recuperar.

Supongamos que desea asignar dinámicamente el almacenamiento necesario para las matrices de valores de columna que se van a recuperar de la tabla de empleados. Debe realizar lo siguiente:

1. Declare una estructura SQLDA y las variables que hacen referencia a la SQLDA.
2. Asignar dinámicamente el SQLDA y las matrices necesarias para los valores de las columnas.
3. Establezca los campos en el SQLDA para los valores de columna que se van a recuperar.
4. Abre el cursor.
5. Recuperar las filas.

Primero debe declarar la estructura SQLDA. La siguiente instrucción SQL INCLUDE solicita una declaración SQLDA estándar:

```
EXEC SQL INCLUDE SQLDA;
```

Su programa también debe declarar variables que hagan referencia a la estructura SQLDA, la estructura SQLVAR dentro de SQLDA y la estructura DECLEN para la precisión y la escala si está recuperando una columna DECIMAL. Para los programas C, el código tiene este aspecto:

```
struct sqlda *sqldaptr;
struct sqlvar *varptr;
struct DECLEN {
    unsigned char precision;
    unsigned char scale;
};
```

Antes de poder configurar los campos en el SQLDA para los valores de columna que se van a recuperar, debe asignar dinámicamente el almacenamiento para la estructura SQLDA. Para los programas C, el código tiene este aspecto:

```
sqldaptr = (struct sqlda *) malloc (3 * 44 + 16);
```

El tamaño del SQLDA es SQLN * 44 + 16, donde el valor del campo SQLN es el número de columnas de salida.

Debe configurar los campos en la estructura SQLDA para su instrucción FETCH. Supongamos que desea recuperar las columnas EMPNO, LASTNAME y SALARY. El código C para configurar los campos SQLDA para estas columnas tiene este aspecto:

```
strcpy(sqldaptr->sqldaid,"SQLDA");
sqldaptr->sqldabc = 148;           /* number bytes of storage allocated for the SQLDA */
sqldaptr->sqln = 3;                /* number of SQLVAR occurrences */
sqldaptr->sqld = 3;
varptr = (struct sqlvar *) (&(sqldaptr->sqlvar[0]));      /* Point to first SQLVAR */
varptr->sqltype = 452;              /* data type CHAR(6) */
varptr->sqllen = 6;
varptr->sqldata = (char *) hva1;
varptr->sqlind = (short *) inda1;
varptr->sqlname.length = 8;
memcpy(varptr->sqlname.data, "\x00\x00\x00\x00\x00\x01\x00\x14", varptr->sqlname.length);
varptr = (struct sqlvar *) (&(sqldaptr->sqlvar[0]) + 1); /* Point to next SQLVAR */
varptr->sqltype = 448;              /* data type VARCHAR(15) */
varptr->sqllen = 15;
varptr->sqldata = (char *) hva2;
varptr->sqlind = (short *) inda2;
varptr->sqlname.length = 8;
memcpy(varptr->sqlname.data, "\x00\x00\x00\x00\x00\x01\x00\x14", varptr->sqlname.length);
varptr = (struct sqlvar *) (&(sqldaptr->sqlvar[0]) + 2); /* Point to next SQLVAR */
varptr->sqltype = 485;              /* data type DECIMAL(9,2) */
((struct DECLEN *) &(varptr->sqllen))->precision = 9;
((struct DECLEN *) &(varptr->sqllen))->scale = 2;
varptr->sqldata = (char *) hva3;
varptr->sqlind = (short *) inda3;
varptr->sqlname.length = 8;
memcpy(varptr->sqlname.data, "\x00\x00\x00\x00\x00\x01\x00\x14", varptr->sqlname.length);
```

La estructura SQLDA tiene estos campos:

- SQLDABC indica el número de bytes de almacenamiento que se asignan para el SQLDA. El almacenamiento incluye un encabezado de 16 bytes y 44 bytes para cada campo SQLVAR. El valor es SQLN x 44 + 16, o 148 para este ejemplo.
- SQLN es el número de apariciones de SQLVAR (o el número de columnas de salida).
- SQLD es el número de variables en el SQLDA que son utilizadas por el Db2 cuando procesa la sentencia FETCH.
- Cada aparición de SQLVAR describe una matriz de variables de host o un búfer en el que se devolverán los valores de una columna de la tabla de resultados. Dentro de cada SQLVAR:
 - SQLTYPE indica el tipo de datos de la columna.
 - SQLLEN indica la longitud de la columna. Si el tipo de datos es DECIMAL, este campo tiene dos partes: la PRECISIÓN y la ESCALA.

- SQLDATA apunta al primer elemento de la matriz para los valores de columna. Para este ejemplo, supongamos que su programa asigna las matrices de variables dinámicas hva1, hva2 y hva3, y sus matrices de indicadores inda1, inda2 y inda3.
- SQLIND apunta al primer elemento de la matriz de valores indicadores para la columna. Si SQLTYPE es un número impar, este atributo es obligatorio. (Si SQLTYPE es un número impar, se permiten valores nulos para la columna)
- SQLNAME tiene dos partes: LENGTH y DATA. La LONGITUD es de 8. Los dos primeros bytes del campo DATA son X'0000'. Los bytes 5 y 6 del campo DATA son un indicador que indica si la variable es una matriz o un valor FOR n ROWS. Los bytes 7 y 8 son una representación entera binaria de dos bytes de la dimensión de la matriz.

Puede abrir el cursor solo después de que se hayan configurado todos los campos en el SQLDA de salida:

```
EXEC SQL OPEN C1;
```

Después de la sentencia OPEN, el programa busca el siguiente conjunto de filas:

```
EXEC SQL
  FETCH NEXT ROWSET FROM C1
  FOR 20 ROWS
  USING DESCRIPTOR :*sqlaptr;
```

La cláusula USING de la sentencia FETCH nombra el SQLDA que describe las columnas que se van a recuperar.

Después de que su programa ejecute una sentencia FETCH para establecer el conjunto de filas actual, puede utilizar una sentencia UPDATE posicionada con cualquiera de las siguientes cláusulas:

- Utilice WHERE CURRENT OF para modificar todas las filas del conjunto de filas actual
- Utilice PARA LA FILA *n* DEL CONJUNTO DE FILAS para modificar la fila *n* en el conjunto de filas actual

Un ejemplo de una instrucción UPDATE posicionada que utiliza la cláusula WHERE CURRENT OF es:

```
EXEC SQL
  UPDATE DSN8C10.EMP
    SET SALARY = 50000
    WHERE CURRENT OF C1
END-EXEC.
```

Cuando se ejecuta la instrucción UPDATE, el cursor debe colocarse en una fila o conjunto de filas de la tabla de resultados. Si el cursor se coloca en una fila, esa fila se actualiza. Si el cursor se coloca en un conjunto de filas, se actualizan todas las filas del conjunto.

Un ejemplo de una sentencia UPDATE posicionada que utiliza la cláusula FOR ROW *n* OF ROWSET es:

```
EXEC SQL
  UPDATE DSN8C10.EMP
    SET SALARY = 50000
    FOR CURSOR C1 FOR ROW 5 OF ROWSET
END-EXEC.
```

Cuando se ejecuta la instrucción UPDATE, el cursor debe colocarse en un conjunto de filas de la tabla de resultados. Se actualiza la fila especificada (en el ejemplo, la fila 5) del conjunto de filas actual.

Después de que su programa ejecute una sentencia FETCH para establecer el conjunto de filas actual, puede utilizar una sentencia DELETE posicionada con cualquiera de las siguientes cláusulas:

- Utilice WHERE CURRENT OF para eliminar todas las filas del conjunto de filas actual
- Utilice PARA LA FILA *n* DEL CONJUNTO DE FILAS para eliminar la fila *n* del conjunto de filas actual

Un ejemplo de una instrucción DELETE posicionada que utiliza la cláusula WHERE CURRENT OF es:

```
EXEC SQL
  DELETE FROM DSN8C10.EMP
```

```
    WHERE CURRENT OF C1  
END-EXEC.
```

Cuando se ejecuta la instrucción DELETE, el cursor debe colocarse en una fila o conjunto de filas de la tabla de resultados. Si el cursor se coloca en una fila, esa fila se elimina y el cursor se coloca antes de la siguiente fila de su tabla de resultados. Si el cursor se coloca en un conjunto de filas, se eliminan todas las filas del conjunto de filas y el cursor se coloca antes del siguiente conjunto de filas de su tabla de resultados.

Un ejemplo de una instrucción DELETE posicionada que utiliza la cláusula *FOR ROW n OF ROWSET* es:

```
EXEC SQL  
  DELETE FROM DSN8C10.EMP  
  FOR CURSOR C1 FOR ROW 5 OF ROWSET  
END-EXEC.
```

Cuando se ejecuta la instrucción DELETE, el cursor debe colocarse en un conjunto de filas de la tabla de resultados. Se elimina la fila especificada del conjunto de filas actual y el cursor permanece posicionado en ese conjunto de filas. La fila eliminada (en el ejemplo, la fila 5 del conjunto de filas) no se puede recuperar ni actualizar.

Tareas relacionadas

[Inclusión de SQL dinámico en el programa](#)

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

[Ejecución de sentencias SQL utilizando un cursor de fila](#)

Puede utilizar cursores de fila para ejecutar sentencias FETCH, sentencias UPDATE de posición y sentencias DELETE de posición.

Referencia relacionada

[Área de descriptor de SQL \(SQLDA\) \(Db2 SQL\)](#)

Especificar el número de filas en un conjunto de filas

Si no se especifica explícitamente el número de filas en un conjunto de filas, Db2 determina implícitamente el número de filas basándose en la última solicitud de obtención.

Acerca de esta tarea

Para establecer explícitamente el tamaño de un conjunto de filas, utilice *la cláusula FOR n ROWS* en la sentencia FETCH. Si una sentencia FETCH especifica la palabra clave ROWSET, y no *la cláusula FOR n ROWS*, el tamaño del conjunto de filas se establece implícitamente al tamaño del conjunto de filas que se especificó más recientemente en una sentencia FETCH anterior. Si una sentencia FETCH anterior no especificaba *la cláusula FOR n ROWS* o la palabra clave ROWSET, el tamaño del conjunto de filas actual se establece implícitamente en 1. Para ver ejemplos de posicionamiento de conjuntos de filas, consulte [Tabla 75 en la página 451](#).

Cerrar un cursor de fila

Cierre un cursor de conjunto de filas cuando termine de procesar las filas si desea liberar los recursos o si desea volver a utilizar el cursor. De lo contrario, puede dejar que Db2 cierre automáticamente el cursor cuando finalice la transacción actual o cuando finalice su programa.

Acerca de esta tarea

Para liberar los recursos que contiene el cursor, ciérrelo explícitamente emitiendo la instrucción CLOSE.

Si desea volver a utilizar el cursor de conjunto de filas, vuelva a abrirlo.

Procedimiento

Emitir una declaración de CIERRE.

Recuperación de filas utilizando un cursor desplazable

Un *cursor desplazable* es un cursor que puede moverse tanto hacia delante como hacia atrás. Los cursores desplazables pueden estar colocados por fila o por conjunto de filas.

Procedimiento

Cuando se abre un cursor, este se coloca antes de la primera fila de la tabla de resultados. Puede mover un cursor desplazable por la tabla de resultados especificando una palabra clave de *orientación de obtención* en una instrucción FETCH.

Una palabra clave de orientación de búsqueda indica la posición absoluta o relativa del cursor cuando se ejecuta la sentencia FETCH. La siguiente tabla enumera las palabras clave de orientación de búsqueda que puede especificar y sus significados. Estas palabras clave se aplican tanto a los cursores desplazables posicionados en fila como a los cursores desplazables posicionados en conjunto de filas.

Tabla 73. Posiciones para un cursor desplazable

Palabra clave en la sentencia FETCH	Posición del cursor cuando se ejecuta FETCH ^{“1.a” en la página 439}
BEFORE	Antes de la primera fila
PRIMERO o ABSOLUTO +1	En la primera fila
ÚLTIMO o ABSOLUTO -1	En la última fila
AFTER	Después de la última fila
absolutos ^{“1.b” en la página 439}	En un número de fila absoluto, desde antes de la primera fila hacia adelante o desde después de la última fila hacia atrás
relativo ^{“1.b” en la página 439}	En la fila que está adelante o atrás un número relativo de filas desde la fila actual
ACTUAL	En la fila actual
PARIENTE O PRIMO -1	En la fila anterior
SIGUIENTE	En la siguiente fila (por defecto)

Notas de la tabla

- La posición del cursor se aplica tanto a la posición de la fila como a la posición del conjunto de filas, por ejemplo, antes de la primera fila o antes del primer conjunto de filas.
- Para obtener más información sobre ABSOLUTO y RELATIVO, consulte [FETCH declaración \(Db2 SQL\)](#)

Ejemplo

Para utilizar el cursor que se declara en “[Tipos de cursores](#)” en la página 424 para obtener la quinta fila de la tabla de resultados, utilice una instrucción FETCH como esta:

```
EXEC SQL FETCH ABSOLUTE +5 C1 INTO :HVDEPTNO, :DEPTNAME, :MGRNO;
```

Para obtener la quinta fila desde el final de la tabla de resultados, utilice esta instrucción FETCH:

```
EXEC SQL FETCH ABSOLUTE -5 C1 INTO :HVDEPTNO, :DEPTNAME, :MGRNO;
```

Conceptos relacionados

Tipos de cursores

Puede declarar cursores colocados por fila o por conjunto de filas de varias formas. Estos cursores pueden ser desplazables o no desplazables, retenidos o no retenidos, o retornables o no retornables.

Referencia relacionada

[FETCH declaración \(Db2 SQL\)](#)

Comparación de cursos desplazables

Si un cursor desplazable puede ver los cambios que realizan otros procesadores o cursos en los datos depende de cómo se declare el cursor. También depende del tipo de operación de captación que se ejecuta.

Cuando declara un cursor como SENSITIVE STATIC, los cambios que otros procesos o cursos realizan en la tabla subyacente **pueden ser visibles** en la tabla de resultados del cursor. Que esos cambios **sean** visibles depende de si especifica SENSITIVE o INSENSITIVE cuando ejecute sentencias FETCH con el cursor. Cuando especifique FETCH INSENSITIVE, los cambios que otros procesos u otros cursos realicen en la tabla subyacente no serán visibles en la tabla de resultados. Cuando especifique FETCH SENSITIVE, los cambios que otros procesos o cursos realicen en la tabla subyacente serán visibles en la tabla de resultados.

Cuando declara un cursor como DINÁMICO SENSIBLE, los cambios que otros procesos o cursos realizan en la tabla subyacente son visibles en la tabla de resultados después de que se confirmen los cambios.

La siguiente tabla resume los valores de sensibilidad y sus efectos en la tabla de resultados de un cursor desplazable.

Tabla 74. Cómo afecta la sensibilidad a la tabla de resultados para un cursor desplazable

DECLARE sensibilidad	OBTENER INSENSIBLE	OBTENER SENSIBLE
INSENSITIVE	No se ven cambios en la tabla subyacente en la tabla de resultados. No se permiten las sentencias UPDATE y DELETE posicionadas con el cursor.	No válido
ESTÁTICA SENSIBLE	En la tabla de resultados solo se muestran las actualizaciones y eliminaciones realizadas con el cursor.	Todas las actualizaciones y eliminaciones son visibles en la tabla de resultados. Los insertos realizados mediante otros procesos no son visibles en la tabla de resultados.
SENSIBLE DINÁMICO	No válido	Todos los cambios realizados son visibles en la tabla de resultados, incluidas las actualizaciones, eliminaciones, inserciones y cambios en el orden de las filas.

Desplazarse por una tabla en cualquier dirección

Utilice un cursor desplazable para moverse por la tabla tanto hacia delante como hacia atrás.

Acerca de esta tarea

Pregunta : ¿Cómo puedo obtener filas de una tabla en cualquier dirección?

Respuesta : Declare que el cursor se puede desplazar. Cuando selecciona filas de la tabla, puede utilizar las diversas formas de la sentencia FETCH para desplazarse a un número de fila absoluto, avanzar o retroceder un determinado número de filas, a la primera o última fila, antes de la primera fila o después de la última, hacia adelante o hacia atrás. Puede utilizar cualquier combinación de estas sentencias FETCH para cambiar de dirección repetidamente.

Puede utilizar un código como el del siguiente ejemplo para avanzar 10 registros en la tabla de departamentos, retroceder cinco registros y avanzar de nuevo tres registros:

```

/*************
/* Declare host variables */
/*************
EXEC SQL BEGIN DECLARE SECTION;
    char[37] hv_deptname;
EXEC SQL END DECLARE SECTION;
/*************
/* Declare scrollable cursor to retrieve department names */
/*************
EXEC SQL DECLARE C1 SCROLL CURSOR FOR
    SELECT DEPTNAME FROM DSN8C10.DEPT;
:
/*************
/* Open the cursor and position it before the start of      */
/* the result table.                                         */
/*************
EXEC SQL OPEN C1;
EXEC SQL FETCH BEFORE FROM C1;
/*************
/* Fetch first 10 rows                                     */
/*************
for(i=0;i<10;i++)
{
    EXEC SQL FETCH NEXT FROM C1 INTO :hv_deptname;
}
/*************
/* Save the value in the tenth row                         */
/*************
tenth_row=hv_deptname;
/*************
/* Fetch backward 5 rows                                    */
/*************
for(i=0;i<5;i++)
{
    EXEC SQL FETCH PRIOR FROM C1 INTO :hv_deptname;
}
/*************
/* Save the value in the fifth row                          */
/*************
fifth_row=hv_deptname;
/*************
/* Fetch forward 3 rows                                    */
/*************
for(i=0;i<3;i++)
{
    EXEC SQL FETCH NEXT FROM C1 INTO :hv_deptname;
}
/*************
/* Save the value in the eighth row                         */
/*************
eighth_row=hv_deptname;
/*************
/* Close the cursor                                       */
/*************
EXEC SQL CLOSE C1;

```

Determinar el número de filas en la tabla de resultados para un cursor desplazable estático

Puede determinar cuántas filas hay en la tabla de resultados de un cursor desplazable INSENSITIVE o SENSITIVE STATIC.

Procedimiento

Para determinar el número de filas de la tabla de resultados para un cursor estático desplazable, siga estos pasos:

1. Ejecutar una sentencia FETCH, como FETCH AFTER, que posicione el cursor después de la última fila.
2. Realice una de las acciones siguientes:
 - Recuperar los valores de los campos SQLERRD(1) y SQLERRD(2) en SQLCA (campos sqlerrd[0] y sqlerrd[1] para C y C++). SQLERRD(1) y SQLERRD(2) juntos forman un valor de doble palabra que contiene el número de filas en la tabla de resultados.

- Emitir una declaración GET DIAGNOSTICS para recuperar el valor del elemento DB2_NUMBER_ROWS.

Ejemplo

El siguiente código en lenguaje C muestra cómo obtener el número de filas en una tabla de resultados de un cursor estático sensible.

```

EXEC SQL INCLUDE SQLCA;
long int rowcount;
EXEC SQL
  DECLARE SENSTAT SENSITIVE STATIC SCROLL CURSOR FOR
    SELECT * FROM EMP;
EXEC SQL OPEN SENSTAT;
if (SQLCODE==0) {
  EXEC SQL FETCH AFTER SENSTAT; /* Position the cursor after the end */
                                  /* of the result table           */
if (SQLCODE==0) {
  /***** */
  /* Get the row count from the SQLCA */
  /***** */
  printf("%s \n", "Row count from SQLCA: ");
  printf("%s %d\n", "SQLERRD1: High-order word: ",sqlca.sqlerrd[0]);
  /* Get the high-order word of the   */
  /* result table size               */
  /* result table size               */
printf("%s %d\n", "SQLERRD2: Low-order word: ",sqlca.sqlerrd[1]);
  /* Get the low-order word of the   */
  /* result table size               */
  /* result table size               */
  /***** */
  /* Get the row count from GET DIAGNOSTICS */
  /***** */
  EXEC SQL GET DIAGNOSTICS :rowcount = DB2_NUMBER_ROWS;
  if (SQLCODE==0) {
    printf("%s %d\n", "Row count from GET DIAGNOSTICS: ",rowcount);
  }
}
}

```

Eliminación de un orificio de borrado o de actualización

Si intenta recuperar datos de un agujero de eliminación o de actualización, Db2 emite una advertencia SQL. Si intenta actualizar o eliminar un agujero de eliminación o eliminar un agujero de actualización, Db2 emite un error SQL.

Acerca de esta tarea

Solo puede eliminar un agujero de borrado abriendo el cursor desplazable, estableciendo un punto de guardado, ejecutando una instrucción DELETE posicionada con el cursor desplazable y retrocediendo al punto de guardado.

Puede convertir un agujero de actualización de nuevo en una fila de la tabla de resultados actualizando la fila en la tabla base, como se muestra en la siguiente figura. Puede actualizar la tabla base con una sentencia UPDATE buscada en el mismo proceso de aplicación, o con una sentencia UPDATE buscada o posicionada en otro proceso de aplicación. Después de actualizar la tabla base, si la fila cumple los requisitos para la tabla de resultados, el agujero de actualización desaparece.

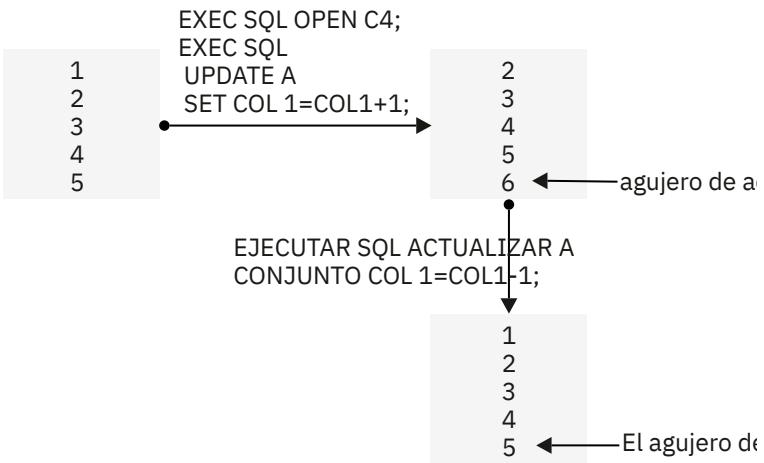


Figura 21. Eliminación de un orificio de actualización

Un agujero se vuelve visible para un cursor cuando una operación del cursor devuelve un SQLCODE distinto de cero. El punto en el que un agujero se hace visible depende de los siguientes factores:

- Si el cursor desplazable crea el agujero
- Si la sentencia FETCH es FETCH SENSITIVE o FETCH INSENSITIVE

Si el cursor desplazable crea el agujero, este será visible cuando ejecute una instrucción FETCH para la fila que contiene el agujero. La sentencia FETCH puede ser FETCH INSENSITIVE o FETCH SENSITIVE.

Si una operación de actualización o eliminación fuera del cursor desplazable crea el agujero, el agujero es visible en los siguientes momentos:

- Si ejecuta una sentencia FETCH SENSITIVE para la fila que contiene el agujero, el agujero es visible cuando ejecuta la sentencia FETCH.
- Si ejecuta una sentencia FETCH INSENSITIVE, el agujero no es visible cuando ejecuta la sentencia FETCH. Db2 devuelve la fila tal y como estaba antes de que se produjera la operación de actualización o eliminación. Sin embargo, si sigue la sentencia FETCH INSENSITIVE con una sentencia UPDATE o DELETE posicionada, el agujero se vuelve visible.

Espacios vacíos en la tabla de resultados de un cursor desplazable

Un espacio vacío en la tabla de resultados significa que la tabla de resultados no se reduce para llenar el espacio de filas suprimidas. Tampoco se contrae para llenar el espacio de filas que se han actualizado y ya no satisfacen la condición de búsqueda. No se puede acceder a un espacio vacío de supresión o actualización. Sin embargo, puede eliminar espacios vacíos en situaciones específicas.

En algunas situaciones, es posible que no pueda obtener una fila de la tabla de resultados de un cursor desplazable, dependiendo de cómo se declare el cursor:

- Los cursos desplazables que se declaran como INSENSITIVE o SENSITIVE STATIC siguen *un modelo estático*, lo que significa que Db2 determina el tamaño de la tabla de resultados y el orden de las filas cuando se abre el cursor.

Eliminar o actualizar filas después de que un cursor estático esté abierto puede dar lugar a agujeros en la tabla de resultados. Consulte “[Eliminación de un orificio de borrado o de actualización](#)” en la página 442.

- Los cursos desplazables que se declaran como SENSITIVE DYNAMIC siguen *un modelo dinámico*, lo que significa que el tamaño y el contenido de la tabla de resultados, así como el orden de las filas, pueden cambiar después de que abra el cursor.

Un cursor dinámico se desplaza directamente sobre la tabla base. Si la fila actual del cursor se elimina o se actualiza de modo que ya no cumple la condición de búsqueda, y la siguiente operación del cursor es FETCH CURRENT, entonces Db2 emite una advertencia SQL.

Los siguientes ejemplos demuestran cómo pueden producirse agujeros de borrado y actualización cuando se utiliza un cursor desplazable SENSITIVE STATIC.

Crear un agujero de eliminación con un cursor estático desplazable:

Supongamos que la tabla A consta de una columna de enteros, COL1, que tiene los valores que se muestran en la siguiente figura.

1
2
3
4
5

Figura 22. Valores para COL1 de la tabla A

Ahora suponga que declara el siguiente cursor desplazable SENSITIVE STATIC, que utiliza para eliminar filas de A:

```
EXEC SQL DECLARE C3 SENSITIVE STATIC SCROLL CURSOR FOR
  SELECT COL1
    FROM A
   FOR UPDATE OF COL1;
```

Ahora ejecute las siguientes sentencias SQL:

```
EXEC SQL OPEN C3;
EXEC SQL FETCH ABSOLUTE +3 C3 INTO :HVCOL1;
EXEC SQL DELETE FROM A WHERE CURRENT OF C3;
```

La instrucción delete posicionada crea un agujero de eliminación, como se muestra en la siguiente figura.

1	EXEC SQL	1
2	FETCH ABSOLUTE +3 C3	2
3	INTO :HVCOL1;	X
4	EJECUTAR SQL	4
5	ELIMINAR DE A DONDE	5
	ACTUAL DE C3;	

Borre agujero

Figura 23. agujero de supresión

Después de ejecutar la instrucción de eliminación posicionada, la tercera fila se elimina de la tabla de resultados, pero la tabla de resultados no se reduce para llenar el espacio que crea la fila eliminada.

Crear un agujero de actualización con un cursor estático desplazable

Supongamos que declara el siguiente cursor desplazable SENSITIVE STATIC, que utiliza para actualizar filas en A:

```
EXEC SQL DECLARE C4 SENSITIVE STATIC SCROLL CURSOR FOR
  SELECT COL1
    FROM A
   WHERE COL1<6;
```

Ahora ejecute las siguientes sentencias SQL:

```
EXEC SQL OPEN C4;
UPDATE A SET COL1=COL1+1;
```

La sentencia UPDATE buscada crea un agujero de actualización, como se muestra en la siguiente figura.

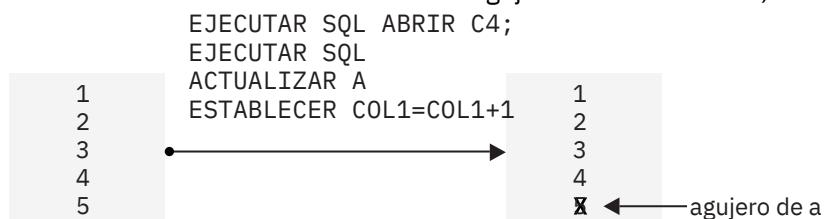


Figura 24. agujero de actualización

Después de ejecutar la instrucción UPDATE buscada, la última fila ya no cumple los requisitos para la tabla de resultados, pero la tabla de resultados no se reduce para llenar el espacio que crea la fila descalificada.

Acceso a datos XML o LOB rápidamente utilizando FETCH WITH CONTINUE

Utilice la sentencia FETCH WITH CONTINUE para mejorar el rendimiento de algunas consultas que hacen referencia a columnas XML y LOB con longitudes máximas muy grandes o desconocidas.

Acerca de esta tarea

FETCH WITH CONTINUE divide los valores XML y LOB en partes manejables y procesa las partes de una en una para evitar los siguientes problemas de asignación de búfer:

- Asignar un espacio excesivamente grande o innecesario para los búferes. Si algunos valores LOB son más cortos que la longitud máxima para valores en una columna, puede desperdiciar espacio de búfer si asigna suficiente espacio para la longitud máxima. El problema de la asignación de búfer puede ser aún peor para los datos XML porque una columna XML no tiene una longitud máxima definida. Si utiliza FETCH WITH CONTINUE, puede asignar un espacio de búfer más apropiado para la longitud real de los valores XML y LOB.
- Truncado de datos XML y LOB muy grandes. Si un valor XML o LOB muy grande no cabe en el espacio del búfer de la variable host que proporciona el programa de aplicación, Db2 truncará el valor. Si el programa de aplicación vuelve a intentar esta obtención con un búfer más grande, existen dos problemas. En primer lugar, cuando se utiliza un cursor no desplazable, no se puede volver a recuperar la fila actual sin cerrar, volver a abrir y volver a colocar el cursor en la fila que se truncó. En segundo lugar, si no utiliza FETCH WITH CONTINUE, Db2 no devuelve la longitud real del valor completo al programa de aplicación. Por lo tanto, Db2 no sabe cuán grande es el búfer que debe reasignar. Si utiliza FETCH WITH CONTINUE, Db2 conserva la parte truncada de los datos para su posterior recuperación y devuelve la longitud real del valor de datos completo para que la aplicación pueda reasignar un búfer del tamaño adecuado.

Db2 proporciona dos métodos para utilizar FETCH WITH CONTINUE con datos LOB y XML:

- [“Asignación dinámica de búferes al recuperar datos XML y LOB” en la página 445](#)
- [“Mover datos a través de búferes de tamaño fijo al recuperar datos XML y LOB” en la página 446](#)

Asignación dinámica de búferes al recuperar datos XML y LOB

Si especifica FETCH WITH CONTINUE, Db2 devuelve información sobre qué datos no caben en el búfer. Su aplicación puede entonces utilizar la información sobre los datos truncados para asignar un búfer de destino apropiado y ejecutar una operación de obtención con la cláusula CURRENT CONTINUE para recuperar los datos restantes.

Procedimiento

Para utilizar la asignación dinámica de búfer para datos LOB y XML:

1. Utiliza una FETCH WITH CONTINUE inicial para obtener datos en un búfer preasignado de tamaño moderado.

2. Si el valor es demasiado grande para caber en el búfer, utilice la información de longitud que devuelve Db2 para asignar la cantidad adecuada de almacenamiento.
3. Utilice una sola sentencia FETCH CURRENT CONTINUE para recuperar el resto de los datos.

Ejemplo

Supongamos que la tabla T1 se creó con la siguiente instrucción:

```
CREATE TABLE T1 (C1 INT, C2 CLOB(100M), C3 CLOB(32K), C4 XML);
```

Existe una fila en T1 donde C1 contiene un entero válido, C2 contiene 10MB de datos, C3 contiene 32KB de datos y C4 contiene 4MB de datos.

Ahora, suponga que declara CURSOR1, prepara y describe la sentencia DYNSQLSTMT1 con el descriptor SQLDA, y abre CURSOR1 con las siguientes sentencias:

```
EXEC SQL DECLARE CURSOR1 CURSOR FOR DYNSQLSTMT1;
EXEC SQL PREPARE DYNSQLSTMT1 FROM 'SELECT * FROM T1';
EXEC SQL DESCRIBE DYNSQLSTMT1 INTO DESCRIPTOR :SQLDA;
EXEC SQL OPEN CURSOR1;
```

A continuación, suponga que asigna búferes de tamaño moderado (32 KB para cada columna CLOB o XML) y establece punteros de datos y longitudes en SQLDA. A continuación, utiliza la siguiente instrucción FETCH WITH CONTINUE:

```
EXEC SQL FETCH WITH CONTINUE CURSOR1 INTO DESCRIPTOR :SQLDA;
```

Debido a que C2 y C4 contienen datos que no caben en el búfer, algunos de los datos se truncan. Su aplicación puede utilizar la información que devuelve Db2 para asignar búferes lo suficientemente grandes para los datos restantes y restablecer los punteros de datos y los campos de longitud en SQLDA. En ese momento, puede reanudar la obtención y completar el proceso con la siguiente instrucción FETCH CURRENT CONTINUE y la instrucción CLOSE CURSOR:

```
EXEC SQL FETCH CURRENT CONTINUE CURSOR1 INTO DESCRIPTOR :SQLDA;
EXEC SQL CLOSE CURSOR1;
```

La aplicación necesita concatenar los dos elementos devueltos del valor de datos. Una técnica consiste en mover el primer dato al búfer más grande asignado dinámicamente antes del FETCH CONTINUE. Establezca el puntero SQLDATA en la estructura SQLDA para que apunte inmediatamente después del último byte de este valor truncado. Db2 y, a continuación, escribe los datos restantes en esta ubicación y, de este modo, completa la concatenación.

Mover datos a través de búferes de tamaño fijo al recuperar datos XML y LOB

Si utiliza la cláusula WITH CONTINUE, Db2 devuelve información sobre qué datos no caben en el búfer. Su aplicación puede entonces utilizar operaciones FETCH CURRENT CONTINUE repetidas "para transmitir" de forma efectiva grandes datos XML y LOB a través de un búfer de tamaño fijo, de una en una.

Procedimiento

Para utilizar la asignación de búfer fijo para datos LOB y XML, siga estos pasos:

1. Utiliza una FETCH WITH CONTINUE inicial para obtener datos en un búfer preasignado de tamaño moderado.
2. Si el valor es demasiado grande para caber en el búfer, utilice tantas sentencias FETCH CONTINUE como sea necesario para procesar todos los datos a través de un búfer fijo.

Después de cada operación FETCH, compruebe si una columna se ha truncado examinando primero el campo "SQLWARN1" en el SQLCA devuelto. Si ese campo contiene un valor «W», al menos una columna de la fila devuelta se ha truncado. Para determinar si una columna LOB o XML concreta se ha truncado, su aplicación debe comparar el valor que se devuelve en el campo de longitud con la longitud declarada de la variable host. Si una columna está truncada, continúe utilizando sentencias FETCH CONTINUE hasta que se hayan recuperado todos los datos.

Después de obtener cada parte de los datos, muévalos fuera del búfer para dejar espacio para la siguiente obtención. Su aplicación puede escribir las piezas en un archivo de salida o reconstruir el valor de datos completo en un búfer por encima de la barra de 2 GB.

Ejemplo

Supongamos que la tabla T1 se creó con la siguiente instrucción:

```
CREATE TABLE T1 (C1 INT, C2 CLOB(100M), C3 CLOB(32K), C4 XML);
```

Existe una fila en T1 donde C2 contiene 10 MB de datos.

Ahora, suponga que declara una sección CLOBHV de 32 KB:

```
EXEC SQL BEGIN DECLARE SECTION  
DECLARE CLOBHV SQL TYPE IS CLOB(32767);  
EXEC SQL END DECLARE SECTION.
```

A continuación, supongamos que utiliza las siguientes sentencias para declarar y abrir CURSOR1 y para FETCH WITH CONTINUE:

```
EXEC SQL DECLARE CURSOR1 CURSOR FOR SELECT C2 FROM T1;  
EXEC SQL OPEN CURSOR1;  
EXEC SQL FETCH WITH CONTINUE CURSOR1 INTO :CLOBHV;
```

A medida que se recupera cada parte del valor de los datos, muévalo del búfer al archivo de salida.

Debido a que el valor de 10 MB en C2 no cabe en el búfer de 32 KB, algunos de los datos se truncan. Su aplicación puede recorrer los siguientes FETCH CURRENT CONTINUE:

```
EXEC SQL FETCH CURRENT CONTINUE CURSOR1 INTO :CLOBHV;
```

Después de cada operación FETCH, puede determinar si los datos se truncaron comprobando primero si el campo SQLWARN1 en el SQLCA devuelto contiene un valor 'W'. Si es así, compruebe si el valor de longitud, que se devuelve en CLOBHV_LENGTH, es mayor que la longitud declarada de 32767. (CLOBHV_LENGTH se declara como parte de la expansión del precompilador de la declaración CLOBHV) Si el valor es mayor, ese valor se ha truncado y se pueden recuperar más datos con la siguiente operación FETCH CONTINUE.

Cuando todos los datos se hayan trasladado al archivo de salida, puede cerrar el cursor:

```
EXEC SQL CLOSE CURSOR1;
```

Determinar los atributos de un cursor utilizando el SQLCA

Un área de comunicación SQL (SQLCA) es un área que se separa para la comunicación con un servidor de aplicaciones (Db2) y consiste en una colección de variables. El uso de SQLCA es una forma de obtener información sobre cualquier cursor abierto. También puede utilizar la instrucción GET DIAGNOSTICS.

Acerca de esta tarea

Después de abrir un cursor, puede determinar los siguientes atributos del cursor marcando los siguientes campos SQLWARN y SQLERRD de SQLCA:

SQLWARN1

Indica si el cursor se puede desplazar o no.

SQLWARN4

Indica si el cursor es insensible (I), estático sensible (S) o dinámico sensible (D).

SQLWARN5

Indica si el cursor es de solo lectura, legible y eliminable, o legible, eliminable y actualizable.

SQLERRD(1) y SQLERRD(2)

Estos dos campos juntos contienen un entero de doble palabra que representa el número de filas en la tabla de resultados de un cursor cuando el cursor se posiciona después de la última fila. El cursor se posiciona después de la última fila cuando el SQLCODE es 100. Estos campos no están configurados para cursores dinámicos desplazables.

sqlerrd(3)

El número de filas de la tabla de resultados cuando la instrucción SELECT del cursor contiene una instrucción de cambio de datos.

Si la instrucción OPEN se ejecuta sin errores ni advertencias, Db2 no establece SQLWARN0 cuando establece SQLWARN1, SQLWARN4 o SQLWARN5.

Referencia relacionada

[Descripción de campos de SQLCA \(Db2 SQL\)](#)

Determinar los atributos de un cursor mediante la instrucción GET DIAGNOSTICS

El uso de la instrucción GET DIAGNOSTICS es una forma de obtener información sobre cualquier cursor abierto. También puede utilizar el SQLCA.

Acerca de esta tarea

Después de abrir un cursor, puede determinar los siguientes atributos del cursor marcando estos elementos de OBTENER DIAGNÓSTICOS:

DB2_SQL_ATTR_CURSOR_HOLD

Indica si el cursor puede mantenerse abierto a través de las confirmaciones (S o N)

DB2_SQL_ATTR_CURSOR_ROWSET

Indica si el cursor puede utilizar el posicionamiento de conjuntos de filas (S o N)

DB2_SQL_ATTR_CURSOR_SCROLLABLE

Indica si el cursor se puede desplazar (S o N)

DB2_SQL_ATTR_CURSOR_SENSITIVITY

Indica si el cursor es insensible o sensible a los cambios que realizan otros procesos (I o S)

DB2_SQL_ATTR_CURSOR_TYPE

Indica si el cursor está hacia delante (F), declarado estático (S para INSENSITIVO o ESTÁTICO SENSITIVO) o dinámico (D para DINÁMICO SENSITIVO)

Para obtener más información sobre la declaración GET DIAGNOSTICS, consulte [“Comprobación de la ejecución de sentencias SQL utilizando la instrucción GET DIAGNOSTICS”](#) en la página 563.

Desplazarse por datos recuperados anteriormente

Para desplazarse hacia atrás a través de los datos, utilice un cursor desplazable, o utilice una columna ROWID o una columna de identidad para recuperar los datos en orden inverso.

Procedimiento

Cuando un programa recupera datos de la base de datos, puede desplazarse hacia atrás a través de los datos utilizando una de las siguientes técnicas:

- Utilice un cursor desplazable para retroceder a través de los datos siguiendo estos pasos:
 - a) Declare el cursor con la palabra clave SCROLL.
 - b) Abre el cursor.
 - c) Ejecutar una sentencia FETCH para posicionar el cursor al final de la tabla de resultados.
 - d) En un bucle, ejecute sentencias FETCH que muevan el cursor hacia atrás y luego recuperen los datos.

e) Cuando haya recuperado todos los datos, cierre el cursor.

Por ejemplo, puede utilizar un código como el del ejemplo siguiente para recuperar los nombres de los departamentos en orden inverso de la tabla DSN8C10.DEPT:

```
*****  
/* Declare host variables */  
*****  
EXEC SQL BEGIN DECLARE SECTION;  
    char[37] hv_deptname;  
EXEC SQL END DECLARE SECTION;  
*****  
/* Declare scrollable cursor to retrieve department names */  
*****  
EXEC SQL DECLARE C1 SCROLL CURSOR FOR  
    SELECT DEPTNAME FROM DSN8C10.DEPT;  
:  
*****  
/* Open the cursor and position it after the end of the */  
/* result table. */  
*****  
EXEC SQL OPEN C1;  
EXEC SQL FETCH AFTER FROM C1;  
*****  
/* Fetch rows backward until all rows are fetched. */  
*****  
while(SQLCODE==0) {  
    EXEC SQL FETCH PRIOR FROM C1 INTO :hv_deptname;  
:  
}  
EXEC SQL CLOSE C1;
```

- Si la tabla contiene un ROWID o una columna de identidad, recupera los valores de esa columna en una matriz. A continuación, utilice los valores de la columna ROWID o de identidad para recuperar las filas en orden inverso.

Puede utilizar la columna ROWID o la columna de identidad para recuperar rápidamente las filas en orden inverso. Cuando realiza la SELECCIÓN original, puede almacenar el ROWID o el valor de la columna de identidad para cada fila que recupera. Luego, para recuperar los valores en orden inverso, puede ejecutar sentencias SELECT con una cláusula WHERE que compare el valor de la columna ROWID o de identidad con cada valor almacenado.

Por ejemplo, supongamos que agrega la columna ROWID DEPTROWID a la tabla DSN8C10.DEPT. Puede utilizar un código como el del siguiente ejemplo para seleccionar todos los nombres de departamentos y, a continuación, recuperar los nombres en orden inverso:

```
*****  
/* Declare host variables */  
*****  
EXEC SQL BEGIN DECLARE SECTION;  
    SQL TYPE IS ROWID hv_dept_rowid;  
    char[37] hv_deptname;  
EXEC SQL END DECLARE SECTION;  
*****  
/* Declare other variables */  
*****  
struct rowid_struct {  
    short int length;  
    char data[40]; /* ROWID variable structure */  
}  
struct rowid_struct rowid_array[200];  
/* Array to hold retrieved */  
/* ROWIDs. Assume no more */  
/* than 200 rows will be */  
/* retrieved. */  
short int i,j,n;  
*****  
/* Declare cursor to retrieve department names */  
*****  
EXEC SQL DECLARE C1 CURSOR FOR  
    SELECT DEPTNAME, DEPTROWID FROM DSN8C10.DEPT;  
:  
*****  
/* Retrieve the department name and ROWID from DEPT table */  
/* and store the ROWID in an array. */  
*****  
EXEC SQL OPEN C1;
```

```

i=0;
while(SQLCODE==0) {
  EXEC SQL FETCH C1 INTO :hv_deptname, :hv_dept_rowid;
  rowid_array[i].length=hv_dept_rowid.length;
  for(j=0;j<hv_dept_rowid.length;j++)
    rowid_array[i].data[j]=hv_dept_rowid.data[j];
  i++;
}
EXEC SQL CLOSE C1;
n=i-1; /* Get the number of array elements */
/*****************/
/* Use the ROWID values to retrieve the department names */
/* in reverse order. */
/*****************/
for(i=n;i>=0;i--) {
  hv_dept_rowid.length=rowid_array[i].length;
  for(j=0;j<hv_dept_rowid.length;j++)
    hv_dept_rowid.data[j]=rowid_array[i].data[j];
  EXEC SQL SELECT DEPTNAME INTO :hv_deptname
    FROM DSN8C10.DEPT
    WHERE DEPTROWID=:hv_dept_rowid;
}

```

Conceptos relacionados

[Valores de ID de fila \(Db2 SQL\)](#)

[Columnas de identidad](#)

Una columna de identidad contiene un valor numérico exclusivo para cada fila de la tabla. Db2 puede generar automáticamente valores numéricos secuenciales para esta columna cuando las filas se insertan en la tabla. Así, las columnas de identidad son ideales para los valores de clave primaria, como los números de empleado o los números de producto.

Tareas relacionadas

[Recuperación de filas utilizando un cursor desplazable](#)

Un cursor desplazable es un cursor que puede moverse tanto hacia delante como hacia atrás. Los curosres desplazables pueden estar colocados por fila o por conjunto de filas.

Actualización de datos recuperados previamente

Para desplazarse hacia atrás por los datos y actualizarlos, utilice un cursor desplazable que se declare con la cláusula FOR UPDATE.

Acerca de esta tarea

Si un cursor utiliza sentencias FETCH para recuperar columnas que se actualizarán más tarde, especifique FOR UPDATE OF al seleccionar las columnas. A continuación, especifique WHERE CURRENT OF en las siguientes sentencias UPDATE o DELETE. Estas cláusulas impiden que Db2 seleccione el acceso a través de un índice en las columnas que se están actualizando, lo que de otro modo podría hacer que Db2 lea la misma fila más de una vez.

Procedimiento

Para actualizar los datos recuperados anteriormente, siga estos pasos:

1. Declare el cursor con las palabras clave SENSITIVE STATIC SCROLL.
2. Abre el cursor.
3. Emitir una sentencia FETCH para posicionar el cursor al final de la tabla de resultados.
4. Emite sentencias FETCH para mover el cursor hacia atrás hasta la fila que deseas actualizar.
5. Especifique la cláusula WHERE CURRENT OF en la sentencia UPDATE o DELETE que actualiza la fila actual.
6. Repita los pasos “4” en la página 450 y “5” en la página 450 hasta que se actualicen todas las filas necesarias.
7. Cuando haya recuperado y actualizado todos los datos, cierre el cursor.

Referencia relacionada

[cláusula-update \(Db2 SQL\)](#)

[DECLARE CURSOR declaración \(Db2 SQL\)](#)

[FETCH declaración \(Db2 SQL\)](#)

[UPDATE declaración \(Db2 SQL\)](#)

[DELETE declaración \(Db2 SQL\)](#)

Interacción de la sentencia **FETCH** entre el posicionamiento de fila y conjunto de filas

Cuando declara un cursor con la cláusula WITH ROWSET POSITIONING, puede entremezclar sentencias **FETCH** colocadas por fila con sentencias **FETCH** colocadas por conjunto de filas.

La siguiente tabla muestra la interacción entre el posicionamiento de fila y conjunto de filas para un cursor desplazable. Supongamos que declara el cursor desplazable en una tabla con 15 filas.

Tabla 75. Interacción entre el posicionamiento de fila y conjunto de filas para un cursor desplazable

Palabras clave en la sentencia FETCH	Posición del cursor cuando se ejecuta FETCH
PRIMERO	En la fila 1
Primer conjunto de filas	En una fila de tamaño 1, que consta de la fila 1
PRIMERA FILA PREPARADA PARA 5 FILAS	En una fila de tamaño 5, compuesta por las filas 1, 2, 3, 4 y 5
FILA ACTUAL	En una fila de tamaño 5, compuesta por las filas 1, 2, 3, 4 y 5
ACTUAL	En la fila 1
SIGUIENTE (predeterminado)	En la fila 2
Siguiente conjunto de filas	En una fila de tamaño 1, que consta de la fila 3
PRÓXIMA FILA PARA 3 FILAS	En un conjunto de filas de tamaño 3, que consta de las filas 4, 5 y 6
Siguiente conjunto de filas	En una fila de tamaño 3, compuesta por las filas 7, 8 y 9
LAST	En la fila 15
ÚLTIMA FILA PARA 2 FILAS	En una fila de tamaño 2, compuesta por las filas 14 y 15
Fila anterior	En una fila de tamaño 2, compuesta por las filas 12 y 13
ABSOLUTE 2	En la fila 2
FILAS A PARTIR DE LA 2 EN ABSOLUTO PARA 3 FILAS	En una fila de tamaño 3, compuesta por las filas 2, 3 y 4
PARIENTE 2	En la fila 4
FILAS A PARTIR DE LA 2ª ABSOLUTA PARA 4 FILAS	En una fila de tamaño 4, compuesta por las filas 2, 3, 4 y 5
PARIENTE -1	En la fila 1
FILAS A PARTIR DE LA 3 ABSOLUTA POR 2 FILAS	En una fila de tamaño 2, que consta de las filas 3 y 4
FILAS A PARTIR DE RELATIVO 4	En una fila de tamaño 2, compuesta por las filas 7 y 8
PRIOR	En la fila 6
FILAS A PARTIR DE LA 13 ABSOLUTA POR 5 FILAS	En una fila de tamaño 3, compuesta por las filas 13, 14 y 15

Tabla 75. Interacción entre el posicionamiento de fila y conjunto de filas para un cursor desplazable (continuación)

Palabras clave en la sentencia FETCH	Posición del cursor cuando se ejecuta FETCH
Primer conjunto de filas	En una fila de tamaño 5, compuesta por las filas 1, 2, 3, 4 y 5

Referencia relacionada

[FETCH declaración \(Db2 SQL\)](#)

Ejemplos de capturas de filas con cursos

Puede utilizar sentencias SQL incluidas en un programa COBOL para definir y utilizar el cursor no desplazable para las actualizaciones de posicionamiento de filas, los cursos desplazables para recuperar filas hacia atrás, los cursos no desplazables para actualizaciones de posicionamiento de conjuntos de filas y los cursos desplazables para las operaciones de posicionamiento de conjuntos de filas.

El siguiente ejemplo muestra cómo actualizar una fila utilizando un cursor.

```
*****
* Declare a cursor that will be used to update   *
* the JOB column of the EMP table.               *
*****
EXEC SQL
DECLARE THISEMP CURSOR FOR
  SELECT EMPNO, LASTNAME,
         WORKDEPT, JOB
    FROM DSN8C10.EMP
   WHERE WORKDEPT = 'D11'
  FOR UPDATE OF JOB
END-EXEC.
*****
* Open the cursor                                *
*****
EXEC SQL
  OPEN THISEMP
END-EXEC.
*****
* Indicate what action to take when all rows      *
* in the result table have been fetched.          *
*****
EXEC SQL
  WHENEVER NOT FOUND
    GO TO CLOSE-THISEMP
END-EXEC.
*****
* Fetch a row to position the cursor.             *
*****
EXEC SQL
  FETCH FROM THISEMP
    INTO :EMP-NUM, :NAME2,
         :DEPT, :JOB-NAME
END-EXEC.
*****
* Update the row where the cursor is positioned. *
*****
EXEC SQL
  UPDATE DSN8C10.EMP
    SET JOB = :NEW-JOB
   WHERE CURRENT OF THISEMP
END-EXEC.
:
*****
* Branch back to fetch and process the next row. *
:
*****
* Close the cursor                               *
*****
CLOSE-THISEMP.
EXEC SQL
```

```

CLOSE THISEMP
END-EXEC.
```

El siguiente ejemplo muestra cómo recuperar datos hacia atrás con un cursor.

```

*****
* Declare a cursor to retrieve the data backward *
* from the EMP table. The cursor has access to   *
* changes by other processes.                      *
*****
EXEC SQL
  DECLARE THISEMP SENSITIVE STATIC SCROLL CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT, JOB
      FROM DSN8C10.EMP
  END-EXEC.
*****
* Open the cursor                                *
*****
EXEC SQL
  OPEN THISEMP
  END-EXEC.
*****
* Indicate what action to take when all rows     *
* in the result table have been fetched.        *
*****
EXEC SQL
  WHENEVER NOT FOUND GO TO CLOSE-THISEMP
  END-EXEC.
*****
* Position the cursor after the last row of the  *
* result table. This FETCH statement cannot      *
* include the SENSITIVE or INSENSITIVE keyword   *
* and cannot contain an INTO clause.            *
*****
EXEC SQL
  FETCH AFTER FROM THISEMP
  END-EXEC.
*****
* Fetch the previous row in the table.           *
*****
EXEC SQL
  FETCH SENSITIVE PRIOR FROM THISEMP
    INTO :EMP-NUM, :NAME2, :DEPT, :JOB-NAME
  END-EXEC.
*****
* Check that the fetched row is not a hole       *
* (SQLCODE +222). If not, print the contents.   *
*****
IF SQLCODE IS GREATER THAN OR EQUAL TO 0 AND
  SQLCODE IS NOT EQUAL TO +100 AND
  SQLCODE IS NOT EQUAL TO +222 THEN
  PERFORM PRINT-RESULTS.
:
*****
* Branch back to fetch the previous row.         *
:
*****
* Close the cursor                               *
*****
CLOSE-THISEMP.
EXEC SQL
  CLOSE THISEMP
  END-EXEC.
```

El siguiente ejemplo muestra cómo actualizar un conjunto de filas completo con un cursor.

```

*****
* Declare a rowset cursor to update the JOB      *
* column of the EMP table.                      *
*****
EXEC SQL
  DECLARE EMPSET CURSOR
    WITH ROWSET POSITIONING FOR
      SELECT EMPNO, LASTNAME, WORKDEPT, JOB
        FROM DSN8C10.EMP
      WHERE WORKDEPT = 'D11'
```

```

FOR UPDATE OF JOB
END-EXEC.
*****
* Open the cursor. *
*****
EXEC SQL
  OPEN EMPSET
END-EXEC.
*****
* Indicate what action to take when end-of-data *
* occurs in the rowset being fetched. *
*****
EXEC SQL
  WHENEVER NOT FOUND
    GO TO CLOSE-EMPSET
END-EXEC.
*****
* Fetch next rowset to position the cursor. *
*****
EXEC SQL
  FETCH NEXT ROWSET FROM EMPSET
    FOR :SIZE-ROWSET ROWS
      INTO :HVA-EMPNO, :HVA-LASTNAME,
            :HVA-WORKDEPT, :HVA-JOB
  END-EXEC.
*****
* Update rowset where the cursor is positioned. *
*****
UPDATE-ROWSET.
  EXEC SQL
    UPDATE DSN8C10.EMP
      SET JOB = :NEW-JOB
        WHERE CURRENT OF EMPSET
  END-EXEC.
END-UPDATE-ROWSET.
:
*****
* Branch back to fetch the next rowset. *
*****
:
*****
* Update the remaining rows in the current   *
* rowset and close the cursor. *
*****
CLOSE-EMPSET.
  PERFORM UPDATE-ROWSET.
  EXEC SQL
    CLOSE EMPSET
  END-EXEC.

```

El siguiente ejemplo muestra cómo actualizar filas específicas con un cursor de conjunto de filas.

```

*****
* Declare a static scrollable rowset cursor. *
*****
EXEC SQL
  DECLARE EMPSET SENSITIVE STATIC SCROLL CURSOR
    WITH ROWSET POSITIONING FOR
      SELECT EMPNO, WORKDEPT, JOB
        FROM DSN8C10.EMP
          FOR UPDATE OF JOB
  END-EXEC.
*****
* Open the cursor. *
*****
EXEC SQL
  OPEN EMPSET
END-EXEC.
*****
* Fetch next rowset to position the cursor. *
*****
EXEC SQL
  FETCH SENSITIVE NEXT ROWSET FROM EMPSET
    FOR :SIZE-ROWSET ROWS
      INTO :HVA-EMPNO,
            :HVA-WORKDEPT :INDA-WORKDEPT,
            :HVA-JOB :INDA-JOB
  END-EXEC.

```

```

*****
* Process fetch results if no error and no hole.  *
*****
IF SQLCODE >= 0
  EXEC SQL GET DIAGNOSTICS
    :HV-ROWCNT = ROW_COUNT
END-EXEC
PERFORM VARYING N FROM 1 BY 1 UNTIL N > HV-ROWCNT
  IF INDA-WORKDEPT(N) NOT = -3
    EVALUATE HVA-WORKDEPT(N)
      WHEN ('D11')
        PERFORM UPDATE-ROW
      WHEN ('E11')
        PERFORM DELETE-ROW
    END-EVALUATE
  END-IF
END-PERFORM
IF SQLCODE = 100
  GO TO CLOSE-EMPSET
END-IF
ELSE
  EXEC SQL GET DIAGNOSTICS
    :HV-NUMCOND = NUMBER
END-EXEC
PERFORM VARYING N FROM 1 BY 1 UNTIL N > HV-NUMCOND
  EXEC SQL GET DIAGNOSTICS CONDITION :N
    :HV-SQLCODE = DB2_RETURNED_SQLCODE,
    :HV-ROWNUM = DB2_ROW_NUMBER
  END-EXEC
  DISPLAY "SQLCODE = " HV-SQLCODE
  DISPLAY "ROW NUMBER = " HV-ROWNUM
END-PERFORM
GO TO CLOSE-EMPSET
END-IF.

:
*****  

* Branch back to fetch and process          *
* the next rowset.                         *
*****  

:  

*****  

* Update row N in current rowset.          *
*****  

UPDATE-ROW.
  EXEC SQL
    UPDATE DSN8C10.EMP
    SET JOB = :NEW-JOB
    FOR CURSOR EMPSET FOR ROW :N OF ROWSET
  END-EXEC.
END-UPDATE-ROW.
*****  

* Delete row N in current rowset.          *
*****  

DELETE-ROW.
  EXEC SQL
    DELETE FROM DSN8C10.EMP
    WHERE CURRENT OF EMPSET FOR ROW :N OF ROWSET
  END-EXEC.
END-DELETE-ROW.
:  

*****  

* Close the cursor.                         *
*****  

CLOSE-EMPSET.
  EXEC SQL
    CLOSE EMPSET
  END-EXEC.

```

Especificación del acceso a filas directo utilizando los ID de fila

En algunas aplicaciones, es posible utilizar el valor de una columna ROWID para ir directamente a una fila.

Antes de empezar

Asegúrese de que la consulta cumple los requisitos para el acceso directo a la fila. Para cumplir los requisitos, la condición de búsqueda debe ser un término booleano, un predicado de la etapa 1 que se ajuste a uno de los siguientes criterios:

- Un simple predicado de término booleano de la siguiente forma:

```
RID (table designator) = noncolumn expression
```

Cuando la expresión no de columna contiene un resultado de una función RID.

- Término booleano compuesto que combina varios predicados simples mediante el operador AND, donde uno de los predicados simples se ajusta al primer criterio.

Acerca de esta tarea

Conceptos introductorios

Tipo de datos ROWID (Introducción a Db2 para z/OS)

Una columna *ROWID* identifica de forma única cada fila de una tabla. Con las columnas ROWID, puede escribir consultas que naveguen directamente a una fila de la tabla porque la columna contiene implícitamente la ubicación de la fila. Puede definir una columna ROWID como GENERADA POR DEFECTO o GENERADA SIEMPRE:

- Si define la columna como GENERATED BY DEFAULT, puede insertar un valor. Db2 proporciona un valor predeterminado si no se proporciona uno. Sin embargo, para poder insertar un valor explícito (mediante la instrucción INSERT con la cláusula VALUES), debe crear un índice único en esa columna.
- Si define la columna como GENERATED ALWAYS (que es la opción predeterminada), Db2 siempre genera un valor único para la columna. No puede insertar datos en esa columna. En este caso, Db2 no requiere un índice para garantizar valores únicos.

Cuando selecciona una columna ROWID, el valor contiene implícitamente la ubicación de la fila recuperada. Si utiliza el valor de la columna ROWID en la condición de búsqueda de una consulta posterior, Db2 puede optar por navegar directamente a esa fila.

Si define una columna de una tabla para que tenga el tipo de datos ROWID, Db2 proporciona un valor único para cada fila de la tabla solo si define la columna como GENERATED ALWAYS. El propósito del valor en la columna ROWID es identificar de forma única las filas de la tabla.

Puede utilizar una columna ROWID para escribir consultas que naveguen directamente a una fila, lo que puede ser útil en situaciones en las que se requiera un alto rendimiento. Esta navegación directa, sin utilizar un índice o escanear el espacio de la tabla, se denomina *acceso directo a filas*. Además, una columna ROWID es un requisito para las tablas que contienen columnas LOB. Este tema trata sobre el uso de una columna ROWID en el acceso directo a filas.

Por ejemplo, supongamos que una tabla EMPLOYEE se define de la siguiente manera:

```
CREATE TABLE EMPLOYEE
  (EMP_ROWID  ROWID NOT NULL GENERATED ALWAYS,
   EMPNO      SMALLINT,
   NAME       CHAR(30),
   SALARY     DECIMAL(7,2),
   WORKDEPT   SMALLINT);
```

El siguiente código utiliza la instrucción SELECT de INSERT para recuperar el valor de la columna ROWID de una nueva fila que se inserta en la tabla EMPLOYEE. Este valor se utiliza entonces para hacer referencia a esa fila para la actualización de la columna SALARIO.

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS ROWID hv_emp_rowid;
  short        hv_dept, hv_empno;
  char         hv_name[30];
  decimal(7,2) hv_salary;
EXEC SQL END DECLARE SECTION;
...
```

```

EXEC SQL
  SELECT EMP_ROWID INTO :hv_emp_rowid
  FROM FINAL_TABLE (INSERT INTO EMPLOYEE
                     VALUES (DEFAULT, :hv_empno, :hv_name, :hv_salary, :hv_dept));
EXEC SQL
  UPDATE EMPLOYEE
  SET SALARY = SALARY + 1200
  WHERE EMP_ROWID = :hv_emp_rowid;
EXEC SQL COMMIT;

```

Para que Db2 pueda utilizar el acceso directo a filas para la operación de actualización, la instrucción SELECT de INSERT y la instrucción UPDATE deben ejecutarse dentro de la misma unidad de trabajo. Como alternativa, puede utilizar una instrucción SELECT from MERGE. La sentencia MERGE realiza operaciones INSERT y UPDATE como una sola sentencia coordinada.

Requisito: Para utilizar el acceso directo de fila, debe utilizar un valor ROWID recuperado antes de confirmar. Cuando la aplicación se confirma, libera su reclamación en el espacio de tabla. Después de la confirmación, si se ejecuta un REORG en el espacio de tabla, la ubicación física de las filas puede cambiar.

Restricción: En general, no se puede utilizar una columna ROWID como clave que se va a utilizar como valor de una sola columna en varias tablas. El valor ROWID de una fila concreta de una tabla puede cambiar con el tiempo debido a una REORG del espacio de la tabla. En particular, no puede utilizar una columna ROWID como parte de una clave principal o clave externa.

El valor que recupera de una columna ROWID es un valor de caracteres de longitud variable que no es monótonamente ascendente o descendente (el valor no siempre aumenta o no siempre disminuye). Por lo tanto, una columna ROWID no proporciona valores adecuados para muchos tipos de claves de entidad, como números de pedido o números de empleado.

Procedimiento

Llamar a la función integrada RID en la condición de búsqueda de una instrucción SELECT, DELETE o UPDATE.

La función RID devuelve el RID de una fila, que puede utilizar para identificar de forma única una fila.

Restricción: Debido a que Db2 podría reutilizar los números RID cuando se ejecuta la utilidad REORG, la función RID podría devolver valores diferentes cuando se invoca para una fila varias veces.

Si especifica un RID y Db2 no puede localizar la fila a través del acceso directo a la fila, Db2 no cambia a otro método de acceso. En su lugar, Db2 no devuelve ninguna fila.

Conceptos relacionados

[Reglas para insertar datos en una columna ROWID](#)

Una columna ROWID contiene valores únicos que identifican cada fila de una tabla. Si puede insertar datos en una columna ROWID y cómo se insertan esos datos depende de cómo se define la columna.

[Acceso directo de fila \(PRIMARY_ACCESTYPE='D'\) \(Db2 Performance\)](#)

[Valores de ID de fila \(Db2 SQL\)](#)

Referencia relacionada

[RID función escalar \(Db2 SQL\)](#)

Formas de manipular datos LOB

Puede utilizar instrucciones SQL, localizadores LOB y variables de referencia de archivos LOB en sus programas de aplicación para manipular datos LOB almacenados en un Db2.

Por ejemplo, puede utilizar las siguientes instrucciones para extraer información sobre el departamento de un empleado del currículum:

```

EXEC SQL BEGIN DECLARE SECTION;
  char    employeeenum[6];
  long    deptInfoBeginLoc;
  long    deptInfoEndLoc;
  SQL TYPE IS CLOB_LOCATOR resume;
  SQL TYPE IS CLOB_LOCATOR deptBuffer;

```

```

EXEC SQL END DECLARE SECTION;
:
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT EMPNO, EMP_RESUME FROM EMP;
:
EXEC SQL FETCH C1 INTO :employeenum, :resume;
:
EXEC SQL SET :deptInfoBeginLoc =
  POSSTR(:resume.data, 'Department Information');

EXEC SQL SET :deptInfoEndLoc =
  POSSTR(:resume.data, 'Education');

EXEC SQL SET :deptBuffer =
  SUBSTR(:resume, :deptInfoBeginLoc,
  :deptInfoEndLoc - :deptInfoBeginLoc);

```

Estas declaraciones utilizan variables de host de tipo de datos localizador de objetos grandes (localizador LOB). Los localizadores LOB le permiten manipular datos LOB sin mover los datos LOB a variables de host. Al utilizar localizadores LOB, necesita cantidades de memoria mucho menores para sus programas.

También puede utilizar variables de referencia de archivos LOB cuando trabaje con datos LOB. Puede utilizar variables de referencia de archivos LOB para insertar datos LOB de un archivo en una tabla de tipo "Db2" o para recuperar datos LOB de una tabla de tipo "Db2".

Ejemplos de aplicaciones LOB : La siguiente tabla muestra los programas de ejemplo que Db2 proporciona para ayudarle a escribir aplicaciones para manipular datos LOB. Todos los programas residen en el conjunto de datos DSN1210.SDSNSAMP.

Tabla 76. Muestras LOB enviadas con Db2

Miembro que contiene código fuente	Idioma	Función
DSNTEJ7	JCL	Demuestra cómo crear una tabla con columnas LOB, una tabla auxiliar y un índice auxiliar. También muestra cómo cargar datos LOB de 32 KB o menos en un espacio de tabla LOB.
DSN8DLPL	C	Demuestra el uso de localizadores LOB y sentencias UPDATE para mover datos binarios a una columna de tipo BLOB.
DSN8DLRV	C	Demuestra cómo utilizar un localizador para manipular datos de tipo CLOB.
DSNTEP2	PL/I	Demuestra cómo asignar un SQLDA para filas que incluyen datos LOB y utilizar ese SQLDA para describir una instrucción de entrada y recuperar datos de columnas LOB.

Conceptos relacionados

Variables de referencia de archivo LOB

En una aplicación de host, puede utilizar una variable de referencia de archivo para insertar un valor LOB o XML desde un archivo en una tabla de Db2. También puede utilizar una variable de referencia de archivo para seleccionar un valor LOB o XML desde una tabla de Db2 en un archivo.

Fase 7: Acceso a los datos LOB (Db2 Installation and Migration)

Tareas relacionadas

Cómo guardar almacenamiento al manipular LOB mediante localizadores LOB

Los localizadores LOB le permiten manipular datos LOB sin recuperar datos de la tabla de Db2. Utilizando localizadores, evita la necesidad de asignar la gran cantidad de almacenamiento necesario para que las variables host contengan datos LOB.

Declaraciones de variable de referencia de archivo LOB, variable host LOB y localizador LOB

Cuando escribe aplicaciones para manipular datos LOB, necesita declarar variables hosto para contener los datos LOB o el localizador LOB. De lo contrario, debe declarar variables de referencia de archivo LOB para apuntar a datos LOB.

Puede declarar variables host LOB y localizadores LOB en ensamblador, C, C++, COBOL, Fortran, y PL/I. Además, puede declarar variables de referencia de archivos LOB en ensamblador, C, C++, COBOL y PL/I. Db2 genera una declaración equivalente que utiliza tipos de datos de lenguaje host para cada variable de host, localizador o variable de referencia de archivo de tipo SQL BLOB, CLOB o DBCLOB que usted declare. Cuando se hace referencia a una variable host LOB, un localizador LOB o una variable de referencia de archivo LOB en una instrucción SQL, se debe utilizar la variable especificada en la declaración de tipo SQL. Cuando se hace referencia a la variable host en una declaración de lenguaje host, se debe utilizar la variable que genera Db2 .

Db2 admite declaraciones de variables de host para LOB con longitudes de hasta 2 GB - 1. Sin embargo, el tamaño de una variable de host LOB está limitado por las restricciones del lenguaje de host y la cantidad de almacenamiento disponible para el programa.

Declare las variables de host LOB a las que hace referencia el precompilador en las sentencias SQL utilizando las palabras clave SQL TYPE IS BLOB, SQL TYPE IS CLOB o SQL TYPE IS DBCLOB.

Las variables de host LOB a las que se hace referencia únicamente mediante una instrucción SQL que utiliza un DESCRIPTOR deben utilizar la misma forma que la declarada por el precompilador. En este formato, la matriz de variables de host LOB consta de una longitud de 31 bits, seguida de los datos, seguida de otra longitud de 31 bits, seguida de los datos, y así sucesivamente. La longitud de 31 bits debe estar alineada con palabras completas.

Ejemplo : Supongamos que desea asignar una matriz LOB de 10 elementos, cada uno con una longitud de 5 bytes. Debe asignar los siguientes bytes para cada elemento, para un total de 120 bytes:

- 4 bytes para el entero de 31 bits
- 5 bytes para los datos
- 3 bytes para forzar la alineación de palabra completa

Los siguientes ejemplos muestran cómo declarar variables de host LOB en cada lenguaje compatible. En cada tabla, la columna de la izquierda contiene la declaración que codifica en su programa de aplicación. La columna de la derecha contiene la declaración que genera Db2 .

Declaraciones de variables de host LOB en ensamblador

La siguiente tabla muestra las declaraciones de lenguaje ensamblador para algunos tipos de LOB típicos.

Tabla 77. Ejemplo de declaraciones de variables LOB de ensamblador

Usted declara esta variable	Db2 genera esta variable
clob_var SQL TYPE IS CLOB 40000K	clob_var DS 0FL4 clob_var_length DS FL4 clob_var_data DS CL65535 ¹ ORG clob_var_data +(40960000-65535)
dbclob_var SQL TYPE IS DBCLOB 4000K	dbclob_var DS 0FL4 dbclob_var_length DS FL4 dbclob_var_data DS GL65534 ² ORG dbclob_var_data+(8192000-65534)

Tabla 77. Ejemplo de declaraciones de variables LOB de ensamblador (continuación)

Usted declara esta variable	Db2 genera esta variable
blob_var SQL TYPE IS BLOB 1M	blob_var DS 0FL4 blob_var_length DS FL4 blob_var_data DS CL65535 ¹ ORG blob_var_data+(1048476-65535)
clob_loc SQL TYPE IS CLOB_LOCATOR	clob_loc DS FL4
dbclob_loc SQL TYPE IS DBCLOB_LOCATOR	dbclob_loc DS FL4
blob_loc SQL TYPE IS BLOB_LOCATOR	blob_loc DS FL4
clob_file SQL TYPE IS CLOB_FILE	clob_file DS FL4
dbclob_file SQL TYPE IS DBCLOB_FILE	dbclob_file DS FL4
blob_file SQL TYPE IS BLOB_FILE	blob_file DS FL4

Notas:

1. Debido a que el lenguaje ensamblador permite declaraciones de caracteres de no más de 65535 bytes, Db2 separa en dos partes las declaraciones de lenguaje de host para variables de host BLOB y CLOB que son más largas de 65535 bytes.
2. Debido a que el lenguaje ensamblador permite declaraciones gráficas de no más de 65534 bytes, Db2 separa en dos partes las declaraciones del lenguaje host para las variables host DBCLOB que tienen más de 65534 bytes.

Declaraciones de variables de host LOB en C y C++

La siguiente tabla muestra las declaraciones de lenguaje C y C++ para algunos tipos de LOB típicos.

Tabla 78. Ejemplos de declaraciones de variables en lenguaje C

Usted declara esta variable	Db2 genera esta variable
SQL TYPE IS BLOB (1M) blob_var;	struct { unsigned long length; char data[1048576]; } blob_var;
SQL TYPE IS CLOB(400K) clob_var;	struct { unsigned long length; char data[409600]; } clob_var;
SQL TYPE IS DBCLOB (4000K) dbclob_var;	struct { unsigned long length; sqldbchar data[4096000]; } dbclob_var;
SQL TYPE IS BLOB_LOCATOR blob_loc;	unsigned long blob_loc;
SQL TYPE IS CLOB_LOCATOR clob_loc;	unsigned long clob_loc;
SQL TYPE IS DBCLOB_LOCATOR dbclob_loc;	unsigned long dbclob_loc;

Tabla 78. Ejemplos de declaraciones de variables en lenguaje C (continuación)

Usted declara esta variable	Db2 genera esta variable
SQL TYPE IS BLOB_FILE FBLOBhv;	<pre>#pragma pack(full) struct { unsigned long name_length; unsigned long data_length; unsigned long file_options; char name??(255??); } FBLOBhv ; #pragma pack(reset)</pre>
SQL TYPE IS CLOB_FILE FCLOBhv;	<pre>#pragma pack(full) struct { unsigned long name_length; unsigned long data_length; unsigned long file_options; char name??(255??); } FCLOBhv ; #pragma pack(reset)</pre>
SQL TYPE IS DBCLOB_FILE FDBCLOBhv;	<pre>#pragma pack(full) struct { unsigned long name_length; unsigned long data_length; unsigned long file_options; char name??(255??); } FDBCLOBhv ; #pragma pack(reset)</pre>

Declaraciones de variables de host LOB en COBOL

Las declaraciones que se generan para COBOL dependen de si se utiliza el precompilador Db2 o el coprocesador Db2 . La siguiente tabla muestra las declaraciones COBOL que el precompilador Db2 genera para algunos tipos de LOB típicos. Las declaraciones que genera el coprocesador de Db2 podrían ser diferentes.

Tabla 79. Ejemplos de declaraciones de variables COBOL por el precompilador de Db2

Usted declara esta variable	Db2 el precompilador genera esta variable
01 BLOB-VAR SQL TYPE IS BLOB(1M) .	01 BLOB-VAR. 49 BLOB-VAR-LENGTH PIC S9(9) COMP-5. 49 BLOB-VAR-DATA PIC X(1048576) .
01 CLOB-VAR SQL TYPE IS CLOB(40000K) .	01 CLOB-VAR. 49 CLOB-VAR-LENGTH PIC S9(9) COMP-5. 49 CLOB-VAR-DATA PIC X(40960000) .
01 DBCLOB-VAR SQL TYPE IS DBCLOB(4000K) .	01 DBCLOB-VAR. 49 DBCLOB-VAR-LENGTH PIC S9(9) COMP-5 49 DBCLOB-VAR-DATA PIC G(40960000) DISPLAY-1.
01 BLOB-LOC SQL TYPE IS BLOB-LOCATOR.	01 BLOB-LOC PIC S9(9) COMP-5.
01 CLOB-LOC SQLTYPE IS CLOB-LOCATOR.	01 CLOB-LOC PIC S9(9) COMP-5.

Tabla 79. Ejemplos de declaraciones de variables COBOL por el precompilador de Db2 (continuación)

Usted declara esta variable	Db2 el precompilador genera esta variable
01 DBCLOB-LOC SQLTYPE IS DBCLOB-LOCATOR.	01 DBCLOB-LOC PIC S9(9) COMP-5.
01 BLOB-FILE SQLTYPE IS BLOB-FILE.	01 BLOB-FILE. 49 BLOB-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC. 49 BLOB-FILE-DATA-LENGTH PIC S9(9) COMP-5. 49 BLOB-FILE-FILE-OPTION PIC S9(9) COMP-5. 49 BLOB-FILE-NAME PIC X(255) .
01 CLOB-FILE SQLTYPE IS CLOB-FILE.	01 CLOB-FILE. 49 CLOB-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC. 49 CLOB-FILE-DATA-LENGTH PIC S9(9) COMP-5. 49 CLOB-FILE-FILE-OPTION PIC S9(9) COMP-5. 49 CLOB-FILE-NAME PIC X(255) .
01 DBCLOB-FILE SQLTYPE IS DBCLOB-FILE.	01 DBCLOB-FILE. 49 DBCLOB-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC. 49 DBCLOB-FILE-DATA-LENGTH PIC S9(9) COMP-5. 49 DBCLOB-FILE-FILE-OPTION PIC S9(9) COMP-5. 49 DBCLOB-FILE-NAME PIC X(255) .

Declaraciones de variables de host LOB en Fortran

La siguiente tabla muestra declaraciones de Fortran para algunos tipos de LOB típicos.

Tabla 80. Ejemplos de declaraciones de variables de tipo Fortran

Usted declara esta variable	Db2 genera esta variable
SQL TYPE IS BLOB(1M) blob_var	CHARACTER blob_var(1048580) INTEGER*4 blob_var_LENGTH CHARACTER blob_var_DATA EQUIVALENCE(blob_var(1), + blob_var_LENGTH) EQUIVALENCE(blob_var(5), + blob_var_DATA)
SQL TYPE IS CLOB(40000K) clob_var	CHARACTER clob_var(4096004) INTEGER*4 clob_var_length CHARACTER clob_var_data EQUIVALENCE(clob_var(1), + clob_var_length) EQUIVALENCE(clob_var(5), + clob_var_data)
SQL TYPE IS BLOB_LOCATOR blob_loc	INTEGER*4 blob_loc
SQL TYPE IS CLOB_LOCATOR clob_loc	INTEGER*4 clob_loc

Declaraciones de variables de host LOB en PL/I

Las declaraciones que se generan para PL/I dependen de si se utiliza el precompilador Db2 o el coprocesador Db2. La siguiente tabla muestra las declaraciones PL/I que el precompilador Db2 genera

para algunos tipos de LOB típicos. Las declaraciones que genera el coprocesador de Db2 podrían ser diferentes.

Tabla 81. Ejemplos de declaraciones de variables PL/I por el precompilador de Db2

Usted declara esta variable	Db2 el precompilador genera esta variable
DCL BLOB_VAR SQL TYPE IS BLOB (1M);	DCL 1 BLOB_VAR, 2 BLOB_VAR_LENGTH FIXED BINARY(31), 2 BLOB_VAR_DATA, ¹ 3 BLOB_VAR_DATA1(32) CHARACTER(32767), 3 BLOB_VAR_DATA2 CHARACTER(1048576-32*32767);
DCL CLOB_VAR SQL TYPE IS CLOB (40000K);	DCL 1 CLOB_VAR, 2 CLOB_VAR_LENGTH FIXED BINARY(31), 2 CLOB_VAR_DATA, ¹ 3 CLOB_VAR_DATA1(1250) CHARACTER(32767), 3 CLOB_VAR_DATA2 CHARACTER(40960000-1250*32767);
DCL DBCLOB_VAR SQL TYPE IS DBCLOB (4000K);	DCL 1 DBCLOB_VAR, 2 DBCLOB_VAR_LENGTH FIXED BINARY(31), 2 DBCLOB_VAR_DATA, ² 3 DBCLOB_VAR_DATA1(250) GRAPHIC(16383), 3 DBCLOB_VAR_DATA2 GRAPHIC(4096000-250*16383);
DCL blob_loc SQL TYPE IS BLOB_LOCATOR;	DCL blob_loc FIXED BINARY(31);
DCL clob_loc SQL TYPE IS CLOB_LOCATOR;	DCL clob_loc FIXED BINARY(31);
DCL dbblob_loc SQL TYPE IS DBCLOB_LOCATOR;	DCL dbblob_loc FIXED BINARY(31);
DCL blob_file SQL TYPE IS BLOB_FILE;	DCL 1 blob_file, 2 blob_file_NAME_LENGTH BIN FIXED(31) ALIGNED, 2 blob_file_DATA_LENGTH BIN FIXED(31), 2 blob_file_FILE_OPTIONS BIN FIXED(31), 2 blob_file_NAME CHAR(255) ;
DCL clob_file SQL TYPE IS CLOB_FILE;	DCL 1 clob_file, 2 clob_file_NAME_LENGTH BIN FIXED(31) ALIGNED, 2 clob_file_DATA_LENGTH BIN FIXED(31), 2 clob_file_FILE_OPTIONS BIN FIXED(31), 2 clob_file_NAME CHAR(255) ;

Tabla 81. Ejemplos de declaraciones de variables PL/I por el precompilador de Db2 (continuación)

Usted declara esta variable	Db2 el precompilador genera esta variable
DCL dbclob_file SQL TYPE IS DBCLOBFILE;	DCL 1 dbclob_file, 2 dbclob_file_NAME_LENGTH BIN FIXED(31) ALIGNED, 2 dbclob_file_DATA_LENGTH BIN FIXED(31), 2 dbclob_file_OPTIONS BIN FIXED(31), 2 dbclob_file_NAME CHAR(255) ;

Notas:

1. Para las variables de host BLOB o CLOB que tienen una longitud superior a 32 767 bytes, Db2 crea declaraciones de lenguaje de host PL/I de la siguiente manera:
 - Si la longitud del LOB es superior a 32767 bytes y divisible por 32767, Db2 crea una matriz de cadenas de 32767 bytes. La dimensión de la matriz es *longitud/32767*.
 - Si la longitud del LOB es mayor que 32767 bytes pero no es divisible por 32767, Db2 crea dos declaraciones: La primera es una matriz de cadenas de 32767 bytes, donde la dimensión de la matriz, *n*, es *longitud/32767*. El segundo es una cadena de caracteres con *una longitud* de longitud - *n * 32767*.
2. Para las variables de host DBCLOB que tienen más de 16 383 caracteres de doble byte de longitud, Db2 crea declaraciones de lenguaje de host PL/I de la siguiente manera:
 - Si la longitud del LOB es superior a 16 383 caracteres y es divisible por 16 383, Db2 crea una matriz de cadenas de 16 383 caracteres. La dimensión de la matriz es *longitud/16383*.
 - Si la longitud del LOB es superior a 16 383 caracteres, pero no divisible de manera uniforme entre 16 383, Db2 crea dos declaraciones: la primera es una matriz de cadenas de 16 383 bytes, donde la dimensión de la matriz, *m*, es *longitud/16 383*. El segundo es una cadena de caracteres con *una longitud* de longitud - *m * 16383*.

Conceptos relacionados

[Variables de referencia de archivo LOB](#)

En una aplicación de host, puede utilizar una variable de referencia de archivo para insertar un valor LOB o XML desde un archivo en una tabla de Db2. También puede utilizar una variable de referencia de archivo para seleccionar un valor LOB o XML desde una tabla de Db2 en un archivo.

Tareas relacionadas

[Cómo guardar almacenamiento al manipular LOB mediante localizadores LOB](#)

Los localizadores LOB le permiten manipular datos LOB sin recuperar datos de la tabla de Db2. Utilizando localizadores, evita la necesidad de asignar la gran cantidad de almacenamiento necesario para que las variables host contengan datos LOB.

Materialización de LOB y XML

La materialización significa que Db2 pone los datos que selecciona en un almacenamiento intermedio a procesar. Esta acción puede ralentizar el rendimiento. Puesto que los valores LOB pueden ser muy grandes, Db2 evita la materialización de datos LOB hasta que es absolutamente necesario.

A partir de la versión DB2 10, la materialización LOB y XML se ha reducido o eliminado en la versión Db2 para varios casos locales y distribuidos, incluidos los servicios públicos (LOAD y cross-loader). Algunos de los casos en los que se ha eliminado o reducido la materialización incluyen la transmisión DRDA, el procesamiento de variables de referencia de archivos, la conversión CCSID y el procesamiento de recuperación distribuida de XML. Sin embargo, que los valores se materialicen o no y en qué medida también depende del número y el tamaño de cada LOB o XML.

Db2 almacena los valores LOB en un almacenamiento contiguo. Db2 debe materializar LOB cuando su programa de aplicación realice las siguientes acciones:

- Llama a una función definida por el usuario con un LOB como argumento
- Mueve un LOB dentro o fuera de un procedimiento almacenado
- Asigna una variable host LOB a una variable host de localizador LOB

La cantidad de almacenamiento que se utiliza para la materialización LOB y XML depende de una serie de factores, entre los que se incluyen:

- El tamaño de los LOB
- El número de LOB que deben materializarse en un extracto

Db2 carga LOB en grupos virtuales por encima de la barra. Si no hay espacio suficiente para la materialización de LOB, su aplicación recibe SQLCODE -904.

Aunque no se puede evitar por completo la materialización de LOB, se puede minimizar utilizando localizadores LOB, en lugar de variables host LOB en los programas de aplicación.

Tareas relacionadas

[Cómo guardar almacenamiento al manipular LOB mediante localizadores LOB](#)

Los localizadores LOB le permiten manipular datos LOB sin recuperar datos de la tabla de Db2. Utilizando localizadores, evita la necesidad de asignar la gran cantidad de almacenamiento necesario para que las variables host contengan datos LOB.

Cómo guardar almacenamiento al manipular LOB mediante localizadores LOB

Los localizadores LOB le permiten manipular datos LOB sin recuperar datos de la tabla de Db2. Utilizando localizadores, evita la necesidad de asignar la gran cantidad de almacenamiento necesario para que las variables host contengan datos LOB.

Acerca de esta tarea

Para recuperar datos LOB de una tabla de tipo "Db2", puede definir variables de host que sean lo suficientemente grandes como para contener todos los datos LOB. Esto requiere que su aplicación asigne grandes cantidades de almacenamiento y que Db2 mueva grandes cantidades de datos, lo que puede resultar ineficiente o poco práctico. En su lugar, puede utilizar los localizadores LOB. Los localizadores LOB le permiten manipular datos LOB sin recuperar datos de la tabla de Db2. El uso de localizadores LOB para la recuperación de datos LOB es una buena opción en las siguientes situaciones:

- Cuando trasladas solo una pequeña parte de un LOB a un programa de cliente
- Cuando el LOB completo no cabe en la memoria de la aplicación
- Cuando el programa necesita un valor LOB temporal de una expresión LOB, pero no necesita guardar el resultado
- Cuando el rendimiento es importante

Un localizador LOB está asociado con un valor o expresión LOB, no con una fila en una tabla de almacenamiento lógico (Db2) o una ubicación de almacenamiento físico en un espacio de tabla.

Por lo tanto, después de seleccionar un valor LOB utilizando un localizador, el valor en el localizador normalmente no cambia hasta que finaliza la unidad de trabajo actual. Sin embargo, el valor del LOB en sí mismo puede cambiar.

Si desea eliminar la asociación entre un localizador LOB y su valor antes de que finalice una unidad de trabajo, ejecute la sentencia FREE LOCATOR. Para mantener la asociación entre un localizador LOB y su valor después de que finalice la unidad de trabajo, ejecute la instrucción HOLD LOCATOR.

Después de ejecutar una instrucción HOLD LOCATOR, el localizador mantiene la asociación con el valor correspondiente hasta que se ejecute una instrucción FREE LOCATOR o hasta que finalice el programa.

Si ejecuta HOLD LOCATOR o FREE LOCATOR de forma dinámica, no puede utilizar EXECUTE IMMEDIATE.

Las aplicaciones que utilizan un gran número de localizadores, que se comprometen con poca frecuencia o no liberan explícitamente los localizadores, pueden utilizar grandes cantidades de valioso almacenamiento de espacio de direcciones de servicios de bases de datos (*ssnmDBM1*) y costes de CPU. Utilice con frecuencia COMMIT o FREE LOCATORS para evitar la falta de almacenamiento en el

espacio de direcciones de servicios de bases de datos (*ssnmDBM1*) y la falta de recursos de la CPU del sistema.

Para liberar los localizadores LOB después de recuperar sus valores LOB asociados, ejecute la instrucción FREE LOCATOR:

```
EXEC SQL FREE LOCATOR :LOCRES, :LOCHIST, :LOCPICText
```

Referencia relacionada

[FREE LOCATOR declaración \(Db2 SQL \)](#)

[HOLD LOCATOR declaración \(Db2 SQL \)](#)

Variables indicadoras y localizadores LOB

Db2 utiliza variables indicadoras para localizadores LOB de manera diferente a como utiliza variables indicadoras para variables host.

Para variables de host que no sean localizadores LOB, cuando seleccionas un valor nulo en una variable de host, Db2 asigna un valor negativo a la variable indicadora asociada. Sin embargo, para los localizadores LOB, Db2 utiliza las variables indicadoras de manera diferente. Un localizador LOB nunca es nulo. Cuando seleccionas una columna LOB utilizando un localizador LOB y la columna LOB contiene un valor nulo, el lenguaje de programación de aplicaciones (Db2) asigna un valor nulo a la variable indicadora asociada. El valor en el localizador LOB no cambia. En un entorno cliente/servidor, esta información nula se registra solo en el cliente.

Cuando utilice localizadores LOB para recuperar datos de columnas que pueden contener valores nulos, defina variables indicadoras para los localizadores LOB y compruebe las variables indicadoras después de recuperar datos en los localizadores LOB. Si una variable indicadora es nula después de una operación de obtención, no puede utilizar el valor en el localizador LOB.

Asignaciones válidas para localizadores LOB

Aunque normalmente se utilizan localizadores LOB para asignar y recuperar datos de columnas LOB, también se pueden utilizar para asignar datos a columnas que no son LOB.

Puede utilizar los localizadores LOB para realizar las siguientes asignaciones:

- Se puede asignar un localizador CLOB o DBCLOB a una columna CHAR, VARCHAR, GRAPHIC o VARGRAPHIC. Sin embargo, no puede obtener datos de columnas CHAR, VARCHAR, GRAPHIC o VARGRAPHIC en localizadores CLOB o DBCLOB.
- Se puede asignar un localizador BLOB a una columna BINARIA o VARBINARIA. Sin embargo, no puede obtener datos de una columna BINARY o VARBINARY en un localizador BLOB.

Cómo impedir la conversión de caracteres de los localizadores de LOB

En algunas situaciones, Db2 materializa el valor de LOB completo y lo convierte al esquema de codificación de una sentencia de SQL concreta. Este proceso adicional puede degradar el rendimiento y debe evitarse.

Acerca de esta tarea

Puede utilizar una sentencia VALUES INTO o SET para obtener los resultados de funciones que operan sobre localizadores LOB, como LENGTH o SUBSTR. Las sentencias VALUES INTO y SET se procesan en el esquema de codificación de la aplicación para el plan o paquete que contiene la sentencia. Si ese esquema de codificación es diferente del esquema de codificación de los datos LOB, el valor LOB completo se materializa y se convierte al esquema de codificación de la declaración. Este proceso de materialización y conversión puede causar una degradación del rendimiento.

Para evitar la conversión de caracteres, SELECCIONE de la tabla de muestra SYSIBM.SYSDUMMYA, SYSIBM.SYSDUMMYE o SYSIBM.SYSDUMMYU. Estas tablas ficticias realizan funciones similares a SYSIBM.SYSDUMMY1, y cada una de ellas está asociada a un esquema de codificación:

SYSIBM.SYSDUMMYA

ASCII

SYSIBM.SYSDUMMYE

EBCDIC

SYSIBM.SYSDUMMYU

Unicode

Al utilizar estas tablas, puede obtener el mismo resultado que con una instrucción VALUES INTO o SET.

Ejemplo

Supongamos que el esquema de codificación de la siguiente declaración es EBCDIC:

```
SET : unicode_hv = SUBSTR(:Unicode_lob_locator,X,Y);
```

Db2 debe materializar el LOB especificado por :Unicode_lob_locator y convertir todo ese LOB a EBCDIC antes de ejecutar la sentencia. Para evitar la materialización y la conversión, puede ejecutar la siguiente instrucción, que produce el mismo resultado pero se procesa mediante el esquema de codificación Unicode de la tabla:

```
SELECT SUBSTR(:Unicode_lob_locator,X,Y) INTO :unicode_hv
  FROM SYSIBM.SYSDUMMYU;
```

Aplazar la evaluación de una expresión LOB para mejorar el rendimiento

Db2 no mueve ningún byte de un valor LOB hasta que un programa asigne una expresión LOB a un destino. Cuando se utiliza un localizador LOB con funciones y operadores de cadena, Db2 no evalúa la expresión hasta el momento de la asignación. Esta evaluación diferida puede mejorar el rendimiento.

Acerca de esta tarea

El siguiente ejemplo es un programa en lenguaje C que aplaza la evaluación de una expresión LOB. El programa se ejecuta en un cliente y modifica los datos LOB en un servidor. El programa busca un currículum en particular (EMPNO = '000130') en la tabla EMP_RESUME. A continuación, utiliza localizadores LOB para reorganizar una copia del currículum (con EMPNO = 'A00130'). En la copia, la sección de información del departamento aparece al final del currículum. A continuación, el programa inserta la copia en EMP_RESUME sin modificar el currículum original.

Debido a que el programa de la siguiente figura utiliza localizadores LOB, en lugar de colocar los datos LOB en variables de host, no se mueve ningún dato LOB hasta que se ejecuta la instrucción INSERT. Además, no se transfieren datos LOB entre el cliente y el servidor.

```
EXEC SQL INCLUDE SQLCA;
/*
* Declare host variables */
EXEC SQL BEGIN DECLARE SECTION;
  char userid[9];
  char passwd[19];
  long      HV_START_DEPTINFO;
  long      HV_START_EDUC;
  long      HV_RETURN_CODE;
  SQL TYPE IS CLOB_LOCATOR HV_NEW_SECTION_LOCATOR;
  SQL TYPE IS CLOB_LOCATOR HV_DOC_LOCATOR1;
  SQL TYPE IS CLOB_LOCATOR HV_DOC_LOCATOR2;
  SQL TYPE IS CLOB_LOCATOR HV_DOC_LOCATOR3;
EXEC SQL END DECLARE SECTION;
/*
* Delete any instance of "A00130" from previous */
/* executions of this sample */
EXEC SQL DELETE FROM EMP_RESUME WHERE EMPNO = 'A00130';
/*
* Use a single row select to get the document */
EXEC SQL SELECT RESUME
  INTO :HV_DOC_LOCATOR1
```

1

2

```

        FROM EMP_RESUME
       WHERE EMPNO = '000130'
         AND RESUME_FORMAT = 'ascii';
/*****************************************************************/
/* Use the POSSTR function to locate the start of */
/* sections "Department Information" and "Education" */ 3
/*****************************************************************/
EXEC SQL SET :HV_START_DEPTINFO =
  POSSTR(:HV_DOC_LOCATOR1, 'Department Information');

EXEC SQL SET :HV_START_EDUC =
  POSSTR(:HV_DOC_LOCATOR1, 'Education');

/*****************************************************************/
/* Replace Department Information section with nothing */
/*****************************************************************/
EXEC SQL SET :HV_DOC_LOCATOR2 =
  SUBSTR(:HV_DOC_LOCATOR1, 1, :HV_START_DEPTINFO -1)
 || SUBSTR(:HV_DOC_LOCATOR1, :HV_START_EDUC);
/*****************************************************************/
/* Associate a new locator with the Department */
/* Information section */
/*****************************************************************/
EXEC SQL SET :HV_NEW_SECTION_LOCATOR =
  SUBSTR(:HV_DOC_LOCATOR1, :HV_START_DEPTINFO,
 :HV_START_EDUC -:HV_START_DEPTINFO);

/*****************************************************************/
/* Append the Department Information to the end */
/* of the resume */
/*****************************************************************/
EXEC SQL SET :HV_DOC_LOCATOR3 =
  :HV_DOC_LOCATOR2 || :HV_NEW_SECTION_LOCATOR;
/*****************************************************************/
/* Store the modified resume in the table. This is */
/* where the LOB data really moves. */ 4
/*****************************************************************/
EXEC SQL INSERT INTO EMP_RESUME VALUES ('A00130', 'ascii',
 :HV_DOC_LOCATOR3, DEFAULT);

/*****************************************************************/
/* Free the locators */
/*****************************************************************/
EXEC SQL FREE LOCATOR :HV_DOC_LOCATOR1, :HV_DOC_LOCATOR2, :HV_DOC_LOCATOR3; 5

```

Notas:

- 1** Declare aquí los localizadores LOB.
- 2** Esta sentencia SELECT asocia el localizador LOB (HV_DOC_LOCATOR1) con el valor de la columna RESUME para el número de empleado 000130.
- 3** Las siguientes cinco sentencias SQL utilizan localizadores LOB para manipular los datos del currículum sin moverlos.
- 4** La evaluación de las expresiones LOB en las sentencias anteriores se ha aplazado hasta la ejecución de esta sentencia INSERT.
- 5** Libera todos los localizadores LOB para liberarlos de sus valores asociados.

Variables de referencia de archivo LOB

En una aplicación de host, puede utilizar una variable de referencia de archivo para insertar un valor LOB o XML desde un archivo en una tabla de Db2. También puede utilizar una variable de referencia de archivo para seleccionar un valor LOB o XML desde una tabla de Db2 en un archivo.

Las variables de referencia de archivo son BLOB_FILE, CLOB_FILE o DBCLOB_FILE. Para COBOL, las variables de referencia de archivo son BLOB-FILE, CLOB-FILE o DBCLOB-FILE.

Cuando se utiliza una variable de referencia de archivo, se puede seleccionar o insertar un LOB o valor XML completo sin almacenamiento de aplicaciones contiguo para contener el LOB o valor XML completo. Las variables de referencia de archivos LOB mueven valores LOB o XML del servidor de base de datos a una aplicación o de una aplicación al servidor de base de datos sin pasar por la memoria de la aplicación. Además, las variables de referencia de archivos LOB eluden la limitación del lenguaje del host sobre el tamaño máximo permitido para que el almacenamiento dinámico contenga un valor LOB.

Puede declarar valores LOB o XML como variables de referencia de archivos LOB o matrices de referencia de archivos LOB para aplicaciones escritas en C, COBOL, PL/I y ensamblador. Las variables de referencia de archivo LOB no contienen datos LOB; representan un archivo que contiene datos LOB. Las consultas, actualizaciones e inserciones de bases de datos pueden utilizar variables de referencia de archivos para almacenar o recuperar valores de columna. Al igual que con otras variables de host, una variable de referencia de archivo LOB puede tener una variable indicadora asociada.

Db2-construcciones de variables de referencia de archivos LOB generados

Db2 , para cada variable de referencia de archivo LOB que una aplicación declara para un valor LOB o XML, genera una construcción equivalente que utiliza los tipos de datos del lenguaje host. Cuando una aplicación hace referencia a una variable de referencia de archivo LOB, debe utilizar la construcción equivalente que genera Db2 ; de lo contrario, el precompilador Db2 emite un error.

La construcción describe las siguientes propiedades del archivo:

Tipo de datos

BLOB, CLOB o DBCLOB. Esta propiedad se especifica cuando la variable se declara utilizando el tipo de datos BLOB_FILE, CLOB_FILE o DBCLOB_FILE.

Para COBOL, los tipos de datos son BLOB-FILE, CLOB-FILE o DBCLOB-FILE.

Dirección

Esta propiedad debe ser especificada por el programa de aplicación en tiempo de ejecución como parte de la propiedad de opción de archivo. La propiedad direction puede tener los siguientes valores:

Entrada

Se utiliza como fuente de datos en una instrucción EXECUTE, OPEN, UPDATE, INSERT, DELETE, SET o MERGE.

Salida

Se utiliza como destino de datos en una sentencia FETCH o SELECT INTO.

Nombre de archivo

Esta propiedad debe ser especificada por el programa de aplicación en tiempo de ejecución. La propiedad del nombre de archivo puede tener los siguientes valores:

- El nombre de ruta completa del archivo. Es lo que se recomienda.

Longitud de nombre de archivo

Esta propiedad debe ser especificada por el programa de aplicación en tiempo de ejecución.

Opciones de archivo

Una aplicación debe asignar una de las opciones de archivo a una variable de referencia de archivo antes de que la aplicación pueda utilizar esa variable. Las opciones de archivo se establecen mediante el valor INTEGER en un campo en la construcción de la variable de referencia de archivo. Se debe especificar alguna de estas opciones para cada variable de referencia a archivos:

- Entrada (desde la aplicación a la base de datos):

SQL_FILE_READ

Un archivo normal que se puede abrir, leer y cerrar.

- Salida (de la base de datos a la aplicación):

SQL_FILE_CREATE

Si el archivo no existe, se crea uno nuevo. Si el archivo ya existe, se devuelve un error.

SQL_FILE_OVERWRITE

Si el archivo no existe, se crea uno nuevo. Si el archivo ya existe, se sobrescribe.

SQL_FILE_APPEND

Si el archivo no existe, se crea uno nuevo. Si el archivo ya existe, el resultado se añade al archivo existente.

Longitud de datos

La longitud, en bytes, de los nuevos datos escritos en el archivo

Ejemplos de declaración de variables de referencia de archivos

Puede declarar una variable de referencia de archivo en C, COBOL y PL/I, y declarar la construcción de variable de referencia de archivo que genera Db2 .

Ejemplo C : Considere la siguiente declaración C:

```
EXEC SQL BEGIN DECLARE SECTION
  SQL TYPE IS CLOBFILE hv_text_file;
  CHAR hv_thesis_title[64];
EXEC SQL END DECLARE SECTION
```

Esa declaración da como resultado la siguiente construcción generada por Db2:

```
EXEC SQL BEGIN DECLARE SECTION
  /* SQL TYPE IS CLOBFILE hv_text_file; */
  struct {
    unsigned long name_length; // File name length
    unsigned long data_length; // Data length
    unsigned long file_options; // File options
    char name [255]; // File name
    } hv_text_file;
  char hv_thesis_title[64]
```

Con la construcción generada por Db2, puede utilizar el siguiente código para seleccionar de una columna CLOB en la base de datos en un nuevo archivo al que hace referencia :hv_text_file. El nombre del archivo debe ser una ruta absoluta.

```
strcpy(hv_text_file.name, "/u/gainer/papers/sigmod.94");
  hv_text_file.name_length = strlen("/u/gainer/papers/sigmod.94");
  hv_text_file.file_options = SQL_FILE_CREATE;

EXEC SQL SELECT CONTENT INTO :hv_text_file FROM PAPERS
  WHERE TITLE = 'The Relational Theory Behind Juggling';
```

Del mismo modo, puede utilizar el siguiente código para insertar los datos de un archivo al que hace referencia :hv_text_file en una columna CLOB. El nombre del archivo debe ser una ruta absoluta.

```
strcpy(hv_text_file.name, "/u/gainer/patents/chips.13");
  hv_text_file.name_length = strlen("/u/gainer/patents/chips.13");
  hv_text_file.file_options = SQL_FILE_READ;
strcpy(:hv_patent_title, "A Method for Pipelining Chip Consumption");

EXEC SQL INSERT INTO PATENTS(TITLE, TEXT)
  VALUES(:hv_patent_title, :hv_text_file);
```

Ejemplo de COBOL : Considere la siguiente declaración COBOL:

```
01 MY-FILE SQL TYPE IS BLOB-FILE
```

Esa declaración da como resultado la siguiente construcción generada por Db2:

```
01 MY-FILE.
  49 MY-FILE-NAME-LENGTH PIC S9(9) COMP-5.
  49 MY-FILE-DATA-LENGTH PIC S9(9) COMP-5.
  49 MY-FILE-FILE-OPTION PIC S9(9) COMP-5.
  49 MY-FILE-NAME PIC(255);
```

Ejemplo de PL/I : Considere la siguiente declaración de PL/I:

```
DCL MY_FILE SQL TYPE IS CLOBFILE
```

Esa declaración da como resultado la siguiente construcción generada por Db2:

```
DCL 1 MY_FILE,
  3 MY_FILE_NAME_LENGTH BINARY FIXED (31) UNALIGNED,
  3 MY_FILE_DATA_LENGTH BINARY FIXED (31) UNALIGNED,
  3 MY_FILE_FILE_OPTIONS BINARY FIXED (31) UNALIGNED,
  3 MY_FILE_NAME CHAR(255);
```

Para ver ejemplos de cómo declarar variables de referencia de archivo para datos XML en C, COBOL y PL/I, consulte [“Tipos de datos de variable host para datos XML en aplicaciones de SQL incorporado”](#) en la página 572.

Referenciar un objeto de secuencia

Un *objeto de secuencia* es un objeto definido por el usuario que genera una secuencia de valores numéricos de acuerdo con la especificación con la que se creó la secuencia. Puede recuperar el valor siguiente o anterior de la secuencia.

Acerca de esta tarea

Usted hace referencia a una secuencia utilizando la expresión NEXT VALUE (VALOR SIGUIENTE) o la expresión PREVIOUS VALUE (VALOR ANTERIOR), especificando el nombre de la secuencia:

- Una expresión NEXT VALUE genera y devuelve el siguiente valor para la secuencia especificada. Si una consulta contiene varias instancias de una expresión NEXT VALUE con el mismo nombre de secuencia, el valor de la secuencia se incrementa solo una vez para esa consulta. La sentencia ROLLBACK no tiene efecto sobre los valores ya generados.
- Una expresión PREVIOUS VALUE (VALOR ANTERIOR) devuelve el valor generado más recientemente para la secuencia especificada para una expresión NEXT VALUE (VALOR SIGUIENTE) anterior que especificaba la misma secuencia dentro del proceso de solicitud actual. El valor de la expresión VALOR ANTERIOR persiste hasta que se genera el siguiente valor para la secuencia, se elimina la secuencia o finaliza la sesión de la aplicación. La sentencia COMMIT y la sentencia ROLLBACK no tienen efecto sobre este valor.

Puede especificar una expresión NEXT VALUE o PREVIOUS VALUE en una cláusula SELECT, dentro de una cláusula VALUES de una operación de inserción, dentro de la cláusula SET de una operación de actualización (con ciertas restricciones) o dentro de una instrucción SET de *variable de host*.

Recuperación de miles de filas

Al recuperar un gran número de filas, tenga en cuenta las posibilidades de escalada de bloqueo y otros problemas de bloqueo.

Acerca de esta tarea

Pregunta : ¿Existen técnicas especiales para obtener y mostrar grandes volúmenes de datos?

Respuesta : No hay técnicas especiales, pero para un gran número de filas, la eficiencia puede ser muy importante. En particular, debe tener en cuenta las consideraciones de bloqueo, incluidas las posibilidades de escalada de bloqueo.

Si su programa permite la entrada desde un terminal antes de confirmar los datos y, por lo tanto, liberar los bloqueos, es posible que se produzca una pérdida significativa de concurrencia.

Determinar cuándo se cambió una fila

Si una tabla tiene una columna ROW CHANGE TIMESTAMP, puede determinar cuándo se cambió una fila.

Procedimiento

Emitir una sentencia SELECT con la columna ROW CHANGE TIMESTAMP en la lista de columnas.

Si una fila que cumple los requisitos no tiene un valor para la columna ROW CHANGE TIMESTAMP, Db2 devuelve la hora en que se actualizó la página en la que reside esa fila.

Ejemplo

Supongamos que emite las siguientes sentencias para crear, llenar y modificar una tabla:

```
CREATE TABLE T1 (C1 INTEGER NOT NULL);
INSERT INTO T1 VALUES (1);
ALTER TABLE T1 ADD COLUMN C2 NOT NULL GENERATED ALWAYS
    FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP;
SELECT T1.C2 FROM T1 WHERE T1.C1 = 1;
```

Debido a que la columna ROW CHANGE TIMESTAMP se añadió después de insertar los datos, la siguiente instrucción devuelve la hora en que se modificó la página por última vez:

```
SELECT T1.C2 FROM T1 WHERE T1.C1 = 1;
```

Supongamos que usted emite la siguiente declaración:

```
INSERT INTO T1(C1) VALUES (2);
```

Supongamos que esta fila se añade a la misma página que la primera fila. La siguiente declaración devuelve la hora en que se insertó el valor «2» en la tabla:

```
SELECT T1.C2 FROM T1 WHERE T1.C1 = 2;
```

Debido a que la fila con el valor «1» todavía no tiene un valor para la columna ROW CHANGE TIMESTAMP, la siguiente declaración sigue devolviendo la hora en que la página fue modificada por última vez, que en este caso es la hora en que se insertó el valor «2»:

```
SELECT T1.C2 FROM T1 WHERE T1.C1 = 1;
```

Referencia relacionada

[CREATE TABLE declaración \(Db2 SQL\)](#)

Comprobación de si una columna XML contiene un valor determinado

Puede determinar qué filas contienen cualquier fragmento de datos XML que especifique.

Procedimiento

Especifique el predicado XMLEXISTS en la cláusula WHERE de su declaración SQL.

Incluya los siguientes parámetros para el predicado XMLEXISTS:

- Expresión XPath incrustada en un literal de cadena de caracteres. Especifique una expresión XPath que identifique los datos XML que está buscando. Si el resultado de la expresión XPath es una secuencia vacía, XMLEXISTS devuelve false. Si el resultado no es vacío, XMLEXISTS devuelve true. Si la evaluación de la expresión XPath devuelve un error, XMLEXISTS devuelve un error.
- El nombre de la columna XML. Especifique este valor después de la palabra clave PASSING.

Ejemplo

Supongamos que desea devolver solo los pedidos de compra que tienen una dirección de facturación.

Supongamos que la columna XMLPO almacena los documentos XML de la orden de compra y que los nodos billTo dentro de estos documentos contienen direcciones de facturación. Puede utilizar la siguiente instrucción SELECT con el predicado XMLEXISTS:

```
SELECT XMLPO FROM T1
    WHERE XMLEXISTS ('declare namespace ipo="http://www.example.com/IP0";
        /ipo:purchaseOrder[billTo]
        PASSING XMLPO');
```

Referencia relacionada

[Predicado XMLEXISTS \(Db2 SQL\)](#)

Acceder a datos de Db2 que no están en una tabla

Puede acceder a datos de Db2 s que no están en una tabla devolviendo el valor de una expresión SQL en una variable de host.

Procedimiento

Para devolver el valor de una expresión SQL que no incluye el valor de una columna de tabla en la variable de host, utilice uno de los siguientes métodos:

- Utilice la instrucción SET de asignación de variables de host para establecer el contenido de una variable de host en el valor de una expresión.

```
EXEC SQL SET :hvrandval = RAND(:hvrand);
```

- Utilice la instrucción VALUES INTO para devolver el valor de una expresión en una variable de host.

```
EXEC SQL VALUES RAND(:hvrand)
INTO :hvrandval;
```

- Utilice la siguiente declaración para seleccionar la expresión de la tabla EBCDIC proporcionada por el Db2, denominada SYSIBM.SYSDUMMY1, que consta de una fila.

```
EXEC SQL SELECT RAND(:hvrand)
INTO :hvrandval
FROM SYSIBM.SYSDUMMY1;
```

Referencia relacionada

[SET assignment-statement declaración \(Db2 SQL\)](#)

[VALUES INTO declaración \(Db2 SQL\)](#)

[Tabla de catálogo SYSDUMMY1 \(Db2 SQL\)](#)

Garantizar que las consultas se realicen de manera satisfactoria

Es importante asegurarse de que las consultas individuales que se incluyen en su programa no ralentizan el rendimiento del mismo.

Antes de empezar

Consejo: Las capacidades de ajuste de consultas que pueden ayudarle con esta tarea, como *la explicación visual y el asesor de estadísticas*, están disponibles en [IBM Db2 Administration Foundation for z/OS](#) y [IBM Db2 for z/OS Developer Extension](#).

Procedimiento

Para garantizar que las consultas funcionen correctamente:

1. Ajuste cada consulta en su programa siguiendo las pautas generales de ajuste para escribir consultas eficientes. Para obtener más información, consulte [Escritura eficiente de consultas SQL \(Db2 Performance\)](#).
2. Si sospecha que una consulta no es tan eficiente como podría ser, supervise su rendimiento.

Puede utilizar varias funciones y técnicas diferentes para supervisar el rendimiento de SQL, incluida la instrucción EXPLAIN.

Conceptos relacionados

[Investigación del rendimiento de SQL utilizando EXPLAIN \(Db2 Performance\)](#)

[Interpretación del acceso a datos mediante el uso de EXPLAIN \(Db2 Performance\)](#)

Tareas relacionadas

[Investigación de problemas de la vía de acceso \(Db2 Performance\)](#)

Referencia relacionada

[EXPLAIN declaración \(Db2 SQL\)](#)

Elementos que deben incluirse en un programa DL/I por lotes

Al utilizar un programa DL/I por lotes con Db2, es necesario incluir ciertos elementos en el programa.

Un programa de DL/I por lotes puede emitir:

- Cualquier IMS llamada por lotes, excepto las llamadas ROLS, SETS y SYNC. Las llamadas ROLS y SETS proporcionan un procesamiento de punto de retroceso intermedio, que no es compatible con Db2 . La llamada SYNC proporciona el procesamiento del punto de confirmación sin identificar el punto de confirmación con un valor. IMS no permite una llamada SYNC en lote, y tampoco lo hace el soporte de lote DL/I de Db2 .

La emisión de una llamada ROLS, SETS o SYNC en un programa de aplicación provoca un fallo del sistema X'04E' con el código de motivo X'00D44057' en el registro 15.

- Llamadas de GSAM.
- IMS llamadas de servicios del sistema.
- Cualquier instrucción SQL, excepto COMMIT y ROLLBACK. IMS y CICS entornos no permiten esas sentencias SQL; sin embargo, IMS y CICS permiten ROLLBACK TO SAVEPOINT. Puede utilizar la IMS Llamada CHKP para confirmar los datos y la IMS ROLL o ROLB para revertir los cambios.

La emisión de una instrucción COMMIT provoca SQLCODE -925; la emisión de una instrucción ROLLBACK provoca SQLCODE -926. Esas declaraciones también devuelven SQLSTATE '2D521'.

- Cualquier llamada a un método de acceso estándar o tradicional (por ejemplo, QSAM, VSAM, etc.).

Las capacidades de reinicio para Db2 y IMS bases de datos, así como para conjuntos de datos secuenciales a los que se accede a través de GSAM, están disponibles a través de la IMS Función Checkpoint and Restart.

Db2 permite el acceso a datos de Db2 y DL/I mediante el uso de las siguientes Db2 y IMS instalaciones:

- IMS llamadas de sincronización, que comprometen y terminan de forma anormal unidades de recuperación
- Db2IMS, que maneja el protocolo de compromiso de dos fases y permite que ambos sistemas sincronicen una unidad de recuperación durante un reinicio después de un fallo
- El IMS registro, que se utiliza para registrar el instante de confirmación

En un entorno de intercambio de datos, DL/I batch admite adjuntos de grupo o de subgrupo. Puede especificar un nombre de archivo adjunto de grupo en lugar de un nombre de subsistema en el parámetro SSN del conjunto de datos de DDITV02 para el trabajo por lotes DL/I.

Requisitos para utilizar Db2 en un trabajo por lotes DL/I

El uso de Db2 en un trabajo por lotes DL/I requiere los siguientes cambios en el programa de aplicación y en el paso de trabajo JCL:

- Añada instrucciones SQL a su programa de aplicación para obtener acceso a datos de Db2 . A continuación, debe precompilar el programa de aplicación y vincular el DBRM resultante en un paquete.
- Antes de ejecutar el programa de aplicación, utilice JOBLIB, STEPLIB o el libro de enlaces para acceder a la biblioteca de carga de Db2 , de modo que se puedan cargar los módulos de Db2 .
- En un conjunto de datos especificado por una instrucción DD (DDITV02), especifique el nombre del programa y el nombre del plan para la aplicación, y el nombre de conexión para el trabajo por lotes DL/I.

En un conjunto de datos de entrada o en un miembro del subsistema, especifique información sobre la conexión entre Db2 y IMS. El nombre del conjunto de datos de entrada se especifica con

una instrucción DD (DDITV02). El nombre del miembro del subsistema se especifica mediante el parámetro SSM= en el procedimiento de invocación por lotes DL/I.

- Especifique opcionalmente un conjunto de datos de salida utilizando la sentencia DD (DDOTV02). Es posible que necesite este conjunto de datos para recibir mensajes del IMS función de adjuntos sobre hilos en duda e información de diagnóstico.

Consideraciones de diseño del programa para el uso de DL/I por lotes

Espacios de dirección en lote DL/I:

Una región de lote DL/I es independiente tanto de la IMS región de control y el CICS espacio de direcciones. La región del lote DL/I carga el código DL/I en la región de la aplicación junto con el programa de aplicación.

Compromisos en lote DL/I:

Realice IMS las solicitudes por lotes con frecuencia para no utilizar recursos durante mucho tiempo.

Declaraciones SQL y IMS llamadas en lotes DL/I:

Las aplicaciones por lotes DL/I no pueden utilizar las sentencias SQL COMMIT y ROLLBACK; de lo contrario, obtendrá un código de error SQL. Las aplicaciones por lotes DL/I tampoco pueden utilizar llamadas ROLS, SETS y SYNC; de lo contrario, el programa de aplicación se cierra de forma anormal.

Llamadas de punto de control en lote DL/I:

Escriba su programa con sentencias SQL y llamadas DL/I, y utilice llamadas de punto de control.

La frecuencia de los puntos de control depende del diseño de la aplicación. Todos los puntos de control que emite un programa de aplicación por lotes deben ser únicos. En un punto de control, se pierde la posición de DL/I, se cierran los cursos de Db2 (con la posible excepción de los cursos definidos como WITH HOLD), se liberan los bloqueos de duración de commit (de nuevo con algunas excepciones) y los cambios en la base de datos se consideran permanentes tanto para IMS y Db2.

Sincronización del programa de aplicación en DL/I batch:

Puede diseñar un programa de aplicación sin utilizar IMS puntos de control. En ese caso, si el programa se cierra de forma anormal antes de completarse, Db2 deshace cualquier actualización, y puede utilizar la IMS utilidad de reversión por lotes para revertir los cambios de DL/I.

También puede IMS retroceder dinámicamente las actualizaciones dentro del mismo trabajo. Debe especificar el parámetro BKO como 'Y' y asignar el IMS registro a DASD.

Podría tener un problema si el sistema en el que se ejecuta el trabajo falla después de que el programa finalice, pero antes de que termine el paso del trabajo. Si no llama a un punto de control antes de que finalice el programa, Db2 compromete la unidad de trabajo sin involucrar a IMS. Si el sistema falla antes de que DL/I confirme los datos, los datos de la e Db2 encia no están sincronizados con los cambios de DL/I. Si el sistema falla durante el procesamiento de la confirmación de la transacción (Db2), los datos de la confirmación de la transacción (Db2) podrían ser dudosos. Cuando reinicie el programa de aplicación, utilice la llamada XRST para obtener información de punto de control y resolver cualquier unidad de trabajo en duda (Db2).

Recomendación: Siempre emita un punto de control simbólico al final de cualquier trabajo de actualización para coordinar la entrega de la unidad de trabajo pendiente para IMS y Db2.

Consideraciones de punto de control y XRST en lote DL/I:

Si utiliza una llamada XRST, Db2 asume que cualquier punto de control que se emita es un punto de control simbólico. Las opciones de la llamada de punto de control simbólico difieren de las opciones de una llamada de punto de control básico. Usar la forma incorrecta de la llamada de punto de control puede causar problemas.

Si no utiliza una llamada XRST, Db2 asume que cualquier llamada de punto de control que se emita es un punto de control básico.

Para facilitar el reinicio, utilice caracteres EBCDIC para los ID de punto de control.

Cuando un programa de aplicación necesita ser reiniciable, debe utilizar llamadas simbólicas de punto de control y XRST. Si utiliza una llamada XRST, debe ser la primera IMS que se emite, y debe producirse antes de cualquier instrucción SQL. Además, solo debe utilizar una llamada XRST.

La llamada de sincronización falla en el lote DL/I:

Si el programa de aplicación contiene una IMS llamada de sincronización (CHKP, ROLB, ROLL o XRST), lo que provoca IMS emitir un código de estado incorrecto en el PCB, Db2 a el programa de aplicación. Asegúrese de probar estas llamadas antes de poner los programas en producción.

Conceptos relacionados

[Conjuntos de datos de entrada y salida para trabajos por lotes DL/I](#)

Los trabajos por lotes DL/I requieren un conjunto de datos de entrada con el nombre de definición de datos DDITV02 y un conjunto de datos de salida con el nombre de definición de datos DDOTV02.

[Coherencia de varios sistemas \(Db2 Administration Guide\)](#)

Tareas relacionadas

[Preparación de una aplicación para ejecutarse en Db2 for z/OS](#)

Para preparar y ejecutar aplicaciones que contengan sentencias SQL estáticas incrustadas o sentencias SQL dinámicas, debe procesar, compilar, editar vínculos y vincular las sentencias SQL.

Invocación de una función definida por el usuario

Puede utilizar una función definida por el usuario siempre que pueda utilizar una función incorporada.

Puede invocar una función escalar definida por el usuario de origen o externa en una instrucción SQL siempre que utilice una expresión. Para una función de tabla, puede invocar la función definida por el usuario solo en la cláusula FROM de una instrucción SELECT. La instrucción SQL invocada puede estar en un programa independiente, un procedimiento almacenado, un cuerpo de activador u otra función definida por el usuario.

Recomendaciones para invocar funciones definidas por el usuario:

Invocar funciones definidas por el usuario con acciones externas y funciones definidas por el usuario no deterministas desde listas de selección: es preferible invocar funciones definidas por el usuario con acciones externas desde una lista de selección y funciones definidas por el usuario no deterministas desde una lista de selección que invocar estas funciones definidas por el usuario desde un predicado.

La ruta de acceso que Db2 elige para un predicado determina si se invoca una función definida por el usuario en ese predicado. Para asegurarse de que Db2 ejecute la acción externa para cada fila de la tabla de resultados, ponga la invocación de la función definida por el usuario en la lista SELECT.

Invocar una función no determinista definida por el usuario desde un predicado puede producir resultados no deseados. El siguiente ejemplo demuestra esta idea.

Supongamos que ejecuta esta consulta:

```
SELECT COUNTER(), C1, C2 FROM T1 WHERE COUNTER() = 2;
```

La tabla de T1 o tiene este aspecto:

C1	C2
--	--
1	b
2	c
3	a

CONTADOR es una función definida por el usuario que incrementa una variable en el bloc de notas cada vez que se invoca.

Db2 invoca una instancia de COUNTER en el predicado 3 veces. Supongamos que COUNTER se invoca primero para la fila 1, luego para la fila 2 y finalmente para la fila 3. Entonces COUNTER devuelve 1 para la fila 1, 2 para la fila 2 y 3 para la fila 3. Por lo tanto, la fila 2 satisface el predicado WHERE COUNTER()=2, por lo que Db2 evalúa la lista SELECT para la fila 2. Db2 utiliza una instancia de COUNTER diferente en la

lista de selección de la instancia en el predicado. Como la instancia de COUNTER en la lista de selección se invoca solo una vez, devuelve un valor de 1. Por lo tanto, el resultado de la consulta es:

COUNTER()	C1	C2
1	2	c

Este no es el resultado que cabría esperar.

Los resultados pueden diferir aún más, dependiendo del orden en el que Db2 recupera las filas de la tabla. Supongamos que se define un índice ascendente en la columna C2. Db2 , entonces, recupera primero la fila 3, después la fila 1 y, por último, la fila 2. Esto significa que la fila 1 satisface el predicado WHERE COUNTER()=2. El valor de COUNTER en la lista de selección es de nuevo 1, por lo que el resultado de la consulta en este caso es:

COUNTER()	C1	C2
1	1	b

Comprender la interacción entre los cursores desplazables y las funciones no deterministas definidas por el usuario o las funciones definidas por el usuario con acciones externas : Cuando se utiliza un cursor desplazable, es posible recuperar la misma fila varias veces mientras el cursor está abierto. Si la lista de selección de la instrucción SELECT del cursor contiene una función definida por el usuario, dicha función se ejecuta cada vez que se recupera una fila. Por lo tanto, si la función definida por el usuario tiene una acción externa y recupera la misma fila varias veces, la acción externa se ejecuta varias veces para esa fila.

Una situación similar ocurre con los cursores desplazables y las funciones no deterministas. El resultado de una función no determinista definida por el usuario puede ser diferente cada vez que se ejecuta la función definida por el usuario. Si la lista de selección de un cursor desplazable contiene una función no determinista definida por el usuario, y utiliza ese cursor para recuperar la misma fila varias veces, los resultados pueden diferir cada vez que recupera la fila.

Una función no determinista definida por el usuario en el predicado de una instrucción SELECT de un cursor desplazable no cambia el resultado del predicado mientras el cursor está abierto. Db2 evalúa una función definida por el usuario en el predicado solo una vez mientras el cursor está abierto.

Conceptos relacionados

[Terminación anómala de una función externa definida por el usuario](#)

Si una función externa definida por el usuario termina de forma anómala, el programa recibe SQLCODE -430 para invocar la sentencia.

[Invocación de funciones \(Db2 SQL\)](#)

[Referencia relacionada](#)

[cláusula-from \(Db2 SQL\)](#)

Cómo determina Db2 la autorización para invocar funciones definidas por el usuario

Tanto la autorización utilizada para invocar una función definida por el usuario (UDF) como la autorización utilizada para ejecutar cada instrucción SQL en la función influyen en el procesamiento de una UDF.

La autorización que se requiere para invocar una función definida por el usuario depende de si la UDF se invoca de forma estática o dinámica:

- Para invocaciones estáticas, se utiliza la autorización del propietario del paquete que contiene la invocación de la UDF.
- Para invocaciones dinámicas, la opción DYNAMICRULES bind del paquete que contiene la invocación de lo definido por el usuario determina la autorización que se utiliza. Para obtener más información sobre cómo Db2 aplica la opción de enlace DYNAMICRULES, consulte [DYNAMICRULES opción bind \(comandos Db2 \)](#).

Del mismo modo, la autorización que utiliza Db2 para procesar cada instrucción SQL dentro de una UDF depende de si la instrucción es una instrucción SQL estática o dinámica:

- Para las sentencias SQL estáticas, se utiliza la autorización del propietario de la UDF.
- Para las sentencias SQL dinámicas, la opción DYNAMICRULES de la sentencia CREATE FUNCTION determina la autorización que se utiliza.

Conceptos relacionados

[Invocación de funciones \(Db2 SQL\)](#)

[Privilegios necesarios para la ejecución de rutinas \(Managing Security\)](#)

Tareas relacionadas

[Ejemplos de concesión de privilegios para rutinas \(Gestión de la seguridad\)](#)

Referencia relacionada

[Instrucción CREATE FUNCTION \(resumen\) \(Db2 SQL\)](#)

Asegurarse de que Db2 ejecute la función definida por el usuario prevista

Pueden existir varias funciones con el mismo nombre en el mismo esquema o en esquemas distintos. Puede realizar ciertas acciones para que Db2 elija la función correcta a ejecutar.

Acerca de esta tarea

La combinación del nombre de la función y la lista de parámetros forman *la firma* que utiliza Db2 para identificar una función. Para obtener información detallada sobre las reglas y el proceso que utiliza Db2 para identificar la función a invocar, consulte [Resolución de función \(Db2 SQL\)](#).

Si las firmas de dos funciones coinciden, incluidas las funciones integradas y las definidas por el usuario, debe tomar las medidas adecuadas para asegurarse de que Db2 invoque la función correcta prevista.

Procedimiento

Para simplificar la resolución de funciones integradas y definidas por el usuario, utilice las siguientes técnicas:

- Cuando invoque una función, utilice el nombre cualificado.

Esto hace que Db2 busque funciones solo en el esquema que especifique. Este enfoque tiene las siguientes ventajas:

- Db2 es menos probable que elija una función que no tenía intención de utilizar. Varias funciones podrían ajustarse igualmente bien a la invocación. Db2 selecciona la función cuyo nombre de esquema aparece en primer lugar en el Vía SQL, que podría no ser la función que usted desea.
- El número de funciones candidatas es menor, por lo que la resolución de funciones de Db2 lleva menos tiempo.
- Emitir parámetros en una invocación de función definida por el usuario a los tipos en la definición de función definida por el usuario. Por ejemplo, si un parámetro de entrada para la función definida por el usuario FUNC se define como DECIMAL(13,2), y el valor que desea pasar a la función definida por el usuario es un valor entero, convierta el valor entero a DECIMAL(13,2):

Por ejemplo, si un parámetro de entrada para la función definida por el usuario FUNC se define como DECIMAL(13,2), y el valor que desea pasar a la función definida por el usuario es un valor entero, convierta el valor entero a DECIMAL(13,2):

```
SELECT FUNC(CAST (INTCOL AS DECIMAL(13,2))) FROM T1;
```

- Utilice el tipo de datos BIGINT para los parámetros numéricos en una función definida por el usuario.

Cuando invoque la función, puede pasar valores SMALLINT, INTEGER o BIGINT. Si utiliza SMALLINT o REAL como tipo de parámetro, debe pasar parámetros del mismo tipo. Por ejemplo, si la función definida por el usuario FUNC se define con un parámetro de tipo SMALLINT, solo una invocación con

un parámetro de tipo SMALLINT se resuelve correctamente. La siguiente llamada no se resuelve en FUNC porque el constante 123 es de tipo INTEGER, no SMALLINT:

```
SELECT FUNC(123) FROM T1;
```

- Evite definir parámetros de cadena de función definidos por el usuario con tipos de cadena de longitud fija.

Si define un parámetro con un tipo de cadena de longitud fija (CHAR, GRAPHIC o BINARY), puede invocar la función definida por el usuario solo con un parámetro de cadena de longitud fija. Sin embargo, si define el parámetro con un tipo de cadena de longitud variable (VARCHAR, VARGRAPHIC o VARBINARY), puede invocar la función definida por el usuario con un parámetro de cadena de longitud fija o con un parámetro de cadena de longitud variable.

Si debe definir parámetros para una función definida por el usuario como CHAR o BINARY, y llama a la función definida por el usuario desde un programa C o un procedimiento SQL, debe convertir los valores de los parámetros correspondientes en la invocación de la función definida por el usuario a CHAR o BINARY para asegurarse de que Db2 invoque la función correcta. Por ejemplo, supongamos que un programa C llama a la función definida por el usuario CVRTNUM, que toma un parámetro de entrada de tipo CHAR(6). Supongamos también que declara la variable de host empnumbr como char empnumbr[6]. Cuando invoques CVRTNUM, lanza empnumbr a CHAR:

```
UPDATE EMP  
SET EMPNO=CVRTNUM(CHAR(:empnumbr))  
WHERE EMPNO = :empnumbr;
```

Conceptos relacionados

[Funciones \(Db2 SQL\)](#)

Cómo resuelve Db2 las funciones

La resolución de funciones es el proceso mediante el cual el motor de ejecución de aplicaciones (Db2) determina qué función definida por el usuario o función integrada se debe ejecutar. Es necesario comprender el proceso de las resoluciones de función que usa Db2 para garantizar que el usuario invoca la función definida por el usuario que desea invocar.

En un subsistema de Db2, pueden existir varias funciones definidas por el usuario con el mismo nombre pero con diferentes números o tipos de parámetros. Varias funciones definidas por el usuario con el mismo nombre pueden tener el mismo número de parámetros, siempre y cuando los tipos de datos de cualquiera de los primeros 30 parámetros sean diferentes. Además, varias funciones definidas por el usuario pueden tener el mismo nombre que una función incorporada. Cuando invoca una función, Db2 debe determinar qué función definida por el usuario o función integrada ejecutar.

Db2 realiza estos pasos para la resolución de funciones:

1. Determina si alguna instancia de función es candidata para su ejecución. Si no existen candidatos, Db2 emite un mensaje de error SQL.
2. Compara los tipos de datos de los parámetros de entrada para determinar qué candidatos se ajustan mejor a la invocación.

Db2 no compara tipos de datos para parámetros de entrada que son marcadores de parámetros sin tipo.

Para una invocación de función calificada, si no hay marcadores de parámetro en la invocación, el resultado de la comparación de tipo de datos es un mejor ajuste. La mejor opción es la elegida para la ejecución. Si hay marcadores de parámetros en la invocación, puede haber más de una mejor opción. Db2 emite un error si hay más de una mejor opción.

Para una invocación de función no cualificada, Db2 podría encontrar múltiples mejores ajustes porque el mismo nombre de función con los mismos parámetros de entrada puede existir en diferentes esquemas, o porque hay marcadores de parámetros en la invocación.

3. Si dos o más candidatos se ajustan igualmente bien a la invocación de función no cualificada porque el mismo nombre de función con los mismos parámetros de entrada existe en diferentes esquemas, Db2 elige la función definida por el usuario cuyo nombre de esquema es el más antiguo en el Vía SQL.

Por ejemplo, supongamos que las funciones SCHEMA1.X y SCHEMA2.X se ajustan igualmente bien a una invocación de función. Supongamos que el Vía SQL es:

```
"SCHEMA2", "SYSPROC", "SYSIBM", "SCHEMA1", "SYSFUN"
```

Db2 , a continuación, elige la función SCHEMA2.X.

Si dos o más candidatos se ajustan igualmente bien a la invocación de función no cualificada porque la invocación de función contiene marcadores de parámetro, Db2 emite un error.

El resto de esta sección trata sobre los detalles del proceso de resolución de funciones y ofrece sugerencias sobre cómo asegurarse de que Db2 elija la función correcta.

Cómo elige a los candidatos para los puestos de trabajo en Db2 :

Una instancia de una función definida por el usuario es candidata para su ejecución solo si cumple todos los criterios siguientes:

- Si el nombre de la función está calificado en la invocación, el esquema de la instancia de la función coincide con el esquema de la invocación de la función.

Si el nombre de la función no está calificado en la invocación, el esquema de la instancia de la función coincide con un esquema en el Vía SQL del invocador.

- El nombre de la instancia de la función coincide con el nombre en la invocación de la función.
- El número de parámetros de entrada en la instancia de la función coincide con el número de parámetros de entrada en la invocación de la función.
- El invocador de la función está autorizado a ejecutar la instancia de la función.
- El tipo de cada uno de los parámetros de entrada en la invocación de la función coincide o es *promocionable* al tipo del parámetro correspondiente en la instancia de la función.

Si un parámetro de entrada en la invocación de la función es un marcador de parámetro sin tipo, Db2 considera que ese parámetro coincide o es promocionable.

Para una invocación de función que pase una tabla de transición, el tipo de datos, la longitud, la precisión y la escala de cada columna de la tabla de transición deben coincidir exactamente con el tipo de datos, la longitud, la precisión y la escala de cada columna de la tabla que se nombra en la definición de instancia de función. Para obtener información sobre las tablas de transición, consulte [“Creación de un desencadenante” en la página 160](#).

- La marca de tiempo de creación de una función definida por el usuario debe ser anterior a la marca de tiempo BIND o REBIND del paquete o plan en el que se invoca la función definida por el usuario.

Si la comprobación de autorización de Db2 está en vigor y Db2 realiza una reasociación automática en un plan o paquete que contiene una invocación de función definida por el usuario, cualquier función definida por el usuario que se haya creado después de la BIND o REBIND original del plan o paquete que la invoca no es candidata para su ejecución.

Si utiliza una rutina de salida de autorización de control de acceso, algunas funciones definidas por el usuario que no eran candidatas a ejecución antes del BIND o REBIND original del plan o paquete de invocación podrían convertirse en candidatas a ejecución durante el rebind automático del plan o paquete de invocación.

Si se invoca una función definida por el usuario durante una reasociación automática, y esa función definida por el usuario se invoca desde un cuerpo de activación y recibe una tabla de transición, entonces la forma de la función invocada que utiliza Db2 para la selección de funciones incluye solo las columnas de la tabla de transición que existían en el momento de la BIND o REBIND original del paquete o plan para el programa invocador.

Durante una reasignación automática, Db2 no tiene en cuenta las funciones integradas para la resolución de funciones si dichas funciones integradas se introdujeron en una versión posterior de Db2 a la versión en la que se produjo la ASIGNACIÓN o REASIGNACIÓN del plan o paquete de invocación.

Cuando vinculas o revinculas explícitamente un plan o paquete, el plan o paquete recibe un marcador de dependencia de versión. Cuando Db2 realiza una reasignación automática de una consulta que contiene una invocación de función, una función incorporada es candidata a la resolución de funciones solo si el marcador de dependencia de versión de la función incorporada es igual o inferior al marcador de dependencia de versión del plan o paquete que contiene la invocación de función.

Ejemplo : Supongamos que en esta declaración, el tipo de datos de A es SMALLINT:

```
SELECT USER1.ADDTWO(A) FROM TABLEA;
```

Se definen dos instancias de USER1.ADDTWO : una con un parámetro de entrada de tipo INTEGER y otra con un parámetro de entrada de tipo DECIMAL. Ambas instancias de función son candidatas para la ejecución porque el tipo SMALLINT es promocionable a INTEGER o DECIMAL. Sin embargo, la instancia con el tipo INTEGER es más adecuada porque INTEGER está más arriba en la lista que DECIMAL.

Cómo elige Db2 la función más adecuada entre las candidatas:

Más de una instancia de función podría ser candidata para la ejecución. En ese caso, Db2 determina qué instancias de función son las más adecuadas para la invocación comparando los tipos de datos de los parámetros.

Si los tipos de datos de todos los parámetros en una instancia de función son los mismos que los de la invocación de la función, esa instancia de función es la más adecuada. Si no existe una coincidencia exacta, Db2 compara los tipos de datos en las listas de parámetros de izquierda a derecha, utilizando este método:

1. Db2 compara los tipos de datos del primer parámetro en la invocación de la función con el tipo de datos del primer parámetro en cada instancia de la función.

Si el primer parámetro en la invocación es un marcador de parámetro sin tipo, Db2 no realiza la comparación.

2. Para el primer parámetro, si una instancia de función tiene un tipo de datos que se ajusta mejor a la invocación de la función que los tipos de datos de las otras instancias, esa función es la que mejor se ajusta.
3. Si los tipos de datos del primer parámetro son los mismos para todas las instancias de la función, o si el primer parámetro en la invocación de la función es un marcador de parámetro sin tipo, Db2 repite este proceso para el siguiente parámetro. Db2 continúa este proceso para cada parámetro hasta que encuentra el que mejor se ajusta.

Ejemplo de resolución de funciones : Supongamos que un programa contiene la siguiente declaración:

```
SELECT FUNC(VCHARCOL,SMINTCOL,DECCOL) FROM T1;
```

En la función definida por el usuario FUNC, VCHARCOL tiene el tipo de datos VARCHAR, SMINTCOL tiene el tipo de datos SMALLINT y DECCOL tiene el tipo de datos DECIMAL. Supongamos también que dos instancias de función con las siguientes definiciones cumplen los criterios adecuados y, por lo tanto, son candidatas para su ejecución.

```
Candidate 1:  
CREATE FUNCTION FUNC(VARCHAR(20),INTEGER,DOUBLE)  
RETURNS DECIMAL(9,2)  
EXTERNAL NAME 'FUNC1'  
PARAMETER STYLE SQL  
LANGUAGE COBOL;
```

```
Candidate 2:  
CREATE FUNCTION FUNC(VARCHAR(20),REAL,DOUBLE)  
RETURNS DECIMAL(9,2)  
EXTERNAL NAME 'FUNC2'  
PARAMETER STYLE SQL  
LANGUAGE COBOL;
```

Db2 compara el tipo de datos del primer parámetro en la invocación de la función definida por el usuario con los tipos de datos de los primeros parámetros en las funciones candidatas. Debido a que el primer parámetro en la invocación tiene el tipo de datos VARCHAR, y ambas funciones candidatas también tienen el tipo de datos VARCHAR, Db2 no puede determinar la mejor candidata basándose en el primer parámetro. Por lo tanto, Db2 compara los tipos de datos de los segundos parámetros.

El tipo de datos del segundo parámetro en la invocación es SMALLINT. INTEGER, que es el tipo de datos del candidato 1, se ajusta mejor a SMALLINT que REAL, que es el tipo de datos del candidato 2. Por lo tanto, el candidato 1 es la opción e Db2 a para la ejecución.

Conceptos relacionados

[Promoción de tipos de datos \(Db2 SQL\)](#)

Tareas relacionadas

[Creación de un desencadenante](#)

Un disparador es un conjunto de instrucciones SQL que se ejecutan cuando se produce un determinado evento en una tabla o vista. Utilice activadores para controlar los cambios en bases de datos Db2.

Los desencadenantes son más eficaces que las restricciones porque pueden supervisar un amplio rango de cambios y pueden realizar las acciones siguientes: En este tema se describe el soporte para desencadenantes avanzados.

Información relacionada

[Rutinas de salida \(Db2 Administration Guide\)](#)

Comprobación de cómo resuelve Db2 las funciones utilizando DSN_FUNCTION_TABLE

Dado que varias funciones definidas por el usuario pueden tener el mismo nombre, debe asegurarse de que Db2 invoca la función que deseaba invocar. Una forma de comprobar que se ha invocado la función correcta es utilizar una tabla de funciones denominada DSN_FUNCTION_TABLE.

Procedimiento

Para comprobar cómo Db2 resuelve una función mediante DSN_FUNCTION_TABLE:

1. Si *your(userID).DSN_FUNCTION_TABLE* no existe, cree esta tabla siguiendo las instrucciones de [DSN_FUNCTION_TABLE \(Db2 Performance\)](#).
2. Rellene *your(userID).DSN_FUNCTION_TABLE* con información sobre qué funciones son invocadas por una instrucción SQL concreta realizando una de las siguientes acciones:
 - Ejecutar la sentencia EXPLAIN en la sentencia SQL.
 - Asegúrese de que el programa que contiene la instrucción SQL está vinculado con EXPLAIN(YES) y ejecute el programa.

Db2 pone una fila en *your(userID).DSN_FUNCTION_TABLE* para cada función a la que se hace referencia en cada instrucción SQL.

3. Compruebe las filas que se añadieron a *your(userID).DSN_FUNCTION_TABLE* para asegurarse de que se invocó la función adecuada. Utilice las siguientes columnas para ayudarle a encontrar las filas aplicables: QUERYNO, APPLNAME, PROGNAM, COLLID y EXPLAIN_TIME.

Referencia relacionada

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2 \)](#)

[EXPLAIN declaración \(Db2 SQL\)](#)

Restricciones al pasar argumentos con tipos distintos a funciones

Db2 , al exigir una tipificación estricta al pasar argumentos a una función, obliga a seguir ciertas reglas al pasar argumentos con tipos distintos a las funciones.

Respete las siguientes normas:

- Puede pasar argumentos que tengan tipos distintos a una función si se cumple alguna de las siguientes condiciones:

- Se define una versión de la función que acepta esos tipos distintos.

Esto también se aplica a los operadores de infijos. Si desea utilizar uno de los cinco operadores infijos integrados (||, /, *, +, -) con sus tipos distintos, debe definir una versión de ese operador que acepte los tipos distintos.

- Puede convertir sus tipos distintos en los tipos de argumento de la función.

- Si se pasan argumentos a una función que solo acepta tipos distintos, los argumentos que se pasen deben tener los mismos tipos distintos que en la definición de la función. Si los tipos son diferentes, debe lanzar sus argumentos a los distintos tipos en la definición de la función.

Si pasa constantes o variables de host a una función que solo acepta tipos distintos, debe convertir las constantes o variables de host a los tipos distintos que acepta la función.

Los siguientes ejemplos demuestran cómo utilizar tipos distintos como argumentos en invocaciones de funciones.

Ejemplo: Definir una función con tipos distintos como argumentos : Supongamos que desea invocar la función integrada HOUR con un tipo distinto que se define así:

```
CREATE DISTINCT TYPE FLIGHT_TIME AS TIME;
```

La función HOUR solo toma el tipo de datos TIME o TIMESTAMP como argumento, por lo que necesita una función fuente basada en la función HOUR que acepte el tipo de datos FLIGHT_TIME. Podría declarar una función como esta:

```
CREATE FUNCTION HOUR(FLIGHT_TIME)
RETURNS INTEGER
SOURCE SYSIBM.HOUR(TIME);
```

Ejemplo: Convertir argumentos de función a tipos aceptables : Otra forma de invocar la función HOUR es convertir el argumento de tipo FLIGHT_TIME al tipo de datos TIME antes de invocar la función HOUR. Supongamos que la tabla FLIGHT_INFO contiene la columna DEPARTURE_TIME, que tiene el tipo de datos FLIGHT_TIME, y que desea utilizar la función HOUR para extraer la hora de salida de la hora de salida. Puede convertir DEPARTURE_TIME al tipo de datos TIME y, a continuación, invocar la función HOUR:

```
SELECT HOUR(CAST(DEPARTURE_TIME AS TIME)) FROM FLIGHT_INFO;
```

Ejemplo: Uso de un operador infijo con argumentos de tipo distinto : Supongamos que desea sumar dos valores de tipo US_DOLLAR. Antes de poder hacer esto, debe definir una versión de la función + que acepte valores de tipo US_DOLLAR como operandos:

```
CREATE FUNCTION "+"(US_DOLLAR,US_DOLLAR)
RETURNS US_DOLLAR
SOURCE SYSIBM."+"(DECIMAL(9,2),DECIMAL(9,2));
```

Dado que el tipo US_DOLLAR se basa en el tipo DECIMAL(9,2), la función fuente debe ser la versión de + con argumentos de tipo DECIMAL(9,2).

Ejemplo: Convertir constantes y variables de host a tipos distintos para invocar una función definida por el usuario : Supongamos que la función CDN_TO_US se define así:

```
CREATE FUNCTION EURO_TO_US(EURO)
RETURNS US_DOLLAR
EXTERNAL_NAME 'CDNCVT'
PARAMETER_STYLE SQL
LANGUAGE C;
```

Esto significa que EURO_TO_US solo acepta el tipo EURO como entrada. Por lo tanto, si desea llamar a CDN_TO_US con un argumento constante o variable de host, debe convertir ese argumento al tipo distinto EURO:

```
SELECT * FROM US_SALES  
WHERE TOTAL = EURO_TO_US(EURO(:H1));
```

```
SELECT * FROM US_SALES  
WHERE TOTAL = EURO_TO_US(EURO(10000));
```

Casos en los que Db2 convierte argumentos para una función definida por el usuario

En determinadas situaciones, cuando invoca una función definida por el usuario, Db2 convierte los valores de argumento de entrada en diferentes tipos de datos y longitudes.

Cada vez que invoque una función definida por el usuario, Db2 asigna los valores de los argumentos de entrada a los parámetros con los tipos de datos y las longitudes de la definición de la función definida por el usuario.

Cuando invoca una función definida por el usuario que se origina en otra función, l Db2 o convierte sus argumentos a los tipos de datos y longitudes de la función de origen.

El siguiente ejemplo muestra lo que ocurre cuando las definiciones de los parámetros de una función fuente difieren de las de la función en la que se basa.

Supongamos que la función externa definida por el usuario TAXFN1 se define de la siguiente manera:

```
CREATE FUNCTION TAXFN1(DEC(6,0))  
RETURNS DEC(5,2)  
PARAMETER STYLE SQL  
LANGUAGE C  
EXTERNAL NAME TAXPROG;
```

La función definida por el usuario TAXFN2, que se obtiene en TAXFN1, se define de la siguiente manera:

```
CREATE FUNCTION TAXFN2(DEC(8,2))  
RETURNS DEC(5,0)  
SOURCE TAXFN1;
```

Usted invoca TAXFN2 utilizando esta instrucción SQL:

```
UPDATE TB1  
SET SALESTAX2 = TAXFN2(PRICE2);
```

TB1 se define de la siguiente manera:

```
CREATE TABLE TB1  
(PRICE1 DEC(6,0),  
SALESTAX1 DEC(5,2),  
PRICE2 DEC(9,2),  
SALESTAX2 DEC(7,2));
```

Ahora supongamos que PRICE2 tiene el valor DECIMAL(9,2) 0001234.56. Db2 debe asignar primero este valor al tipo de datos del parámetro de entrada en la definición de TAXFN2, que es DECIMAL(8,2). El valor del parámetro de entrada se convierte entonces en 001234.56. A continuación, Db2 asigna el valor del parámetro a un parámetro de función de origen, que es DECIMAL(6,0). El valor del parámetro se convierte entonces en 001234. (Cuando se introduce un valor, este se trunca, en lugar de redondearse)

Ahora, si TAXFN1 devuelve el valor DECIMAL(5,2) 123.45, Db2 convierte el valor a DECIMAL(5,0), que es el tipo de resultado para TAXFN2, y el valor se convierte en 00123. Este es el valor que Db2 asigna a la columna SALESTAX2 en la instrucción UPDATE.

Colocación de marcadores de parámetros

Puede utilizar marcadores de parámetros sin tipificar en una invocación de función. Sin embargo, Db2 no puede comparar los tipos de datos de los marcadores de parámetros sin tipificar con los tipos de datos de las funciones candidatas. Por lo tanto, Db2 podría encontrar más de una función que cumpla los

requisitos para su invocación. Si esto ocurre, se produce un error SQL. Para asegurarte de que Db2 elige la función correcta para ejecutar, asigna los marcadores de parámetros en tu invocación de función a los tipos de datos de los parámetros en la función que deseas ejecutar. Por ejemplo, supongamos que existen dos versiones de la función FX. Una versión de FX se define con un parámetro de tipo DECIMAL(9,2), y la otra se define con un parámetro de tipo INTEGER. Quiere invocar FX con un marcador de parámetro, y quiere que Db2 ejecute la versión de FX que tiene un parámetro DECIMAL(9,2). Debe convertir el marcador de parámetro a un tipo DECIMAL(9,2) mediante una especificación CAST:

```
SELECT FX(CAST(? AS DECIMAL(9,2))) FROM T1;
```

Conceptos relacionados

[Asignación y comparación \(Db2 SQL\)](#)

Capítulo 4. Programación de SQL incorporado

Los programas de aplicación escritos en lenguajes de host como COBOL pueden contener sentencias SQL. El formato fuente de una sentencia de SQL estático se incluye dentro del programa de aplicación y la sentencia se prepara antes de que se ejecute el programa y el formato operativo de la sentencia persiste más allá de la ejecución del programa.

Un programa de aplicación también puede contener sentencias SQL dinámicas denominadas SQL dinámicas incrustadas. Los programas que contienen sentencias de SQL dinámico incorporado deben precompilarse al igual que los que contienen SQL estático, pero a diferencia del SQL estático, las sentencias dinámicas se crean y preparan en tiempo de ejecución. El formato de origen de una sentencia dinámica es una serie de caracteres que el programa pasa a Db2 utilizando la sentencia SQL PREPARE o EXECUTE IMMEDIATE estática. Se puede hacer referencia a una sentencia que se ha preparado con la sentencia PREPARE en una sentencia DECLARE CURSOR, DESCRIBE o EXECUTE. Si el formato operativo de la sentencia es persistente o no depende de si se ha habilitado el almacenamiento en memoria caché de sentencias dinámicas.

Descripción general de las aplicaciones de programación que acceden a datos de Db2 for z/OS

Las aplicaciones que interactúan con Db2, en primer lugar se deben conectar a Db2. Entonces pueden leer, añadir o modificar datos o manipular objetos de Db2.

Acerca de esta tarea

Una consulta es una instrucción SQL que devuelve datos de una base de datos de e Db2 . Su programa puede utilizar varios métodos para comunicar instrucciones SQL a Db2 for z/OS. Después de procesar la declaración, Db2 emite un código de retorno, que su programa puede probar para determinar el resultado de la operación.

Conceptos introductorios

[Programación para Db2 for z/OS \(Introducción a Db2 for z/OS \)](#)

[Herramientas e IDE para desarrollar aplicaciones de Db2 \(Introducción a Db2 para z/OS\)](#)

[Proceso de preparación para un programa de aplicación \(Introducción a Db2 para z/OS\)](#)

[Información de rendimiento para la programación de la aplicación SQL \(Introducción a Db2 para z/OS\)](#)

Procedimiento

Para incluir consultas de tipo " Db2 for z/OS " en un programa de aplicación:

1. Elija uno de los siguientes métodos para comunicarse con Db2:

SQL estático

El formulario fuente de una instrucción SQL estática está incrustado en un programa de aplicación escrito en un lenguaje host. La sentencia se prepara antes de que se ejecute y el formato operativo de la sentencia se mantiene después de la ejecución del programa.

SQL dinámico incorporado

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

Open Database Connectivity (ODBC)

Accede a los datos a través de llamadas a funciones de ODBC en su aplicación. Ejecutas instrucciones SQL pasándolas a Db2 a través de una llamada a la función ODBC. ODBC elimina la necesidad de precompilar y vincular su aplicación y aumenta la portabilidad de su aplicación mediante el uso de la interfaz de programación de aplicaciones (ODBC).

JDBC asistencia para la aplicación

Si está escribiendo sus aplicaciones en Java, puede utilizar el soporte de aplicaciones de JDBC para acceder a Db2. JDBC es similar a ODBC, pero está diseñado específicamente para su uso con Java.

Soporte de aplicaciones SQLJ

También puede utilizar el soporte de aplicaciones SQLJ para acceder a Db2. SQLJ está diseñado para simplificar la codificación de llamadas a procedimientos almacenados (Db2) para aplicaciones Java.

Db2 for Linux, UNIX, and Windows controladores

Puede utilizar los controladores de cliente para conectarse a Db2 for z/OS desde lenguajes de programación de aplicaciones como Node.js, Perl, Python, Ruby on Rails, PHP y otros.

2. Opcional: Declare las tablas y vistas que utiliza.

Puede utilizar DCLGEN para generar estas declaraciones.

3. Defina los elementos que su programa puede utilizar para comprobar si una instrucción SQL se ha ejecutado correctamente. Puede definir un área de comunicaciones SQL (SQLCA) o declarar variables de host SQLSTATE y SQLCODE.

4. Definir al menos un área de descriptor SQL (SQLDA).

5. Declare cualquiera de los siguientes elementos de datos para pasar datos entre Db2 y un lenguaje de programación:

- “Variables de host” en la página 503
- “Matrices de variables de host” en la página 504
- “Estructuras host” en la página 506

Asegúrese de utilizar los tipos de datos adecuados. Para obtener más detalles, consulte “Compatibilidad de SQL y tipos de datos de lenguaje” en la página 511

6. Código de instrucciones SQL para acceder a datos de Db2 . Asegúrese de delimitar las sentencias correctamente para el lenguaje de programación específico.

Para obtener más información sobre la codificación de instrucciones SQL en lenguajes de host, consulte la información específica del lenguaje de programación:

- Assembler
- C y C++
- COBOL
- Fortran
- Java
- ODBC
- PL/I
- REXX

Consideré utilizar cursores para seleccionar un conjunto de filas y, a continuación, procesar el conjunto, ya sea una fila a la vez o un conjunto de filas a la vez.

7. Compruebe la ejecución de las instrucciones SQL.

8. Manejar cualquier código de error SQL.

Qué hacer a continuación

“Escritura de aplicaciones que permiten al usuario crear y modificar tablas” en la página 570
“Guardar sentencias SQL que se convierten en solicitudes de usuario” en la página 571

Conceptos relacionados

Programas de ejemplo que llaman a procedimientos almacenados

Los ejemplos se pueden utilizar como modelos cuando escribe aplicaciones que llaman a procedimientos almacenados. Asimismo, *prefijo.SDSNSAMP* contiene trabajos de ejemplo de DSNTEJ6P y DSNTEJ6S y programas DSN8EP1 y DSN8EP2 de ejemplo, que puede ejecutar.

[Datos XML en aplicaciones de SQL incorporado \(Db2 Programming for XML\)](#)

[Introducción a Db2 ODBC \(Db2 Programming for ODBC\)](#)

[Programación de aplicaciones JDBC \(Db2 Application Programming for Java\)](#)

[Programación de aplicaciones SQLJ \(Db2 Application Programming for Java\)](#)

Tareas relacionadas

[Inclusión de SQL dinámico en el programa](#)

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

[Programación de aplicaciones para mejorar el rendimiento \(Db2 Performance\)](#)

[Recuperación de un conjunto de filas utilizando un cursor](#)

En un programa de aplicación, puede recuperar un conjunto de filas de una tabla o una tabla de resultados que devuelve el procedimiento almacenado. Puede recuperar una o más filas a la vez.

[Escritura eficiente de consultas SQL \(Db2 Performance\)](#)

Declaración de definiciones de vista y tabla

Antes de que el programa emita sentencias SQL que seleccionan, insertan, actualicen o supriman datos, el programa debe declarar las tablas y vistas a las que acceden esas sentencias.

Acerca de esta tarea

No es obligatorio que su programa declare tablas o vistas, pero hacerlo ofrece las siguientes ventajas:

- Documentación clara en el programa

La declaración especifica la estructura de la tabla o vista y el tipo de datos de cada columna. Puede consultar la declaración para conocer los nombres de las columnas y los tipos de datos en la tabla o vista.

- Garantía de que su programa utiliza los nombres de columna y los tipos de datos correctos

El precompilador de Db2 utiliza sus declaraciones para asegurarse de que ha utilizado nombres de columna y tipos de datos correctos en sus sentencias SQL. El precompilador de vistas (Db2) emite un mensaje de advertencia cuando los nombres de las columnas y los tipos de datos en las sentencias SQL no se corresponden con las declaraciones de tablas y vistas en su programa.

Procedimiento

Para declarar definiciones de tablas y vistas, utilice uno de los siguientes métodos:

- Incluya una instrucción SQL DECLARE TABLE en su programa. Especifique el nombre de la tabla o vista y enumere cada columna y su tipo de datos.

Cuando declare una tabla o vista que contenga una columna con un tipo distinto, declare esa columna con el tipo de origen del tipo distinto en lugar de con el tipo distinto en sí. Cuando declaras la columna con el tipo de fuente, Db2 puede comprobar las sentencias SQL incrustadas que hacen referencia a esa columna en el momento de la precompilación.

En un programa COBOL, codifique la instrucción DECLARE TABLE en la WORKING-STORAGE SECTION o LINKAGE SECTION dentro de la DATA DIVISION.

Por ejemplo, la siguiente instrucción DECLARE TABLE en un programa COBOL define el tipo de datos (DSN8C10).DEPT Tabla:

```
EXEC SQL
  DECLARE DSN8C10.DEPT TABLE
    (DEPTNO   CHAR(3)          NOT NULL,
     DEPTNAME VARCHAR(36)      NOT NULL,
     MGRNO    CHAR(6)          );
```

```
ADMRDEPT CHAR(3)          NOT NULL,  
LOCATION  CHAR(16)          )  
END-EXEC.
```

- Utilice DCLGEN, el generador de declaraciones que se suministra con Db2, para crear estas declaraciones por usted y luego incluirlas en su programa.

Restricción: Puede utilizar DCLGEN solo para programas C, COBOL y PL/I.

Conceptos relacionados

[DCLGEN \(generador de declaraciones\)](#)

El programa debe declarar las tablas y vistas a las que accede. El generador de declaraciones de Db2, DCLGEN, crea estas sentencias DECLARE para programas C, COBOL y PL/I programs, y así no necesita codificar las sentencias él mismo. DCLGEN genera también las estructuras de variable de lenguaje principal correspondientes.

Referencia relacionada

[DECLARE TABLE declaración \(Db2 SQL\)](#)

[Subcomando DCLGEN \(generador de declaraciones\) \(DSN\) \(Db2 Commands\)](#)

DCLGEN (generador de declaraciones)

El programa debe declarar las tablas y vistas a las que accede. El generador de declaraciones de Db2, DCLGEN, crea estas sentencias DECLARE para programas C, COBOL y PL/I programs, y así no necesita codificar las sentencias él mismo. DCLGEN genera también las estructuras de variable de lenguaje principal correspondientes.

DCLGEN genera una declaración de tabla o vista y la coloca en un miembro de un conjunto de datos particionados que puede incluir en el programa. Cuando se utiliza DCLGEN para generar una declaración de tabla, Db2 obtiene la información relevante del catálogo Db2 . El catálogo contiene información sobre la definición de la tabla o vista y la definición de cada columna dentro de la tabla o vista. DCLGEN utiliza esta información para producir una instrucción SQL DECLARE TABLE para la tabla o vista y una declaración de estructura PL/I o C correspondiente o una descripción de registro COBOL.

Tareas relacionadas

[Generación de declaraciones de tablas y vistas con DCLGEN](#)

El programa debe declarar las tablas y vistas a las que accede. Para los programas C, COBOL y PL/I, el usuario puede utilizar DCLGEN para producir estas declaraciones, y así no tener que codificar las sentencias para sí mismo. DCLGEN genera también las estructuras de variable de lenguaje principal correspondientes.

Referencia relacionada

[Subcomando DCLGEN \(generador de declaraciones\) \(DSN\) \(Db2 Commands\)](#)

Generación de declaraciones de tablas y vistas con DCLGEN

El programa debe declarar las tablas y vistas a las que accede. Para los programas C, COBOL y PL/I, el usuario puede utilizar DCLGEN para producir estas declaraciones, y así no tener que codificar las sentencias para sí mismo. DCLGEN genera también las estructuras de variable de lenguaje principal correspondientes.

Antes de empezar

Requisitos:

- Db2 debe estar activo para poder utilizar DCLGEN.
- Puede utilizar DCLGEN para declaraciones de tablas solo si la tabla o vista que está declarando ya existe.
- Si utiliza DCLGEN, debe hacerlo antes de precompilar su programa.

Procedimiento

Para generar declaraciones de tabla y vista mediante DCLGEN:

1. Invoque DCLGEN realizando una de las siguientes acciones:

- **Para iniciar DCLGEN desde ISPF a través de DB2I:** Seleccione la opción DCLGEN en el panel del menú de opciones principales de DB2I. A continuación, siga las instrucciones detalladas para generar declaraciones de tabla y vista utilizando DCLGEN desde DB2I.
- **Para iniciar DCLGEN directamente desde TSO :** Inicie sesión en TSO, ejecute el comando DSN de TSO y, a continuación, ejecute el subcomando DCLGEN.
- **Para iniciar DCLGEN directamente desde un CLIST :** Desde un CLIST, ejecutándose en primer plano o en segundo plano en TSO, emita DSN y luego DCLGEN.
- **Para iniciar DCLGEN con JCL :** Proporcione la información necesaria en JCL y ejecute DCLGEN en lote. Utilice los trabajos de muestra DSNEJ2C y DSNEJ2P en *la biblioteca prefix.SDSNSAMP* como modelos.

Requisito: Si desea iniciar DCLGEN en primer plano y los nombres de sus tablas incluyen caracteres DBCS, debe proporcionar y mostrar caracteres de doble byte. Si no dispone de un terminal que muestre caracteres DBCS, puede introducirlos utilizando el modo hexadecimal de la edición de ISPF.

DCLGEN crea las declaraciones en el conjunto de datos especificado.

DCLGEN genera un nombre de tabla o columna en la sentencia DECLARE como identificador no delimitado, a menos que se cumpla al menos una de las siguientes condiciones:

- El nombre contiene caracteres especiales y no es una cadena DBCS.
 - El nombre es una cadena DBCS y ha solicitado nombres DBCS delimitados.
2. Si utiliza una palabra reservada de SQL como identificador, edite la salida DCLGEN para añadir los delimitadores de SQL adecuados.
 3. Realice cualquier otra modificación necesaria en la salida DCLGEN.

DCLGEN produce un resultado que está destinado a satisfacer las necesidades de la mayoría de los usuarios, pero en ocasiones es necesario editar el resultado de DCLGEN para que funcione en su caso específico. Por ejemplo, DCLGEN no puede determinar si una columna definida como NOT NULL también contiene la cláusula DEFAULT, por lo que debe editar el resultado de DCLGEN para añadir la cláusula DEFAULT a las definiciones de columna apropiadas.

DCLGEN produce declaraciones basadas en el esquema de codificación de la tabla fuente. Por lo tanto, si su aplicación utiliza un esquema de codificación diferente, es posible que tenga que ajustar manualmente las declaraciones. Por ejemplo, si su tabla fuente está en EBCDIC con columnas CHAR y su aplicación está en COBOL, DCLGEN produce declaraciones de tipo PIC X. Sin embargo, supongamos que las variables de host en su aplicación COBOL son UTF-16. En este caso, deberá cambiar manualmente las declaraciones para que sean del tipo PIC N USAGE NATIONAL.

Referencia relacionada

[Subcomando DCLGEN \(generador de declaraciones\) \(DSN\) \(Db2 Commands\)](#)

[DSN comando \(TSO\) \(Db2 Commands\)](#)

[Palabras reservadas en Db2 for z/OS \(Db2 SQL\)](#)

Generación de declaraciones de tabla y vista mediante DCLGEN de DB2I

DCLGEN genera declaraciones de tablas y vistas y las declaraciones de variables correspondientes para programas C, COBOL y PL/I, de modo que no es necesario que codifique estas instrucciones usted mismo. La forma más fácil de iniciar DCLGEN es a través de DB2I.

Procedimiento

Para generar declaraciones de tabla y vista mediante DCLGEN desde DB2I:

1. En el panel del menú de opciones principales de DB2I, seleccione la opción **DCLGEN**.

Se muestra el siguiente panel DCLGEN:

```
DSNEDP01          DCLGEN          SSID: DSN
====>

Enter table name for which declarations are required:
1 SOURCE TABLE NAME ===>

2 TABLE OWNER ..... ===>

3 AT LOCATION ..... ===>                               (Optional)
Enter destination data set:      (Can be sequential or partitioned)
4 DATA SET NAME ... ===>
5 DATA SET PASSWORD ===>                               (If password protected)
Enter options as desired:
6 ACTION ..... ===> ADD      (ADD new or REPLACE old declaration)
7 COLUMN LABEL .... ===> NO     (Enter YES for column label)
8 STRUCTURE NAME .. ===>                               (Optional)
9 FIELD NAME PREFIX ===>                               (Optional)
10 DELIMIT DBCS .... ===> YES    (Enter YES to delimit DBCS identifiers)
11 COLUMN SUFFIX ... ===> NO     (Enter YES to append column name)
12 INDICATOR VARS .. ===> NO     (Enter YES for indicator variables)
13 ADDITIONAL OPTIONS==> YES    (Enter YES to change additional options)

PRESS: ENTER to process    END to exit    HELP for more information
```

Figura 25. Panel DCLGEN

2. Complete los siguientes campos en el panel DCLGEN:

1 FUENTE NOMBRE DE LA TABLA

Es el nombre sin cualificar de la tabla, vista o tabla temporal creada para la que desea que DCLGEN produzca declaraciones de datos SQL. La mesa puede almacenarse en su ubicación de Db2 o en otra ubicación de Db2 . Para especificar un nombre de tabla en otra ubicación de Db2 , introduzca el calificador de tabla en el campo TABLE OWNER y el nombre de la ubicación en el campo AT LOCATION. DCLGEN genera un nombre de tabla de tres partes a partir de los campos NOMBRE DE TABLA DE ORIGEN, PROPIETARIO DE LA TABLA y EN UBICACIÓN. También puede utilizar un alias para el nombre de una tabla.

Para especificar un nombre de tabla que contenga caracteres especiales o espacios en blanco, escriba el nombre entre comillas. Si el nombre contiene apóstrofes, debe duplicar cada uno de ellos (' '). Por ejemplo, para especificar una tabla llamada DON'S TABLE, introduzca el siguiente texto:

```
'DON''S TABLE'
```

El guión bajo no se maneja como un carácter especial en DCLGEN. Por ejemplo, el nombre de la tabla JUNE_PROFITS no necesita ir entre comillas. Debido a que los nombres de campo COBOL no pueden contener guiones bajos, DCLGEN sustituye los guiones bajos de un solo byte en los nombres de campo COBOL que se construyen a partir del nombre de la tabla por guiones (-)

No es necesario que escriba los nombres de las tablas DBCS entre apóstrofes.

Si no incluye el nombre de la tabla entre comillas, Db2 convierte los caracteres en minúsculas a mayúsculas.

2 PROPIETARIO DE LA MESA

Es el calificador de esquema de la tabla de origen. Si no especifica este valor y la tabla es local, Db2 asume que el calificador de tabla es su ID de inicio de sesión de TSO. Si la mesa está en una ubicación remota, debe especificar este valor.

3 EN EL LUGAR

Es la ubicación de una mesa o vista en otro subsistema de Db2 . El valor del campo AT LOCATION se convierte en un prefijo para el nombre de la tabla en la instrucción SQL DECLARE, de la siguiente manera: *location_name, schema_name, table_name*. Por ejemplo, si el nombre de la ubicación es PLAINS_GA, el nombre del esquema es CARTER y el nombre de la tabla es CROP_YIELD_89, el siguiente nombre de tabla se incluye en la instrucción SQL DECLARE: PLAINS_GA.CARTER.CROP_YIELD_89

El valor predeterminado es el nombre de la ubicación local. Este campo se aplica únicamente al acceso al protocolo privado de Db2 . La ubicación debe ser otro subsistema de Db2 for z/OS .

4 DATA SET NAME

Es el nombre del conjunto de datos que asignó para contener las declaraciones que produce DCLGEN. Debe proporcionar un nombre; no existe ningún valor predeterminado.

El conjunto de datos debe existir ya y ser accesible para DCLGEN. El conjunto de datos puede ser secuencial o dividido. Si no incluye el nombre del conjunto de datos entre apóstrofes, DCLGEN añade un prefijo TSO estándar (ID de usuario) y un sufijo (idioma). DCLGEN determina el idioma del host desde el panel de valores predeterminados de DB2I.

Por ejemplo, para el nombre de biblioteca LIBNAME(MEMBNAME), el nombre se convierte en *id de usuario.libname.idioma* (*membname*). Para el nombre de biblioteca LIBNAME, el nombre se convierte en *id de usuario.libname.idioma*.

Si este conjunto de datos está protegido por contraseña, debe proporcionar la contraseña en el campo CONTRASEÑA DEL CONJUNTO DE DATOS.

5 CONFIGURACIÓN DE DATOS CONTRASEÑA

Es la contraseña del conjunto de datos que se especifica en el campo NOMBRE DEL CONJUNTO DE DATOS, si el conjunto de datos está protegido por contraseña. La contraseña no se muestra en su terminal y no se reconoce si la emitió en una sesión anterior.

6 ACCIÓN

Especifica lo que DCLGEN debe hacer con la salida cuando se envía a un conjunto de datos particionado. (La opción se ignora si el conjunto de datos que especifica en el campo NOMBRE DEL CONJUNTO DE DATOS es secuencial) Puede especificar uno de los valores siguientes:

ADD

Indica que no existe una versión antigua de la salida y crea un nuevo miembro con el nombre de conjunto de datos especificado. Add es el valor por omisión.

REPLACE

Reemplaza una versión antigua, si ya existe. Si el miembro no existe, esta opción crea un nuevo miembro.

eTIQUETA DE 7 COLUMNAS

Especifica si DCLGEN debe incluir etiquetas que se declaran en cualquier columna de la tabla o vista como comentarios en las declaraciones de datos. (La instrucción SQL LABEL crea etiquetas de columna para utilizarlas como suplementos de los nombres de columna) Puede especificar uno de los valores siguientes:

YES

Incluir etiquetas de columna.

NEE

Ignore las etiquetas de las columnas. NO es el valor predeterminado.

8 NOMBRE DE LA ESTRUCTURA

Es el nombre de la estructura de datos generada. El nombre puede tener hasta 31 caracteres. Si el nombre no es una cadena DBCS y el primer carácter no es alfabético, escriba el nombre entre comillas. Si utiliza caracteres especiales, tenga cuidado de evitar conflictos de nombres.

Si deja este campo en blanco, DCLGEN genera un nombre que contiene el nombre de la tabla o vista con un prefijo de DCL. Si el lenguaje es COBOL o PL/I y el nombre de la tabla o vista consiste en una cadena DBCS, el prefijo consiste en caracteres DBCS.

Para C, los caracteres en minúsculas que introduzca en este campo no se convertirán a mayúsculas.

9 PREFIJO DEL NOMBRE DEL CAMPO

Especifica un prefijo que DCLGEN utiliza para formar nombres de campo en la salida. Por ejemplo, si elige ABCDE, los nombres de campo generados son ABCDE1, ABCDE2, etc.

Puede especificar un prefijo de nombre de campo de hasta 28 bytes que puede incluir caracteres especiales y de doble byte. Si especifica un prefijo de un solo byte o una cadena mixta y el primer carácter no es alfabético, escriba el prefijo entre comillas. Si utiliza caracteres especiales, tenga cuidado de evitar conflictos de nombres.

Para COBOL y PL/I, si el nombre es una cadena DBCS, DCLGEN genera equivalentes DBCS de los números de sufijo.

Para C, los caracteres en minúsculas que introduzca en este campo no se convertirán en mayúsculas.

Si deja este campo en blanco, los nombres de los campos serán los mismos que los nombres de las columnas de la tabla o vista.

10 DELIMIT DBCS

Especifica si DCLGEN debe delimitar los nombres de las tablas DBCS y los nombres de las columnas en la declaración de la tabla. Puede especificar uno de los valores siguientes:

YES

Especifica que DCLGEN debe encerrar la tabla DBCS y los nombres de las columnas con delimitadores SQL.

NEE

Especifica que DCLGEN no debe delimitar la tabla DBCS ni los nombres de las columnas.

11 SUFIJO DE COLUMN

Especifica si DCLGEN debe formar nombres de campo adjuntando el nombre de la columna como sufijo al valor que especifique en PREFIJO DE NOMBRE DE CAMPO. Puede especificar uno de los valores siguientes:

YES

Especifica que DCLGEN debe utilizar el nombre de la columna como sufijo. Por ejemplo, si especifica SÍ, el prefijo del nombre de campo es NUEVO y el nombre de columna es EMPNO, el nombre de campo es NUEVOEMPNO.

Si especifica SÍ, también debe introducir un valor en PREFIJO DE NOMBRE DE CAMPO. Si no introduce un prefijo de nombre de campo, DCLGEN emite un mensaje de advertencia y utiliza los nombres de columna como nombres de campo.

NEE

Especifica que DCLGEN no debe utilizar el nombre de la columna como sufijo. El valor predeterminado es NO.

12 VARIABLES INDICADORAS

Especifica si DCLGEN debe generar una matriz de variables indicadoras para la estructura de variables del host. Puede especificar uno de los valores siguientes:

YES

Especifica que DCLGEN debe generar una matriz de variables indicadoras para la estructura de variables del host.

Si especifica SÍ, el nombre de la matriz es el nombre de la tabla con un prefijo de I (o letra DBCS <I> si el nombre de la tabla consta únicamente de caracteres de doble byte). La forma de la declaración de datos depende del idioma, como se muestra en la siguiente tabla. *n* es el número de columnas de la tabla.

Tabla 82. Declaraciones para matrices de variables indicadoras de DCLGEN

Idioma	Formulario de declaración
C	short int Itable-name[n];
COBOL	01 Itable-name PIC S9(4) USAGE COMP-5 OCCURS n TIMES.
PL/I	DCL Itable-name(n) BIN FIXED(15);

Por ejemplo, supongamos que define la siguiente tabla:

```
CREATE TABLE HASNULLS (CHARCOL1 CHAR(1), CHARCOL2 CHAR(1));
```

Si solicita una matriz de variables indicadoras para un programa COBOL, DCLGEN podría generar la siguiente declaración de variables de host:

```
01 DCLHASNULLS.  
  10 CHARCOL1          PIC X(1).  
  10 CHARCOL2          PIC X(1).  
01 IHASNULLS PIC S9(4) USAGE COMP-5 OCCURS 2 TIMES.
```

NEE

Especifica que DCLGEN no debe generar una matriz de variables indicadoras. El valor predeterminado es NO.

13 OPCIONES ADICIONALES

Indica si se debe mostrar el panel para opciones DCLGEN adicionales, incluido el punto de interrupción para tokens de sentencia y si se deben generar sentencias DECLARE VARIABLE para columnas FOR BIT DATA. Puede especificar SÍ o NO. El valor por omisión es YES.

Si especificó SÍ en el campo OPCIONES ADICIONALES, se muestra el siguiente panel OPCIONES ADICIONALES DE DCLGEN:

```
DSNEDP02           ADDITIONAL DCLGEN OPTIONS           SSID: DSN  
====>  
Enter options as desired:  
1 RIGHT MARGIN .... ===> 72      (Enter 72 or 80)  
2 FOR BIT DATA .... ===> NO       (Enter YES to declare SQL variables for  
FOR BIT DATA columns)  
  
PRESS: ENTER to process    END to exit    HELP for more information
```

Figura 26. Panel OPCIONES ADICIONALES DE DCLGEN

De lo contrario, DCLGEN crea las declaraciones en el conjunto de datos especificado.

3. Si se muestra el panel OPCIONES ADICIONALES DE DCLGEN, complete los siguientes campos en ese panel:

1 MARGEN DERECHO

Especifica el punto de ruptura para los tokens de sentencia que deben envolverse en uno o más registros posteriores. Puede especificar la columna 72 o la columna 80.

El valor predeterminado es 72.

2 PARA DATOS BIT

Especifica si DCLGEN debe generar una instrucción DECLARE VARIABLE para variables SQL para columnas que se declaran como FOR BIT DATA. Esta declaración es obligatoria en las solicitudes de Db2 que cumplan todos los criterios siguientes:

- están escritos en COBOL
- tener variables de host para columnas FOR BIT DATA
- se preparan con la opción SQLCCSID del coprocesador de la biblioteca de enlace dinámico (Db2).

Puede especificar SÍ o NO. El valor predeterminado es NO.

Si la tabla o vista no tiene columnas FOR BIT DATA, DCLGEN no genera esta sentencia.

DCLGEN crea las declaraciones en el conjunto de datos especificado.

Referencia relacionada

El menú de la opción primaria de DB2I (Introducción a Db2 for z/OS)

[LABEL declaración \(Db2 SQL\)](#)

Tipos de datos que DCLGEN utiliza para declaraciones de variables

DCLGEN produce declaraciones para tablas y vistas y las estructuras de variables de host correspondientes para programas C, COBOL y PL/I. DCLGEN deriva los nombres de variables y tipos de datos para estas declaraciones basándose en las tablas de origen de la base de datos.

La siguiente tabla enumera los tipos de datos C, COBOL y PL/I que DCLGEN utiliza para las declaraciones de variables basadas en los tipos de datos SQL correspondientes que se utilizan en las tablas de origen. *var* representa un nombre de variable que proporciona DCLGEN.

Tabla 83. Declaraciones de tipo que genera DCLGEN

Tipo de datos SQL ¹	C	COBOL	PL/I
SMALLINT	short int	PIC S9(4) USAGE COMP-5	BIN FIXED(15)
ENTERO	long int	PIC S9(9) USAGE COMP-5	BIN FIXED(31)
BIGINT	long long int	PIC S9(18) USAGE COMP-5	FIXED BIN(63)
DECIMAL (p, s) o NUMERIC (p, s)	decimal(p,s) ²	PIC S9(p-s)V9(s) USAGE COMP-3	DEC FIXED(p,s) Si p>15, el compilador PL/I debe admitir esta precisión o se generará una advertencia.
REAL o FLOAT(n) 1 <= n <= 21	float	USAGE COMP-1	BIN FLOAT(n)
DOBLE PRECISIÓN, DOBLE o FLOAT(n)	double	USAGE COMP-2	BIN FLOAT(n)
DECFLOAT(16)	_Decimal64	n/a	DEC FLOAT(16)
DECFLOAT(32)	_Decimal128	n/a	DEC FLOAT(16)
CHAR(1)	char	PIC X(1)	CHAR(1)
CHAR(n)	char <i>var</i> [n+1]	PIC X(n)	CHAR(n)
VARCHAR(n)	struct {short int <i>var_len</i> ; char <i>var_data</i> [n]; } <i>var</i> ;	10 <i>var</i> . 49 <i>var_LEN</i> PIC 9(4) USAGE COMP-5. 49 <i>var_TEXT</i> PIC X(n).	CHAR(n) VAR
CLOB(n) ³	SQL TYPE IS CLOB_LOCATOR	USAGE SQL TYPE IS CLOB-LOCATOR	SQL TYPE IS CLOB_LOCATOR
GRAPHIC (1)	sqldbchar	PIC G(1)	GRAPHIC(1)
GRAPHIC(n) n > 1	sqldbchar <i>var</i> [n+1];	PIC G(n) USAGE DISPLAY-1. ⁴ or PIC N(n). ⁴	GRAPHIC(n)

Tabla 83. Declaraciones de tipo que genera DCLGEN (continuación)

Tipo de datos SQL ¹	C	COBOL	PL/I
VARGRAPHIC(n)	<pre>struct VARGRAPH {short len; sqldbcchar data[n]; } var;</pre>	<pre>10 var. 49 var_LEN PIC 9(4) USAGE COMP-5. 49 var_TEXT PIC G(n) USAGE DISPLAY-1.⁴ or 10 var. 49 var_LEN PIC 9(4) USAGE COMP-5. 49 var_TEXT PIC N(n).⁴</pre>	GRAPHIC(n) VAR
DBCLOB(n)3	SQL TYPE IS DBCLOB_LOCATOR	USAGE SQL TYPE IS DBCLOB-LOCATOR	SQL TYPE IS DBCLOB_LOCATOR
BINARIO(n)	SQL TYPE IS BINARY(n)	USAGE SQL TYPE IS BINARY(n)	SQL TYPE IS BINARY(n)
VARBINARIO(n)	SQL TYPE IS VARBINARY(n)	USAGE SQL TYPE IS VARBINARY(n)	SQL TYPE IS VARBINARY(n)
BLOB(n)3	SQL TYPE IS BLOB_LOCATOR	USAGE SQL TYPE IS BLOB-LOCATOR	SQL TYPE IS BLOB_LOCATOR
FECHA	char var[11] ⁵	PIC X(10) ⁵	CHAR(10) ⁵
HORA	char var[9] ⁶	PIC X(8) ⁶	CHAR(8) ⁶
TIMESTAMP	char var[27]	PIC X(26)	CHAR(26)
TIMESTAMP(0)	char var[20]	PIC X(19)	CHAR(19)
MARCA DE TIEMPO(p) p > 0	char var[21+p]	PIC X(20+p)	CHAR(20+p)
TIMESTAMP (0) WITH TIME ZONE	<pre>struct {short int var_len; char var_data[147]; } var;</pre>	<pre>01 var. 49 var_LEN PIC S9(4) COMP-5. 49 var_TEXT PIC X(147).</pre>	DCL var CHAR(147) VAR;
FECHA Y HORA (p) CON ZONA HORARIA	<pre>struct {short int var_len; char var_data[148 + p]; } var;</pre>	<pre>01 var. 49 var_LEN PIC S9(4) COMP-5. 49 var_TEXT PIC X(148 + p).</pre>	DCL var CHAR(148 + p) VAR;
ROWID	SQL TYPE IS ROWID	USAGE SQL TYPE IS ROWID	SQL TYPE IS ROWID
XML7	SQL TYPE IS XML AS CLOB(1M)	SQL TYPE IS XML AS CLOB(1M)	SQL TYPE IS XML AS CLOB(1M)

Tabla 83. Declaraciones de tipo que genera DCLGEN (continuación)

Tipo de datos SQL ¹	C	COBOL	PL/I
Notas:			
1.	Para un tipo distinto, DCLGEN genera el equivalente en el lenguaje del host del tipo de datos de origen.		
2.	Si su compilador C no admite el tipo de datos decimal, edite su salida DCLGEN y reemplace las declaraciones de datos decimales por declaraciones de tipo doble.		
3.	Para un tipo de datos BLOB, CLOB o DBCLOB, DCLGEN genera un localizador LOB.		
4.	DCLGEN elige el formato basándose en el carácter que especifique como símbolo DBCS en el panel de valores predeterminados de COBOL.		
5.	Esta declaración se utiliza a menos que exista una rutina de salida de instalación de fecha para formatear fechas, en cuyo caso la longitud es la especificada para la opción de instalación LOCAL DATE LENGTH.		
6.	Esta declaración se utiliza a menos que exista una rutina de salida de instalación de hora para formatear horas, en cuyo caso la longitud es la especificada para la opción de instalación LOCAL TIME LENGTH.		
7.	La configuración predeterminada para XML es 1M; sin embargo, es posible que necesite ajustarla.		

Incluir declaraciones de DCLGEN en su programa

Después de utilizar DCLGEN para producir declaraciones para tablas, vistas y variables para su programa C, COBOL o PL/I, debe incluir estas declaraciones en su programa.

Antes de empezar

Recomendación: Para asegurarse de que su programa utiliza una descripción actual de la tabla, utilice DCLGEN para generar la declaración de la tabla y almacenarla como miembro en una biblioteca (normalmente un conjunto de datos particionado) justo antes de precompilar el programa.

Procedimiento

Codifique la siguiente instrucción SQL INCLUDE en su programa:

```
EXEC SQL
  INCLUDE member-name
END-EXEC.
```

nombre-miembro es el nombre del miembro del conjunto de datos donde se almacena la salida DCLGEN.

Ejemplo

Supongamos que utilizó DCLGEN para generar una declaración de tabla y la correspondiente descripción de registro COBOL para la tabla DSN8C10.EMP, y esas declaraciones se almacenaron en el miembro del conjunto de datos DECEMP. (Una descripción de registro COBOL es una estructura de host de dos niveles que se corresponde con las columnas de la fila de una tabla) Para incluir esas declaraciones en su programa, incluya la siguiente declaración en su programa COBOL:

```
EXEC SQL
  INCLUDE DECEMP
END-EXEC.
```

Referencia relacionada

[INCLUDE declaración \(Db2 SQL\)](#)

Ejemplo: Añadir declaraciones de DCLGEN a una biblioteca

Puede utilizar DCLGEN para generar declaraciones de tabla y variable para programas C, COBOL y PL/I. Si almacena estas declaraciones en una biblioteca, puede integrarlas posteriormente en el programa con una sola sentencia SQL INCLUDE.

Este ejemplo añade una declaración de tabla y una estructura de variable de host correspondiente a una biblioteca. Este ejemplo se basa en el siguiente escenario:

- El nombre de la biblioteca es *prefijo*. TEMP.COBOL.
- El miembro es un nuevo miembro llamado VPHONE.
- La tabla es una tabla local llamada DSN8C10.VPHONE.
- La estructura de la variable host es para COBOL.
- La estructura recibe el nombre predeterminado DCLVPHONE.

A lo largo de este ejemplo, la información que debe introducir en cada panel aparece en negrita.

En este escenario, para añadir una declaración de tabla y una estructura de variable de host correspondiente para DSN8C10.VPHONE al *prefijo* de la biblioteca. TEMP.COBOL, siga estos pasos:

1. Especifique COBOL como lenguaje de host completando las siguientes acciones:

- a. En el menú ISPF /PDF, seleccione la opción **D** para mostrar el panel DEFAULTS PANEL 1 (DB2I).
- b. Especifique IBMCOB como idioma de la aplicación, como se muestra en la siguiente figura, y pulse Intro.

```
DSNEOP01          DB2I DEFAULTS PANEL 1
COMMAND ==>_

Change defaults as desired:

1 DB2 NAME ..... ==> DSN      (Subsystem identifier)
2 DB2 CONNECTION RETRIES ==> 0    (How many retries for DB2 connection)
3 APPLICATION LANGUAGE ==> IBMCOB (ASM, C, CPP, IBMCOB, FORTRAN, PLI)
4 LINES/PAGE OF LISTING ==> 80   (A number from 5 to 999)
5 MESSAGE LEVEL ..... ==> I     (Information, Warning, Error, Severe)
6 SQL STRING DELIMITER ==> DEFAULT (DEFAULT, ' or ")
7 DECIMAL POINT ..... ==> .     (. or ,)
8 STOP IF RETURN CODE >= ==> 8    (Lowest terminating return code)
9 NUMBER OF ROWS ..... ==> 20   (For ISPF Tables)
10 CHANGE HELP BOOK NAMES?==> NO  (YES to change HELP data set names)
11 AS USER ..... ==>             (Userid to associate with the trusted
connection)

PRESS: ENTER to process      END to cancel      HELP for more information
```

Figura 27. DB2I panel de configuración predeterminada: cambiar el idioma de la aplicación

DB2I. A continuación, se muestra el panel 2 para COBOL.

- c. Complete el panel 2 de valores predeterminados (DB2I) que se muestra en la siguiente figura, según sea necesario, y pulse Intro para guardar los nuevos valores predeterminados, si los hay.

```
DSNEOP02          DB2I DEFAULTS PANEL 2
COMMAND ==>_

Change defaults as desired:

1 DB2I JOB STATEMENT: (Optional if your site has a SUBMIT exit)
  ==> //ADMF001A JOB (ACCOUNT),'NAME'
  ==> ///*
  ==> ///*
  ==> ///*

COBOL DEFAULTS:           (For IBMCOB)
2 COBOL STRING DELIMITER ==> DEFAULT (DEFAULT, ' or ")
3 DBCS SYMBOL FOR DCLGEN ==> G    (G/N - Character in PIC clause)
```

Figura 28. El panel predeterminado de COBOL

Se muestra el menú de Opción principal de DB2I.

2. Genere las declaraciones de la tabla y de la estructura del host completando las siguientes acciones:

- En el menú Opción principal de DB2I, seleccione la opción **DCLGEN** y pulse Intro para mostrar el panel DCLGEN.
- Complete los campos como se muestra en la siguiente figura y pulse Intro.

```

DSNEDP01          DCLGEN          SSID: DSN
====>

Enter table name for which declarations are required:
1 SOURCE TABLE NAME ===>
DSN8C10.VPHONE

2 TABLE OWNER ..... ===>

3 AT LOCATION ..... ===> (Optional)
Enter destination data set: (Can be sequential or partitioned)
4 DATA SET NAME ... ===>

TEMP(VPHONEC)
5 DATA SET PASSWORD ===> (If password protected)
Enter options as desired:
6 ACTION ..... ===> ADD (ADD new or REPLACE old declaration)
7 COLUMN LABEL ..... ===> NO (Enter YES for column label)
8 STRUCTURE NAME .. ===> (Optional)
9 FIELD NAME PREFIX ===> (Optional)
10 DELIMIT DBCS .... ===> YES (Enter YES to delimit DBCS identifiers)
11 COLUMN SUFFIX ... ===> NO (Enter YES to append column name)
12 INDICATOR VARS .. ===> NO (Enter YES for indicator variables)
13 ADDITIONAL OPTIONS==> NO (Enter YES to change additional options)

PRESS: ENTER to process END to exit HELP for more information

```

Figura 29. Panel DCLGEN: selección de la tabla de origen y el conjunto de datos de destino

Un mensaje de finalización correcta, como el de la siguiente figura, se muestra en la parte superior de la pantalla.

```

DSNE905I EXECUTION COMPLETE, MEMBER VPHONEC ADDED
***
```

Figura 30. Mensaje de finalización correcta

Db2 vuelve a mostrar la pantalla DCLGEN, como se muestra en la siguiente figura.

```

DSNEDP01          DCLGEN          SSID: DSN
====>

Enter table name for which declarations are required:
1 SOURCE TABLE NAME ===>
DSN8C10.VPHONE

2 TABLE OWNER ..... ===>

3 AT LOCATION ..... ===> (Optional)
Enter destination data set: (Can be sequential or partitioned)
4 DATA SET NAME ... ===> TEMP(VPHONEC)
5 DATA SET PASSWORD ===> (If password protected)
Enter options as desired:
6 ACTION ..... ===> ADD (ADD new or REPLACE old declaration)
7 COLUMN LABEL ..... ===> NO (Enter YES for column label)
8 STRUCTURE NAME .. ===> (Optional)
9 FIELD NAME PREFIX ===> (Optional)
10 DELIMIT DBCS .... ===> YES (Enter YES to delimit DBCS identifiers)
11 COLUMN SUFFIX ... ===> NO (Enter YES to append column name)
12 INDICATOR VARS .. ===> NO (Enter YES for indicator variables)
13 ADDITIONAL OPTIONS==> NO (Enter YES to change additional options)

PRESS: ENTER to process END to exit HELP for more information

```

Figura 31. Panel DCLGEN: muestra el sistema y los códigos de retorno del usuario

- Pulse Intro para volver al menú de Opción principal de DB2I.

- Salir de DB2I.

4. Examine el resultado de DCLGEN seleccionando la opción de examinar o editar en el menú ISPF /PDF para ver los resultados en el miembro del conjunto de datos especificado.

Para este ejemplo, el conjunto de datos que se va a editar es *prefix.TEMP.COBOL* (VPHONEC). Este miembro del conjunto de datos contiene la siguiente información.

```
***** DCLGEN TABLE(DSN8C10.VPHONE) *****
***** LIBRARY(SYADM TEMP COBOL(VPHONEC)) *****
***** QUOTE *****
***** ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS ***
EXEC SQL DECLARE DSN8C10.VPHONE TABLE
( LASTNAME           VARCHAR(15) NOT NULL,
  FIRSTNAME          VARCHAR(12) NOT NULL,
  MIDDLEINITIAL      CHAR(1) NOT NULL,
  PHONENUMBER        VARCHAR(4) NOT NULL,
  EMPLOYEENUMBER     CHAR(6) NOT NULL,
  DEPTNUMBER         CHAR(3) NOT NULL,
  DEPTNAME           VARCHAR(36) NOT NULL
) END-EXEC.
***** COBOL DECLARATION FOR TABLE DSN8C10.VPHONE *****
01 DCLVPHONE.
 10 LASTNAME.
    49 LASTNAME-LEN      PIC S9(4) USAGE COMP.
    49 LASTNAME-TEXT    PIC X(15).
 10 FIRSTNAME.
    49 FIRSTNAME-LEN    PIC S9(4) USAGE COMP.
    49 FIRSTNAME-TEXT   PIC X(12).
 10 MIDDLEINITIAL     PIC X(1).
 10 PHONENUMBER.
    49 PHONENUMBER-LEN  PIC S9(4) USAGE COMP.
    49 PHONENUMBER-TEXT PIC X(4).
 10 EMPLOYEENUMBER    PIC X(6).
 10 DEPTNUMBER        PIC X(3).
 10 DEPTNAME.
    49 DEPTNAME-LEN     PIC S9(4) USAGE COMP.
    49 DEPTNAME-TEXT    PIC X(36).
***** THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 7 *****
```

Ahora puede incorporar estas declaraciones a su programa mediante una instrucción SQL INCLUDE.

Definición de elementos que su programa puede utilizar para comprobar si una sentencia SQL se ha ejecutado correctamente

Si su programa contiene sentencias SQL, el programa debe definir determinada infraestructura para poder comprobar si las sentencias se han ejecutado correctamente. También puede incluir un área de comunicación SQL (SQLCA), que contenga variables SQLCODE y SQLSTATE, o declarar variables host SQLCODE y SQLSTATE.

Acerca de esta tarea

Que defina las variables SQLCODE o SQLSTATE o un SQLCA en su programa depende de lo que especifique para la opción de procesamiento SQL STDSQL.

Si su aplicación contiene instrucciones SQL y no incluye un área de comunicaciones SQL (SQLCA), debe declarar variables de host SQLCODE y SQLSTATE individuales. Su programa puede utilizar estas variables para comprobar si una instrucción SQL se ha ejecutado correctamente.

Tareas relacionadas

[Definición del área de comunicación SQL, SQLSTATE y SQLCODE en el ensamblador](#)

Los programas de ensamblador que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

[Definición del área de comunicaciones SQL, SQLSTATE y SQLCODE en C y C++](#)

Los programas en C y C++ que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en COBOL

Los programas en COBOL que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicación SQL, SQLSTATE y SQLCODE en Fortran

Los programas en Fortran que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en PL/I

Los programas en PL/I que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en REXX

Cuando Db2 prepara un programa REXX que contiene sentencias SQL, Db2 incluye automáticamente un SQLCA en el programa.

Referencia relacionada

Descripciones de opciones de proceso de SQL

Puede especificar cualquier opción de proceso de SQL tanto si utiliza el precompilador de Db2 como si utiliza el coprocesador de Db2. Sin embargo, es probable que el coprocesador de Db2 omita ciertas opciones ya que existen opciones del compilador del lenguaje principal que proporcionan la misma información.

Descripción de campos de SQLCA (Db2 SQL)

INCLUDE declaración (Db2 SQL)

El SQLCA de REXX (Db2 SQL)

Definición de áreas de descriptor de SQL (SQLDA)

Si su programa incluye ciertas sentencias SQL, debe definir al menos *un área de descriptor SQL (SQLDA)*. Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Acerca de esta tarea

Si su programa incluye alguna de las siguientes variaciones de sentencias, debe incluir un SQLDA en su programa:

- LLAMAR... USO DEL DESCRIPTOR *nombre-descriptor*
- DESCRIBE *nombre-declaración* INTO *nombre-descriptor*
- DESCRIBE CURSOR *variable-host* INTO *nombre-descriptor*
- DESCRIBE ENTRY *nombre-declaración* INTO *nombre-descriptor*
- DESCRIBE PROCEDURE *host-variable* INTO *descriptor-name*
- DESCRIBE TABLE *host-variable* INTO *descriptor-name*
- EXECUTAR. USO DEL DESCRIPTOR *nombre-descriptor*
- OBTENER. EN DESCRIPTOR *nombre-descriptor*
- ABIERTO. USO DEL DESCRIPTOR *nombre-descriptor*
- PREPÁRESE. INTO *nombre-descriptor*

A diferencia del SQLCA, un programa puede tener más de un SQLDA, y un SQLDA puede tener cualquier nombre válido.

Procedimiento

Tome las medidas adecuadas para el lenguaje de programación que utilice:

- “Definición de áreas de descriptor de SQL (SQLDA) en ensamblador” en la página 586
- “Definición de áreas de descriptor de SQL (SQLDA) en C y C++” en la página 615
- “Definición de áreas de descriptor de SQL (SQLDA) en COBOL” en la página 685
- “Definición de áreas de descriptor de SQL en (SQLDA) Fortran” en la página 725
- “Definición de áreas de descriptor de SQL (SQLDA) en PL/I” en la página 744
- “Definición de áreas de descriptor de SQL (SQLDA) en REXX” en la página 785

Referencia relacionada

[Área de descriptor de SQL \(SQLDA\) \(Db2 SQL\)](#)

[Descripciones de opciones de proceso de SQL](#)

Puede especificar cualquier opción de proceso de SQL tanto si utiliza el precompilador de Db2 como si utiliza el coprocesador de Db2. Sin embargo, es probable que el coprocesador de Db2 omita ciertas opciones ya que existen opciones del compilador del lenguaje principal que proporcionan la misma información.

[Descripción de campos de SQLCA \(Db2 SQL\)](#)

[El SQLCA de REXX \(Db2 SQL\)](#)

Declaración de variables host y variables de indicación

Puede utilizar variables host y variables de indicación en sentencias SQL del programa para pasar datos entre Db2 y la aplicación.

Procedimiento

Utilice las técnicas que sean apropiadas para el lenguaje de programación que utilice.

Tareas relacionadas

[Acceso a datos utilizando un cursor colocado por conjunto de filas](#)

Un cursor colocado por conjunto de filas es un cursor que puede devolver una o varias filas para una única operación de captación. El cursor se coloca en el conjunto de filas que se van a captar.

[Determinación de si un valor recuperado en una variable host es nulo o está truncado](#)

Antes de que su aplicación manipule los datos recuperados de Db2 en una variable host, determine si el valor es nulo. Determine también si se han truncado al asignarse a la variable. Puede utilizar las variables indicadores para obtener esta información.

Referencia relacionada

[Descripciones de opciones de proceso de SQL](#)

Puede especificar cualquier opción de proceso de SQL tanto si utiliza el precompilador de Db2 como si utiliza el coprocesador de Db2. Sin embargo, es probable que el coprocesador de Db2 omita ciertas opciones ya que existen opciones del compilador del lenguaje principal que proporcionan la misma información.

Variables de host

Utilice variables host para pasar un único elemento de datos entre Db2 y la aplicación.

Una variable host es un elemento de datos único que se declara en el lenguaje host para ser utilizado dentro de una instrucción SQL. Puede utilizar variables de host en programas de aplicación escritos en los siguientes lenguajes: ensamblador, C, C++, COBOL, Fortran, y PL/I para realizar las siguientes acciones:

- Recuperar datos en la variable de host para el uso de su programa de aplicación
- Colocar datos en la variable de host para insertarlos en una tabla o para cambiar el contenido de una fila
- Utilizar los datos de la variable de host al evaluar una cláusula WHERE o HAVING

- Asignar el valor que está en la variable host a un registro especial, como CURRENT SQLID y CURRENT DEGREE
- Insertar valores nulos en columnas utilizando una variable de indicador de host que contenga un valor negativo
- Utilizar los datos de la variable host en sentencias que procesan SQL dinámico, como EXECUTE, PREPARE y OPEN

Si utiliza el lenguaje de programación C (Db2 precompilador), asegúrese de que los nombres de las variables de host y de las matrices de variables de host sean únicos dentro del programa, incluso si las variables y las matrices de variables se encuentran en bloques, clases, procedimientos, funciones o subrutinas diferentes. Puede calificar los nombres con un nombre de estructura para hacerlos únicos.

Conceptos relacionados

Uso de variables host en sentencias SQL

Utilice variables de host escalares en sentencias de SQL incorporado para representar un único valor. Las variables host son útiles para almacenar datos recuperados o para pasar valores que van a asignar o utilizar las comparaciones.

Referencia relacionada

Variables de host en Assembler

En programas ensamblador, puede especificar variables host numéricas, de caracteres, gráficas, binarias, LOB, XML y ROWID. También puede especificar conjuntos de resultados, tabla y localizadores LOB y variables de referencia de archivos LOB y XML.

Variables de host en C y C++

En programas C y C++, puede especificar variables host numéricas, de caracteres, gráficas, binarias, LOB, XML y ROWID. También puede especificar conjuntos de resultados, tabla y localizadores LOB y variables de referencia de archivos LOB y XML.

Variables de host en COBOL

En programas COBOL, puede especificar variables host numéricas, de caracteres, gráficas, binarias, LOB, XML y ROWID. También puede especificar conjuntos de resultados y localizadores de tabla y LOB, y variables de referencia de archivos LOB y XML.

Variables host en Fortran

En programas Fortran, puede especificar variables host numéricas, de caracteres, LOB y ROWID. También puede especificar conjuntos de resultados y localizadores LOB.

Variables host en PL/I

En programas PL/I, puede especificar variables host numéricas, de caracteres, gráficas, binarias, LOB, XML y ROWID. También puede especificar conjuntos de resultados, tabla y localizadores LOB y variables de referencia de archivos LOB y XML.

Matrices de variables de host

Puede utilizar matrices de variables de host para pasar una matriz de datos entre Db2 y su aplicación. Una matriz de variables de host es una matriz de datos que se declara en el lenguaje de host para ser utilizada dentro de una instrucción SQL.

Puede utilizar matrices de variables de host para las siguientes acciones:

- Recuperar datos en matrices de variables de host para que su aplicación los utilice
- Colocar datos en matrices de variables de host para insertar filas en una tabla
- Recuperar datos para el origen de una operación de fusión.

Solo se puede hacer referencia a las matrices de variables de lenguaje principal como una referencia simple en los contextos siguientes. En los diagramas de sintaxis, *host-variable-array* designa una referencia a una matriz de variables de host.

- En una sentencia FETCH para una captación de varias filas. Consulte [FETCH declaración \(Db2 SQL\)](#).
- En la forma *FOR n ROWS* de la sentencia INSERT con una matriz de variables de host para los datos de origen. Consulte [INSERT declaración \(Db2 SQL\)](#).

- En una sentencia MERGE con varias filas de datos de origen. Consulte [MERGE declaración \(Db2 SQL\)](#).
- En una sentencia EXECUTE para proporcionar un valor para un marcador de parámetro en una forma dinámica *FOR n ROWS* de la sentencia INSERT o una sentencia MERGE. Consulte [EXECUTE declaración \(Db2 SQL\)](#).

Las matrices de variables de lenguaje principal se definen mediante sentencias del lenguaje principal, como se explica en los temas siguientes:

- [“Matrices de variables de host en C y C++” en la página 628](#)
- [“Matrices de variables de host en COBOL” en la página 696](#)
- [“Matrices de variables de host en PL/I” en la página 751](#)

Consejo: Las matrices de variables de lenguaje principal no reciben soporte para los programas assembler, FORTRAN o REXX. Sin embargo, puede utilizar áreas de descriptor de SQL (SQLDA) para obtener resultados similares en cualquier lenguaje de host. Para obtener más información, consulte [“Definición de áreas de descriptor de SQL \(SQLDA\)” en la página 502.](#)

Ejemplo



La sentencia siguiente utiliza la matriz de variables de lenguaje principal, COL1, y la matriz de indicadores correspondiente, COL1IND. Suponga que COL1 tiene 10 elementos. El primer elemento de la matriz corresponde al primer valor, etc. COL1IND debe tener como mínimo 10 entradas.

```
EXEC SQL
  SQL FETCH FIRST ROWSET FROM C1 FOR 5 ROWS
    INTO :COL1 :COL1IND
END-EXEC.
```



Conceptos relacionados

[Matrices de variables de lenguaje principal en PL/I, C, C++ y COBOL \(Db2 SQL\)](#)

[Utilización de matrices de variables de host en sentencias SQL](#)

Utilice matrices de variables de lenguaje de host en sentencias de SQL incorporado para representar valores que el programa no conoce hasta que se ejecuta la consulta. Las matrices de variables de host son útiles para almacenar un conjunto de valores recuperados o para pasar un conjunto de valores que se van a insertar en una tabla.

Tareas relacionadas

[Inserción de varias filas de datos desde matrices de variables de host](#)

Utilice las variables de host en la sentencia INSERT cuando no conozca al menos algunos de los valores que se han de insertar hasta que el programa se ejecuta.

[Recuperación de varias filas de datos en matrices de variables de host](#)

Si sabe que la consulta devuelve varias filas, puede especificar matrices de variables de host para almacenar los valores de columna recuperados.

Referencia relacionada

[Matrices de variables de host en C y C++](#)

En los programas C y C++, puede especificar matrices de variables de host de tipo numérico, carácter, gráfico, binario, LOB, XML y ROWID. También puede especificar localizadores LOB y variables de referencia de archivos LOB y XML.

[Matrices de variables de host en COBOL](#)

En los programas COBOL, puede especificar matrices de variables de host de tipo numérico, carácter, gráfico, binario, LOB, XML y ROWID. También puede especificar localizadores LOB y variables de referencia de archivos LOB y XML.

[Matrices de variables de host en PL/I](#)

En los programas PL/I, puede especificar matrices de variables de host de tipo numérico, carácter, gráfico, binario, LOB, XML y ROWID. También puede especificar localizadores LOB y variables de referencia de archivos LOB y XML.

Estructuras host

Utilice las estructuras host para pasar un grupo de variables host entre Db2 y su aplicación.

Una estructura de host es un grupo de variables de host a las que se puede hacer referencia con un solo nombre. Puede utilizar estructuras de host en todos los lenguajes de host excepto REXX. Usted define las estructuras de host con sentencias en el lenguaje de host. Puede hacer referencia a una estructura de host en cualquier contexto en el que desee hacer referencia a la lista de variables de host de la estructura. Una referencia de estructura de host equivale a una referencia a cada una de las variables de host dentro de la estructura en el orden en que se definen en la declaración de estructura. También puede utilizar variables de indicador (o estructuras de indicador) con estructuras de host.

Tareas relacionadas

[Recuperación de una única fila de datos en una estructura de host](#)

Si sabe que la consulta devuelve varios valores de columna para una única fila, puede especificar una estructura de host para contener los valores de columna.

Referencia relacionada

[Estructuras de host en C y C++](#)

Una estructura de host C contiene un grupo ordenado de campos de datos.

[Estructuras host en COBOL](#)

Una estructura host en COBOL es un conjunto con nombre de variables host que se definen en la WORKING-STORAGE SECTION o LINKAGE SECTION del programa.

[Estructuras de host en PL/I](#)

Una estructura de host PL/I es una estructura de host que contiene niveles subordinados de escalares. Puede utilizar el nombre de la estructura como notación abreviada para hacer referencia a la lista de escalares.

Variables de indicación, matrices y estructuras

Una variable de indicación se asocia con una variable host concreta. Cada variable de indicación contiene un valor entero pequeño que indica alguna información sobre la variable host asociada. Las estructuras y matrices de indicador tienen el mismo propósito para las estructuras y matrices de variables de host.

Puede utilizar variables de indicador para realizar las siguientes acciones:

- Determinar si el valor de una variable de host de salida asociada es nulo o indicar que el valor de una variable de host de entrada es nulo
- Determinar la longitud original de una cadena de caracteres que se truncó cuando se asignó a una variable de host
- Determinar que un valor de carácter no se pudo convertir cuando se asignó a una variable de host
- Determinar la parte de segundos de un valor de tiempo que se truncó cuando se asignó a una variable host
- Indica que la columna de destino de la variable de host debe establecerse en su valor DEFAULT definido, o que el valor de la variable de host es UNASSIGNED y su columna de destino debe tratarse como si no hubiera aparecido en la instrucción.

Puede utilizar matrices de variables indicadoras y estructuras indicadoras para realizar estas mismas acciones para elementos individuales en matrices y estructuras de datos de host.

Si proporciona una variable indicadora para la variable X, cuando Db2 recupera un valor nulo para X, pone un valor negativo en la variable indicadora y no actualiza X. Su programa debe comprobar la variable indicadora antes de usar X. Si la variable indicadora es negativa, sabes que X es nula y que cualquier valor que encuentres en X es irrelevante. Cuando su programa utiliza la variable X para asignar un valor

nulo a una columna, el programa debe establecer la variable indicadora en un número negativo. Db2 y, a continuación, asigna un valor nulo a la columna e ignora cualquier valor en X.

Un array de variables indicadoras contiene una serie de pequeños números enteros para ayudarle a determinar la información asociada al elemento correspondiente en un array de datos host. Cuando recupera datos en una matriz de variables de host, puede comprobar los valores en la matriz de indicadores asociada para determinar cómo manejar cada elemento de datos. Si un valor en la matriz de indicadores asociada es negativo, puede ignorar el contenido del elemento correspondiente en la matriz de variables de host. Los valores en las matrices de indicadores tienen los siguientes significados:

En la salida a la aplicación, la variable indicadora normal puede contener los siguientes valores:

0

Un 0 (cero) o un valor positivo de la variable indicadora especifica que el primer identificador de host proporciona el valor de esta referencia de variable de host.

-1

Un valor " -1 " indica que el valor seleccionado era el valor nulo.

-2

Un valor de " -2 " de la variable indicadora indica que se ha producido un error de conversión numérica (como una división por 0 o un desbordamiento). O indica un resultado nulo debido a advertencias de conversión de cadena de caracteres.

-3

Un valor de " -3 " de la variable indicadora indica que no se ha devuelto ningún valor. Un valor e -3 e de la variable indicadora también puede indicar un resultado nulo porque la fila actual del cursor está en un agujero que se detectó durante una FETCH de varias filas.

entero positivo

Si la variable indicadora contiene un entero positivo, el valor recuperado se trunca y el entero es la longitud original de la cadena.

entero positivo

La parte de segundos de una hora si la hora se trunca al asignarla a una variable de host.

Al entrar en Db2, las variables indicadoras normales o las variables indicadoras extendidas pueden contener los siguientes valores:

0, o número entero positivo

Especifica un valor no nulo. Un 0 (cero) o un valor positivo de la variable indicadora especifica que el primer identificador de host proporciona el valor de esta referencia de variable de host.

-1, -2, -3, -4, -6

Especifica un valor nulo.

-5

- Si las variables de indicador extendido no están habilitadas, un valor de tipo " -5 " especifica el valor NULL.
- Si las variables de indicador extendido están habilitadas, un valor de -5 especifica el valor DEFAULT. Un valor de tipo " -5 " especifica que la columna de destino para esta variable de host debe establecerse en su valor DEFAULT.

-7

- Si las variables de indicador extendido no están habilitadas, un valor de tipo " -7 " especifica el valor NULL.
- Si las variables de indicador extendido están habilitadas, un valor de tipo " -7 " especifica el valor UNASSIGNED. Un valor " -7 " especifica que la columna de destino para esta variable de host debe tratarse como si no se hubiera especificado en la instrucción.

Una estructura de indicadores es una matriz de variables enteras de semicadena que admite una estructura de host específica. Si los valores de columna que su programa recupera en una estructura de host pueden ser nulos, puede adjuntar un nombre de estructura indicador al nombre de la estructura

de host. Este nombre permite a Db2 notificar a su programa cada valor nulo que devuelve a una variable host en la estructura host.

Conceptos relacionados

Espacios vacíos en la tabla de resultados de un cursor desplazable

Un espacio vacío en la tabla de resultados significa que la tabla de resultados no se reduce para llenar el espacio de filas suprimidas. Tampoco se contrae para llenar el espacio de filas que se han actualizado y ya no satisfacen la condición de búsqueda. No se puede acceder a un espacio vacío de supresión o actualización. Sin embargo, puede eliminar espacios vacíos en situaciones específicas.

Tareas relacionadas

Ejecución de sentencias SQL utilizando un cursor de conjunto de filas

Puede utilizar cursos de conjunto de filas para ejecutar sentencias FETCH de varias filas, sentencias UPDATE de posición y sentencias DELETE de posición.

Referencia relacionada

Variables de indicador en Assembler

Una variable indicadora es un entero de 2 bytes (DS HL2). Declare las variables indicadoras de la misma forma que las variables host. Puede mezclar las declaraciones de dos tipos de variables.

Variables de indicador, matrices de indicador y matrices de indicador de estructura de host en C y C++

Una variable indicadora es un entero de 2 bytes (entero corto). Una matriz de variable indicadora es una matriz de enteros de 2 bytes (entero corto). Declare las variables indicadoras de la misma forma que las variables host. Puede mezclar las declaraciones de dos tipos de variables.

Variables indicadoras, matrices de indicador y matrices de indicador de estructura de host en COBOL

Una variable de indicador COBOL es un entero binario de 2 bytes. Una matriz de variables de indicador COBOL es una matriz en la que cada elemento se declara como un entero binario de 2 bytes. Puede utilizar matrices de variables de indicador para dar soporte a estructuras de host COBOL.

Variables indicadoras en Fortran

Una variable indicadora es un entero de 2 bytes (INTEGER*2). Declare las variables indicadoras de la misma forma que las variables host. Puede mezclar las declaraciones de dos tipos de variables.

Variables de indicador en PL/I

Una variable indicadora es un entero de 2 bytes (o un entero declarado como BIN FIXED(15)). Una matriz de variable indicadora es una matriz de enteros de 2 bytes. Declare las variables indicadoras de la misma forma que las variables host. Puede mezclar las declaraciones de dos tipos de variables.

Establecimiento del CCSID para variables host

Todos los datos de serie Db2, distintos a los datos binarios, tienen un esquema de codificación y un ID del conjunto de caracteres codificado (CCSID) asociado a ellos. Se puede asociar un esquema de codificación y un CCSID a variables de host individuales. A continuación, los datos de esas variables de host se asocian a ese esquema de codificación y CCSID.

Procedimiento

Especifique la instrucción DECLARE VARIABLE después de la declaración de la variable de host correspondiente y antes de su primera referencia a dicha variable de host.

Esta declaración asocia un esquema de codificación y un CCSID con variables de host individuales. Puede utilizar esta declaración en aplicaciones SQL estáticas o dinámicas.

Restricción: No puede utilizar la instrucción DECLARE VARIABLE para controlar el CCSID y el esquema de codificación de los datos que recupera o actualiza mediante un SQLDA.

La instrucción DECLARE VARIABLE tiene los siguientes efectos en una variable de host:

- Cuando se utiliza la variable de host para actualizar una tabla, el subsistema local o el servidor remoto asume que los datos de la variable de host están codificados con el CCSID y el esquema de codificación que asigna la instrucción DECLARE VARIABLE.

- Cuando recupera datos de una tabla local o remota en la variable host, los datos recuperados se convierten al CCSID y al esquema de codificación que son asignados por la instrucción DECLARE VARIABLE.

Ejemplo

Supongamos que está escribiendo un programa en C que se ejecuta en un subsistema de e Db2 for z/OS . El subsistema tiene un esquema de codificación de aplicaciones EBCDIC. El programa C recupera datos de las siguientes columnas de una tabla local que se define con la opción CCSID UNICODE:

```
PARTNUM CHAR(10)
JPNNAME GRAPHIC(10)
ENGNAME VARCHAR(30)
```

Debido a que el esquema de codificación de la aplicación para el subsistema es EBCDIC, los datos recuperados son EBCDIC. Para que los datos recuperados sean Unicode, utilice sentencias DECLARE VARIABLE para especificar que los datos recuperados de estas columnas están codificados en los CCSID Unicode predeterminados para el subsistema.

Supongamos que desea recuperar los datos de caracteres en Unicode CCSID 1208 y los datos gráficos en Unicode CCSID 1200. Utilice las siguientes instrucciones DECLARE VARIABLE:

```
EXEC SQL BEGIN DECLARE SECTION;
char hvpartnum[11];
EXEC SQL DECLARE :hvpartnum VARIABLE CCSID 1208;
sqldbchar hvjpnname[11];
EXEC SQL DECLARE :hvjpnname VARIABLE CCSID 1200;
struct {
    short len;
    char d[30];
} hvengname;
EXEC SQL DECLARE :hvengname VARIABLE CCSID 1208;
EXEC SQL END DECLARE SECTION;
```

Referencia relacionada

[DECLARE VARIABLE declaración \(Db2 SQL\)](#)

Determinar qué causó un error al recuperar datos en una variable de host

Los errores que se producen cuando Db2 pasa datos a variables de host en una aplicación suelen deberse a un problema de conversión de un tipo de datos a otro. Estos errores no afectan a la posición del cursor.

Acerca de esta tarea

Por ejemplo, supongamos que introduce un valor entero de 32768 en una variable de host de tipo SMALLINT. La conversión podría causar un error si no proporciona suficiente información de conversión a Db2.

La variable a la que Db2 asigna los datos se denomina *variable de host de salida*. Si proporciona una variable indicadora para la variable host de salida o si no se requiere conversión de tipo de datos, en la mayoría de los casos Db2 devuelve un SQLCODE positivo para la fila. En otros casos en los que se producen problemas de conversión de datos, Db2 devuelve un SQLCODE negativo para esa fila. Independientemente del SQLCODE de la fila, no se asignan nuevos valores a la variable host ni a las variables posteriores de esa fila. Los valores que ya están asignados a variables permanecen asignados. Incluso cuando se devuelve un SQLCODE negativo para una fila, el procesamiento de la instrucción continúa y Db2 devuelve un SQLCODE positivo para la instrucción (SQLSTATE 01668, SQLCODE +354).

Procedimiento

Para determinar qué causó un error al recuperar datos en una variable de host:

1. Cuando Db2 devuelve SQLCODE = +354, utilice la instrucción GET DIAGNOSTICS con la opción NUMBER para determinar el número de errores y advertencias.

Por ejemplo, supongamos que no se proporcionan variables indicadoras para los valores que devuelve la siguiente sentencia:

```
FETCH FIRST ROWSET FROM C1 FOR 10 ROWS INTO :hva_col1, :hva_col2;
```

Db2 registra un SQLCODE negativo para cada fila con un error y continúa procesando hasta que se recuperan las 10 filas. Cuando se devuelve SQLCODE = +354 para la sentencia, puede utilizar la sentencia GET DIAGNOSTICS para determinar qué errores se produjeron en qué filas. La siguiente declaración devuelve num_filas = 10 y num_cond = 3:

```
GET DIAGNOSTICS :num_rows = ROW_COUNT, :num_cond = NUMBER;
```

2. Para investigar los errores y advertencias, utilice sentencias GET DIAGNOSTIC adicionales con la opción CONDITION.

Por ejemplo, para investigar las tres condiciones que se informaron en el ejemplo del paso anterior, utilice las siguientes afirmaciones:

Tabla 84. OBTENER DIAGNÓSTICOS para investigar afecciones

Sentencia	Salida
GET DIAGNOSTICS CONDITION 3 :sqlstate = RETURNED_SQLSTATE, :sqlcode = DB2_RETURNED_SQLCODE, :row_num = DB2_ROW_NUMBER;	sqlstate = 22003 sqlcode = -304 row_num = 5
GET DIAGNOSTICS CONDITION 2 :sqlstate = RETURNED_SQLSTATE, :sqlcode = DB2_RETURNED_SQLCODE, :row_num = DB2_ROW_NUMBER;	sqlstate = 22003 sqlcode = -802 row_num = 7
GET DIAGNOSTICS CONDITION 1 :sqlstate = RETURNED_SQLSTATE, :sqlcode = DB2_RETURNED_SQLCODE, :row_num = DB2_ROW_NUMBER;	sqlstate = 01668 sqlcode = +354 row_num = 0

Este resultado muestra que la quinta fila tiene un error de asignación de datos (-304) para la columna 1 y que la séptima fila tiene un error de asignación de datos (-802) para la columna 2. Estas filas no contienen datos válidos y no deben utilizarse.

Conceptos relacionados

[Variables de indicación, matrices y estructuras](#)

Una variable de indicación se asocia con una variable host concreta. Cada variable de indicación contiene un valor entero pequeño que indica alguna información sobre la variable host asociada. Las estructuras y matrices de indicador tienen el mismo propósito para las estructuras y matrices de variables de host.

Referencia relacionada

[GET DIAGNOSTICS declaración \(Db2 SQL\)](#)

Información relacionada

[+354 \(Db2 Codes\)](#)

Acceder a un módulo predeterminado de la aplicación

Si su programa de aplicación utiliza actualmente LOAD DSNHDECP, considere cambiar el programa de aplicación para utilizar la dirección DECP que devuelve ICFID 373, DSNALI o DSNRLI.

Acerca de esta tarea

Al utilizar la dirección DECP que devuelve IFCID 373, DSNALI o DSNRLI, se garantiza que está utilizando el mismo módulo DECP que se utilizó para iniciar Db2. También permite que el código se salte la

carga por completo, solo después de conectarse correctamente a Db2. Db2 carga DSNHDECP en un almacenamiento global paginable para que todos los programas puedan compartirlo.

Compatibilidad de SQL y tipos de datos de lenguaje

Los tipos de datos de variable host que se utilizan en sentencias SQL deben ser compatibles con los tipos de datos de las columnas con las que tiene intención de utilizarlos.

Al decidir los tipos de datos de las variables de host, tenga en cuenta las siguientes reglas y recomendaciones:

- Los tipos de datos numéricos son compatibles entre sí:

Assembler

Una columna SMALLINT, INTEGER, BIGINT, DECIMAL o FLOAT es compatible con una variable de host ensamblador numérico.

Fortran

Una columna INTEGER es compatible con cualquier variable de host de tipo entero (Fortran) que esté definida como INTEGER*2, INTEGER*4, REAL, REAL*4, REAL*8 o DOUBLE PRECISION.

PL/I

Una columna SMALLINT, INTEGER, BIGINT, DECIMAL o FLOAT es compatible con una variable de host PL/I de BIN FIXED(15), BIN FIXED(31), DECIMAL(s,p) o BIN FLOAT(*n*), donde *n* es de 1 a 53, o DEC FLOAT(*m*) donde *m* es de 1 a 16.

- Los tipos de datos de caracteres son compatibles entre sí:

Assembler

Una columna CHAR, VARCHAR o CLOB es compatible con una variable de host de caracteres ensamblador de longitud fija o variable.

C/C++

Una columna CHAR, VARCHAR o CLOB es compatible con una forma estructurada de un solo carácter, terminada en NUL o VARCHAR de una variable de host de carácter C.

COBOL

Una columna CHAR, VARCHAR o CLOB es compatible con una variable de host de caracteres COBOL de longitud fija o variable.

Fortran

Una columna CHAR, VARCHAR o CLOB es compatible con una variable de host de caracteres de tipo Fortran.

PL/I

Una columna CHAR, VARCHAR o CLOB es compatible con una variable de host de caracteres PL/I de longitud fija o variable.

- Los tipos de datos de caracteres son parcialmente compatibles con los localizadores CLOB. Puede realizar las siguientes asignaciones:

- Asignar un valor en un localizador CLOB a una columna CHAR o VARCHAR
- Utilice una instrucción SELECT INTO para asignar una columna CHAR o VARCHAR a una variable de host localizador CLOB.
- Asignar un parámetro de salida CHAR o VARCHAR de una función definida por el usuario o un procedimiento almacenado a una variable de host localizador CLOB.
- Utilice una sentencia de asignación SET para asignar una variable de transición CHAR o VARCHAR a una variable de host localizador CLOB.
- Utilice una instrucción VALUES INTO para asignar un parámetro de función CHAR o VARCHAR a una variable de host localizador CLOB.

Sin embargo, no puede utilizar una sentencia FETCH para asignar un valor en una columna CHAR o VARCHAR a una variable de host localizador CLOB.

- Los tipos de datos gráficos son compatibles entre sí:

Assembler

Una columna GRAPHIC, VARGRAPHIC o DBCLOB es compatible con una variable host de carácter gráfico ensamblador de longitud fija o variable.

C/C++

Una columna GRAPHIC, VARGRAPHIC o DBCLOB es compatible con una forma estructurada de una variable de host gráfica C de un solo carácter, terminada en NUL o VARGRAPHIC.

COBOL

Una columna GRAPHIC, VARGRAPHIC o DBCLOB es compatible con una variable de host de cadena gráfica COBOL de longitud fija o variable.

PL/I

Una columna GRAPHIC, VARGRAPHIC o DBCLOB es compatible con una variable de host de carácter gráfico PL/I de longitud fija o variable.

- Los tipos de datos gráficos son parcialmente compatibles con los localizadores DBCLOB. Puede realizar las siguientes asignaciones:
 - Asignar un valor en un localizador DBCLOB a una columna GRAPHIC o VARGRAPHIC
 - Utilice una instrucción SELECT INTO para asignar una columna GRAPHIC o VARGRAPHIC a una variable de host localizador DBCLOB.
 - Asignar un parámetro de salida GRAPHIC o VARGRAPHIC de una función definida por el usuario o un procedimiento almacenado a una variable de host localizador DBCLOB.
 - Utilice una sentencia de asignación SET para asignar una variable de transición GRAPHIC o VARGRAPHIC a una variable de host localizador DBCLOB.
 - Utilice una instrucción VALUES INTO para asignar un parámetro de función GRAPHIC o VARGRAPHIC a una variable de host localizador DBCLOB.

Sin embargo, no puede utilizar una sentencia FETCH para asignar un valor en una columna GRAPHIC o VARGRAPHIC a una variable de host localizador DBCLOB.

- Los tipos de datos binarios son compatibles entre sí.
- Los tipos de datos binarios son parcialmente compatibles con los localizadores BLOB. Puede realizar las siguientes asignaciones:
 - Asignar un valor en un localizador BLOB a una columna BINARY o VARBINARY.
 - Utilice una instrucción SELECT INTO para asignar una columna BINARY o VARBINARY a una variable de host localizador BLOB.
 - Asignar un parámetro de salida BINARIO o VARBINARIO de una función definida por el usuario o un procedimiento almacenado a una variable host localizadora BLOB.
 - Utilice una sentencia de asignación SET para asignar una variable de transición BINARY o VARBINARY a una variable host localizadora BLOB.
 - Utilice una instrucción VALUES INTO para asignar un parámetro de función BINARY o VARBINARY a una variable de host localizadora BLOB.

Sin embargo, no puede utilizar una sentencia FETCH para asignar un valor en una columna BINARY o VARBINARY a una variable de host localizador BLOB.

- Los tipos de datos Datetime son compatibles con las variables de host de caracteres.

Fortran

Una columna BINARIA, VARBINARIA o BLOB o un localizador BLOB solo es compatible con una variable de host BLOB.

C:

Para datos BIT de longitud variable, utilice BINARY. Algunas funciones de manipulación de cadenas C procesan cadenas terminadas en NUL y otras funciones procesan cadenas que no están terminadas en NUL. Las funciones de manipulación de cadenas C que procesan cadenas terminadas en NUL no pueden manejar datos de bits porque estas funciones podrían malinterpretar un carácter NUL como un terminador NUL.

Assembler

Una columna FECHA, HORA o MARCA DE TIEMPO es compatible con una variable de host de carácter ensamblador de longitud fija o variable.

C/C++

Una columna FECHA, HORA o MARCA DE TIEMPO es compatible con una forma estructurada de un solo carácter, terminada en NUL o VARCHAR de una variable de host de carácter C.

COBOL

Una columna FECHA, HORA o MARCA DE TIEMPO es compatible con una variable de host de caracteres COBOL de longitud fija o variable.

Fortran

Una columna FECHA, HORA o MARCA DE TIEMPO es compatible con una variable de host de carácter e Fortran.

PL/I

Una columna FECHA, HORA o MARCA DE TIEMPO es compatible con una variable de host de caracteres PL/I de longitud fija o variable.

•

- La columna ROWID solo es compatible con una variable de host ROWID.
- Una variable anfitriona es compatible con un tipo distinto si el tipo de variable anfitriona es compatible con el tipo de origen del tipo distinto.
- Las columnas XML son compatibles con los tipos de variables de host XML, tipos de caracteres y tipos de cadenas binarias.

Recomendación: Utilice los tipos de variables de host XML para los datos de las columnas XML.

• Assembler

Puede asignar datos LOB a una variable de referencia de archivo (BLOB_FILE, CLOB_FILE y DBCLOB_FILE).

Cuando es necesario, Db2 convierte automáticamente una cadena de longitud fija en una cadena de longitud variable, o una cadena de longitud variable en una cadena de longitud fija.

Conceptos relacionados

Tipos diferenciados

Un tipo distinto es un tipo de datos definido por el usuario que comparte su representación interna con un tipo de datos incorporado (*su tipo de origen*), pero se considera un tipo de datos independiente e incompatible para la mayoría de las operaciones.

[Tipos de datos de variable host para datos XML en aplicaciones de SQL incorporado \(Db2 Programming for XML\)](#)

Referencia relacionada

Tipos de datos SQL y ensamblador equivalentes

Cuando declara variables host en los programas ensamblador, el precompilador utiliza tipos de datos SQL equivalentes. Cuando recupera datos de un tipo de datos SQL concreto en una variable host, asegúrese de que la variable host es de un tipo de datos equivalente.

SQL equivalente y tipos de datos C

Cuando declara variables host en los programas C, el precompilador utiliza tipos de datos SQL equivalentes. Cuando recupere datos de un tipo de datos SQL concreto en una variable host, tendrá que asegurarse de que la variable host sea de un tipo de datos equivalente.

SQL equivalente y tipos de datos COBOL

Cuando declara variables host en los programas COBOL, el precompilador utiliza tipos de datos SQL equivalentes. Cuando recupere datos de un tipo de datos SQL concreto en una variable host, tendrá que asegurarse de que la variable host sea de un tipo de datos equivalente.

SQL equivalente y tipos de datos Fortran

Cuando declara variables host en los programas Fortran, el precompilador utiliza tipos de datos SQL equivalentes. Cuando recupere datos de un tipo de datos SQL concreto en una variable host, asegúrese de que la variable host es de un tipo de datos equivalente.

SQL equivalente y tipos de datos PL/I

Cuando declara variables host en los programas PL/I, el precompilador utiliza tipos de datos SQL equivalentes. Cuando recupere datos de un tipo de datos SQL concreto en una variable host, tendrá que asegurarse de que la variable host sea de un tipo de datos equivalente.

SQL equivalente y tipos de datos REXX

Todos los datos REXX son datos de cadena. Por lo tanto, cuando un programa REXX asigna datos de entrada a una columna, Db2 convierte los datos de un tipo de cadena al tipo de columna. Cuando un programa REXX asigna datos de columna a una variable de salida, Db2 convierte los datos del tipo de columna a un tipo de cadena.

Uso de variables host en sentencias SQL

Utilice variables de host escalares en sentencias de SQL incorporado para representar un único valor. Las variables host son útiles para almacenar datos recuperados o para pasar valores que van a asignar o utilizar las comparaciones.

Cuando utilice variables de host, cumpla los siguientes requisitos:

- Debe declarar el nombre de la variable host en el programa host antes de utilizarla. Las variables del host siguen las convenciones de nomenclatura del lenguaje del host.
- Puede utilizar una variable de host para representar un valor de datos, pero no puede utilizarla para representar una tabla, vista o nombre de columna. Puede especificar nombres de tablas, vistas o columnas en tiempo de ejecución mediante SQL dinámico.
- Para utilizar una variable de host en una instrucción SQL, puede especificar cualquier nombre de variable de host válido que esté declarado de acuerdo con las reglas del lenguaje de host.
- Las variables de host que se utilizan en las sentencias SQL deben ir precedidas de dos puntos (:), de modo que Db2 pueda distinguir un nombre de variable de un nombre de columna. Cuando se utilizan variables de host fuera de las sentencias SQL, no se anteponen dos puntos. Los programas PL/I tienen las siguientes excepciones: Si la instrucción SQL cumple alguna de las siguientes condiciones, no preceda una variable de host o matriz de variables de host en esa instrucción con dos puntos:
 - La instrucción SQL está en un programa que también contiene una instrucción DECLARE VARIABLE.
 - La variable de host forma parte de una expresión de cadena, pero no es el único componente de la expresión de cadena.
- Para optimizar el rendimiento, asegúrese de que la declaración del lenguaje de programación se corresponda lo máximo posible con el tipo de datos de los datos asociados en la base de datos.
- Para las asignaciones y comparaciones entre una columna de tipo "Db2" y una variable host de un tipo de datos o longitud diferente, se esperan conversiones.
- Si utiliza el lenguaje de programación C (Db2 precompiler), asegúrese de que los nombres de las variables de host y de las matrices de variables de host sean únicos dentro del programa, incluso si las variables y las matrices de variables se encuentran en bloques, clases, procedimientos, funciones o subrutinas diferentes. Puede calificar los nombres con un nombre de estructura para hacerlos únicos.

Conceptos relacionados

Variables de host

Utilice variables host para pasar un único elemento de datos entre Db2 y la aplicación.

Asignación y comparación (Db2 SQL)

Variables de host (Db2 SQL)

Tareas relacionadas

Inclusión de SQL dinámico en el programa

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

Recuperación de una única fila de datos en variables host

Si sabe que la consulta devuelve una sola fila, puede especificar una o más variables host que contienen los valores de columna de la fila recuperada.

Acerca de esta tarea

Restricción: Estas instrucciones no se aplican si no sabes cuántas filas devolverá Db2 o si esperas que Db2 devuelva más de una fila. En estas situaciones, utilice un cursor. Un cursor permite que una aplicación devuelva un conjunto de filas y extraiga una fila o un conjunto de filas a la vez de la tabla de resultados.

Procedimiento

En la instrucción SELECT, especifique la cláusula INTO con el nombre de una o más variables de host para contener los valores recuperados. Especifique una variable para cada valor que se vaya a recuperar. El valor recuperado puede ser un valor de columna, un valor de una variable de host, el resultado de una expresión o el resultado de una función agregada.

Recomendación: Si desea asegurarse de que solo se devuelve una fila, especifique la cláusula FETCH FIRST 1 ROW ONLY. Considere utilizar la cláusula ORDER BY para controlar qué fila se devuelve. Si especifica tanto la cláusula ORDER BY como la cláusula FETCH FIRST, el orden se realiza en todo el conjunto de resultados antes de que se devuelva la primera fila.

Db2 asigna el primer valor de la fila de resultados a la primera variable de la lista, el segundo valor a la segunda variable, y así sucesivamente.

Si la sentencia SELECT devuelve más de una fila, Db2 devuelve un error y los datos que se devuelven son indefinidos e impredecibles.

Ejemplos

Ejemplo: Recuperar una sola fila en una variable de host

Supongamos que está recuperando los valores de las columnas LASTNAME y WORKDEPT de la base de datos DSN8C10.EMP tabla para un empleado en particular. Puede definir una variable de host en su programa para contener cada valor de columna y, a continuación, nombrar las variables de host en la cláusula INTO de la instrucción SELECT, como se muestra en el siguiente ejemplo de COBOL.

```
MOVE '000110' TO CBLEMPNO.  
EXEC SQL  
  SELECT LASTNAME, WORKDEPT  
    INTO :CBLNAME, :CBLDEPT  
   FROM DSN8C10.EMP  
 WHERE EMPNO = :CBLEMPNO  
END-EXEC.
```

En este ejemplo, la variable de host CBLEMPNO va precedida de dos puntos (:) en la instrucción SQL, pero no va precedida de dos puntos en la instrucción COBOL MOVE.

Este ejemplo también utiliza una variable de host para especificar un valor en una condición de búsqueda. La variable host CBLEMPNO se define para el número de empleado, de modo que se puede recuperar el nombre y el departamento de trabajo del empleado cuyo número es el mismo que el valor de la variable host, CBLEMPNO; en este caso, 000110.

En la sección DATA DIVISION de un programa COBOL, debe declarar las variables de host CBLEMPNO, CBLNAME y CBLDEPT para que sean compatibles con los tipos de datos en las columnas EMPNO, LASTNAME y WORKDEPT de la tabla EMP de la base de datos DSN8C10.

Ejemplo: Asegurarse de que una consulta devuelve solo una fila

Puede utilizar la cláusula FETCH FIRST 1 ROW ONLY en una instrucción SELECT para asegurarse de que solo se devuelve una fila. Esta acción evita que se devuelvan datos indefinidos e impredecibles cuando se especifica la cláusula INTO de la instrucción SELECT. La siguiente sentencia SELECT de ejemplo garantiza que solo una fila de la tabla "DSN8C10" se muestre en la página.EMP se devuelve la tabla.

```
EXEC SQL  
  SELECT LASTNAME, WORKDEPT  
    INTO :CBLNAME, :CBLDEPT  
   FROM DSN8C10.EMP
```

```
      FETCH FIRST 1 ROW ONLY  
END-EXEC.
```

Puede incluir una cláusula ORDER BY en el ejemplo anterior para controlar qué fila se devuelve. La siguiente sentencia SELECT de ejemplo garantiza que la única fila devuelta sea la que tiene un apellido que es el primero alfabéticamente.

```
EXEC SQL  
  SELECT LASTNAME, WORKDEPT  
    INTO :CBLNAME, :CBLDEPT  
    FROM DSN8810.EMP  
   ORDER BY LASTNAME  
      FETCH FIRST 1 ROW ONLY  
END-EXEC.
```

Ejemplo: Recuperar los resultados de los valores y expresiones de las variables del host en las variables del host

Cuando especifique una lista de elementos en la cláusula SELECT, esa lista puede incluir más que los nombres de las columnas de las tablas y vistas. Puede solicitar un conjunto de valores de columna mezclados con valores de variables de host y constantes. Por ejemplo, la siguiente consulta solicita los valores de varias columnas (EMPNO, LASTNAME y SALARY), el valor de una variable de host (RAISE) y el valor de la suma de una columna y una variable de host (SALARY y RAISE). Para cada uno de estos cinco elementos de la lista SELECT, se enumera una variable de host en la cláusula INTO.

```
MOVE 4476 TO RAISE.  
MOVE '000220' TO PERSON.  
EXEC SQL  
  SELECT EMPNO, LASTNAME, SALARY, :RAISE, SALARY + :RAISE  
    INTO :EMP-NUM, :PERSON-NAME, :EMP-SAL, :EMP-RAISE, :EMP-TTL  
    FROM DSN8C10.EMP  
   WHERE EMPNO = :PERSON  
END-EXEC.
```

La instrucción SELECT anterior devuelve los siguientes resultados. Los encabezados de columna representan los nombres de las variables de host.

EMP-NUM	PERSON-NAME	EMP-SAL	EMP-RAISE	EMP-TTL
=====	=====	=====	=====	=====
000220	LUTZ	29840	4476	34316

Ejemplo: Recuperar el resultado de una función agregada en una variable de host

Una consulta puede solicitar que se devuelvan valores de resumen de funciones agregadas y almacenar esos valores en variables de host. Por ejemplo, la siguiente consulta solicita que el resultado de la función AVG se almacene en la variable de host AVG-SALARY.

```
MOVE 'D11' TO DEPTID.  
EXEC SQL  
  SELECT WORKDEPT, AVG(SALARY)  
    INTO :WORK-DEPT, :AVG-SALARY  
    FROM DSN8C10.EMP  
   WHERE WORKDEPT = :DEPTID  
END-EXEC.
```

Tareas relacionadas

Recuperación de un conjunto de filas utilizando un cursor

En un programa de aplicación, puede recuperar un conjunto de filas de una tabla o una tabla de resultados que devuelve el procedimiento almacenado. Puede recuperar una o más filas a la vez.

Referencia relacionada

SELECT INTO declaración (Db2 SQL)

Determinación de si un valor recuperado en una variable host es nulo o está truncado

Antes de que su aplicación manipule los datos recuperados de Db2 en una variable host, determine si el valor es nulo. Determine también si se han truncado al asignarse a la variable. Puede utilizar las variables indicadores para obtener esta información.

Antes de empezar

Antes de determinar si un valor de columna recuperado es nulo o truncado, debe haber definido las variables de indicador, matrices y estructuras adecuadas.

Acerca de esta tarea

Se produce un error si no se utiliza una variable indicadora y Db2 recupera un valor nulo.

Procedimiento

Determinar el valor de la variable, matriz o estructura indicadora que está asociada con la variable, matriz o estructura anfitriona.

Esos valores tienen los siguientes significados:

Tabla 85. Significado de los valores en las variables indicadoras

Valor de la variable indicadora	Significado
Menor que cero	El valor de la columna es nulo. El valor de la variable de host no cambia con respecto a su valor anterior. Si el valor de la variable indicadora es " -2 ", el valor de la columna es nulo debido a un error de conversión numérica o de caracteres
Zero	El valor de la columna no es nulo. Si el valor de la columna es una cadena de caracteres, el valor recuperado no se trunca.
Entero positivo	El valor recuperado se trunca. El número entero es la longitud original de la cadena.

ejemplos

Ejemplo de prueba de una variable indicadora

Supongamos que ha definido la siguiente variable indicadora INDNULL para la variable anfitriona CBLPHONE.

```
EXEC SQL
  SELECT PHONENO
  INTO :CBLPHONE:INDNULL
  FROM DSN8C10.EMP
  WHERE EMPNO = :EMPID
END-EXEC.
```

A continuación, puede probar INDNULL para obtener un valor negativo. Si el valor es negativo, el valor correspondiente de PHONENO es nulo y puede ignorar el contenido de CBLPHONE.

Ejemplo de comprobación de una matriz de variables indicadoras

Supongamos que declara la siguiente matriz de indicadores INDNULL para la matriz de variables de host CBLPHONE.

```
EXEC SQL
  FETCH NEXT ROWSET CURS1
  FOR 10 ROWS
```

```
INTO :CBLPHONE :INDNULL  
END-EXEC .
```

Después de la sentencia FETCH de varias filas, puede comprobar si cada elemento de la matriz INDNULL tiene un valor negativo. Si un elemento es negativo, puede ignorar el contenido del elemento correspondiente en la matriz de variables de host CBLPHONE.

Ejemplo de comprobación de una estructura de indicadores en COBOL

El siguiente ejemplo define la estructura del indicador EMP-IND como una matriz que contiene seis valores y se corresponde con la estructura del host PEMP-ROW.

```
01 PEMP-ROW.  
  10 EMPNO          PIC X(6).  
  10 FIRSTNME.  
    49 FIRSTNME-LEN  PIC S9(4) USAGE COMP.  
    49 FIRSTNME-TEXT PIC X(12).  
  10 MIDINIT        PIC X(1).  
  10 LASTNAME.  
    49 LASTNAME-LEN  PIC S9(4) USAGE COMP.  
    49 LASTNAME-TEXT PIC X(15).  
  10 WORKDEPT       PIC X(3).  
  10 EMP-BIRTHDATE  PIC X(10).  
01 INDICATOR-TABLE.  
  02 EMP-IND         PIC S9(4) COMP OCCURS 6 TIMES.  
:  
MOVE '000230' TO EMPNO.  
:  
EXEC SQL  
  SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, BIRTHDATE  
  INTO :PEMP-ROW:EMP-IND  
  FROM DSN8C10.EMP  
  WHERE EMPNO = :EMPNO  
END-EXEC .
```

Puede probar la estructura del indicador EMP-IND para valores negativos. Si, por ejemplo, EMP-IND(6) contiene un valor negativo, la variable de host correspondiente en la estructura de host (EMP-BIRTHDATE) contiene un valor nulo.

Conceptos relacionados

Errores aritméticos y de conversión

Puede realizar un seguimiento de los errores aritméticos y de conversión utilizando variables indicadoras. Una variable indicadora contiene un valor entero pequeño que indica alguna información sobre la variable host asociada.

Tareas relacionadas

Declaración de variables host y variables de indicación

Puede utilizar variables host y variables de indicación en sentencias SQL del programa para pasar datos entre Db2 y la aplicación.

Actualización de datos utilizando variables de host

Cuando quiera actualizar un valor en una tabla Db2, pero no conoce el valor exacto hasta que el programa se ejecute, utilice variables host. Db2 puede cambiar un valor de tabla para que coincida con el valor de la variable de host.

Procedimiento

Para actualizar datos utilizando variables de host:

1. Declare las variables de host necesarias.
2. Especifique una sentencia UPDATE con los nombres de variables de host adecuados en la cláusula SET.

ejemplos

Ejemplo de actualización de una sola fila mediante el uso de una variable de host

El siguiente ejemplo COBOL cambia el número de teléfono de un empleado al valor de la variable de host NEWPHONE. El valor de identificación del empleado se transfiere a través de la variable de host EMPID.

```
MOVE '4246' TO NEWPHONE.  
MOVE '000110' TO EMPID.  
EXEC SQL  
    UPDATE DSN8C10.EMP  
        SET PHONENO = :NEWPHONE  
        WHERE EMPNO = :EMPID  
END-EXEC.
```

Ejemplo de actualización de una sola fila mediante el uso de una variable de host

El siguiente ejemplo da a los empleados de un departamento en particular un aumento salarial del 10 %. El valor del departamento se pasa a través de la variable de host DEPTID.

```
MOVE 'D11' TO DEPTID.  
EXEC SQL  
    UPDATE DSN8C10.EMP  
        SET SALARY = 1.10 * SALARY  
        WHERE WORKDEPT = :DEPTID  
END-EXEC.
```

Referencia relacionada

[UPDATE declaración \(Db2 SQL\)](#)

Inserción de una sola fila utilizando una variable host

Utilice variables host en la sentencia INSERT cuando no conoce al menos algunos de los valores a insertar hasta que se ejecuta el programa.

Acerca de esta tarea

Restricción: Estas instrucciones solo se aplican a la inserción de una sola fila. Si desea insertar varias filas, utilice matrices de variables de host o la forma de la instrucción INSERT que selecciona valores de otra tabla o vista.

Procedimiento

Especifique una instrucción INSERT con valores de columna en la cláusula VALUES. Especifique variables de host o una combinación de variables de host y constantes como valores de columna.

Db2 inserta el primer valor en la primera columna de la lista, el segundo valor en la segunda columna, y así sucesivamente.

Ejemplo

El siguiente ejemplo utiliza variables de host para insertar una sola fila en la tabla de actividades.

```
EXEC SQL  
    INSERT INTO DSN8C10.ACT  
        VALUES (:HV-ACTNO, :HV-ACTKWD, :HV-ACTDESC)  
END-EXEC.
```

Tareas relacionadas

[Inserción de varias filas de datos desde matrices de variables de host](#)

Utilice las variables de host en la sentencia INSERT cuando no conozca al menos algunos de los valores que se han de insertar hasta que el programa se ejecuta.

Referencia relacionada

[INSERT declaración \(Db2 SQL\)](#)

Utilización de matrices de variables de host en sentencias SQL

Utilice matrices de variables de lenguaje de host en sentencias de SQL incorporado para representar valores que el programa no conoce hasta que se ejecuta la consulta. Las matrices de variables de host son útiles para almacenar un conjunto de valores recuperados o para pasar un conjunto de valores que se van a insertar en una tabla.

Para utilizar una matriz de variables de host en una instrucción SQL, especifique cualquier matriz de variables de host válida que esté declarada de acuerdo con las reglas del lenguaje del host. Puede especificar matrices de variables de host en C o C++, COBOL y PL/I. Debe declarar la matriz en el programa de lenguaje principal antes de utilizarlo.

Las matrices de variables de lenguaje principal se definen mediante sentencias del lenguaje principal, como se explica en los temas siguientes:

- “[Matrices de variables de host en C y C++](#)” en la página 628
- “[Matrices de variables de host en COBOL](#)” en la página 696
- “[Matrices de variables de host en PL/I](#)” en la página 751

Consejo: Las matrices de variables de lenguaje principal no reciben soporte para los programas assembler, FORTRAN o REXX. Sin embargo, puede utilizar áreas de descriptor de SQL (SQLDA) para obtener resultados similares en cualquier lenguaje de host. Para obtener más información, consulte “[Definición de áreas de descriptor de SQL \(SQLDA\)](#)” en la página 502.

Solo se puede hacer referencia a las matrices de variables de lenguaje principal como una referencia simple en los contextos siguientes. En los diagramas de sintaxis, *host-variable-array* designa una referencia a una matriz de variables de host.

- En una sentencia `FETCH` para una captación de varias filas. Consulte [FETCH declaración \(Db2 SQL\)](#).
- En la forma `FOR n ROWS` de la sentencia `INSERT` con una matriz de variables de host para los datos de origen. Consulte [INSERT declaración \(Db2 SQL\)](#).
- En una sentencia `MERGE` con varias filas de datos de origen. Consulte [MERGE declaración \(Db2 SQL\)](#).
- En una sentencia `EXECUTE` para proporcionar un valor para un marcador de parámetro en una forma dinámica `FOR n ROWS` de la sentencia `INSERT` o una sentencia `MERGE`. Consulte [EXECUTE declaración \(Db2 SQL\)](#).

Conceptos relacionados

[Matrices de variables de lenguaje principal en PL/I, C, C++ y COBOL \(Db2 SQL\)](#)

[Matrices de variables de host](#)

Puede utilizar matrices de variables de host para pasar una matriz de datos entre Db2 y su aplicación. Una matriz de variables de host es una matriz de datos que se declara en el lenguaje de host para ser utilizada dentro de una instrucción SQL.

Tareas relacionadas

[Inserción de varias filas de datos desde matrices de variables de host](#)

Utilice las variables de host en la sentencia `INSERT` cuando no conozca al menos algunos de los valores que se han de insertar hasta que el programa se ejecuta.

[Recuperación de varias filas de datos en matrices de variables de host](#)

Si sabe que la consulta devuelve varias filas, puede especificar matrices de variables de host para almacenar los valores de columna recuperados.

Recuperación de varias filas de datos en matrices de variables de host

Si sabe que la consulta devuelve varias filas, puede especificar matrices de variables de host para almacenar los valores de columna recuperados.

Acerca de esta tarea

Puede utilizar matrices de variables de host para especificar un área de datos de programa que contenga varias filas de valores de columna. *Un cursor de conjunto de filas* (Db2) permite a una aplicación recuperar y procesar un conjunto de filas de la tabla de resultados del cursor.

Conceptos relacionados

Matrices de variables de host

Puede utilizar matrices de variables de host para pasar una matriz de datos entre Db2 y su aplicación. *Una matriz de variables de host* es una matriz de datos que se declara en el lenguaje de host para ser utilizada dentro de una instrucción SQL.

Matrices de variables de lenguaje principal en PL/I, C, C++ y COBOL (Db2 SQL)

Tareas relacionadas

Acceso a datos utilizando un cursor colocado por conjunto de filas

Un cursor colocado por conjunto de filas es un cursor que puede devolver una o varias filas para una única operación de captación. El cursor se coloca en el conjunto de filas que se van a captar.

Ejecución de sentencias SQL utilizando un cursor de conjunto de filas

Puede utilizar cursores de conjunto de filas para ejecutar sentencias FETCH de varias filas, sentencias UPDATE de posición y sentencias DELETE de posición.

Inserción de varias filas de datos desde matrices de variables de host

Utilice las variables de host en la sentencia INSERT cuando no conozca al menos algunos de los valores que se han de insertar hasta que el programa se ejecuta.

Inserción de varias filas de datos desde matrices de variables de host

Utilice las variables de host en la sentencia INSERT cuando no conozca al menos algunos de los valores que se han de insertar hasta que el programa se ejecuta.

Acerca de esta tarea

Puede utilizar *la forma FOR n ROWS* de la sentencia INSERT o la sentencia MERGE para insertar varias filas a partir de valores que se proporcionan en matrices de variables de host.

Cada matriz contiene valores para una columna de la tabla de destino. El primer valor de una matriz corresponde al valor de esa columna para la primera fila insertada, el segundo valor de la matriz corresponde al valor de esa columna en la segunda fila insertada, y así sucesivamente. Db2 determina los atributos de los valores basándose en la declaración de la matriz.

Ejemplo

Suponga que las matrices de variables de host HVA1, HVA2 y HVA3 se han declarado y rellenado con los valores que se van a insertar en las columnas ACTNO, ACTKWD y ACTDESC de la tabla ACT. La variable host NUM-ROWS especifica el número de filas a insertar, que debe ser menor o igual que la dimensión de cada matriz de variables host.

Puede insertar el número de filas especificadas en la variable de host NUM-ROWS utilizando la siguiente instrucción INSERT:

```
EXEC SQL
  INSERT INTO DSN8C10.ACT
    (ACTNO, ACTKWD, ACTDESC)
  VALUES (:HVA1, :HVA2, :HVA3)
  FOR :NUM-ROWS ROWS
END-EXEC.
```

Conceptos relacionados

Matrices de variables de lenguaje principal en PL/I, C, C++ y COBOL (Db2 SQL)

Tareas relacionadas

Recuperación de varias filas de datos en matrices de variables de host

Si sabe que la consulta devuelve varias filas, puede especificar matrices de variables de host para almacenar los valores de columna recuperados.

Referencia relacionada

[INSERT declaración \(Db2 SQL\)](#)
[MERGE declaración \(Db2 SQL\)](#)

Inserción de valores nulos en columnas utilizando las matrices o variables indicadoras

Si necesita insertar valores nulos en una columna, utilizar una matriz o variable indicadora es una forma fácil de hacerlo. Una variable o matriz de indicador está asociada a una variable de host o matriz de variables de lenguaje de host determinada.

Procedimiento

Para insertar valores nulos en columnas mediante variables indicadoras o matrices:

1. Definir una variable o matriz indicadora para una variable o matriz de host en particular.
2. Asignar un valor negativo a la variable o matriz del indicador.
3. Emitir la sentencia INSERT, UPDATE o MERGE adecuada con la variable o matriz de host y su variable o matriz de indicador.

Cuando Db2 procesa instrucciones INSERT, UPDATE y MERGE, comprueba la variable indicadora si existe. Si la variable indicadora es negativa, el valor de la columna es nulo. Si la variable indicadora es mayor que -1, la variable host asociada contiene un valor para la columna.

ejemplos

Ejemplo de cómo establecer un valor de columna en nulo mediante una variable indicadora

Supongamos que su programa lee un ID de empleado y un nuevo número de teléfono y debe actualizar la tabla de empleados con el nuevo número. El nuevo número podría faltar si el antiguo es incorrecto, pero aún no hay uno nuevo disponible. Si el nuevo valor de la columna PHONENO puede ser nulo, puede utilizar una variable indicadora, como se muestra en la siguiente instrucción UPDATE.

```
EXEC SQL
  UPDATE DSN8C10.EMP
    SET PHONENO = :NEWPHONE:PHONEIND
      WHERE EMPNO = :EMPID
END-EXEC.
```

Cuando NEWPHONE contiene un valor no nulo, establezca la variable indicadora PHONEIND en cero precediendo la sentencia UPDATE con la siguiente línea:

```
MOVE 0 TO PHONEIND.
```

Cuando NEWPHONE contiene un valor nulo, establezca PHONEIND en un valor negativo precediendo la instrucción UPDATE con la siguiente línea:

```
MOVE -1 TO PHONEIND.
```

Ejemplo de cómo establecer un valor de columna en nulo mediante el uso de una matriz de variables indicadoras

Suponga que las matrices de variables de host hva1 y hva2 se han llenado con valores que se van a insertar en las columnas ACTNO y ACTKWD. Asumir que la columna ACTDESC permite valores nulos. Para establecer la columna ACTDESC en nula, asigne -1 a los elementos de su matriz indicadora, ind3, como se muestra en el siguiente ejemplo:

```
/* Initialize each indicator array */
for (i=0; i<10; i++) {
  ind1[i] = 0;
  ind2[i] = 0;
```

```

        ind3[i] = -1;
    }

EXEC SQL
    INSERT INTO DSN8C10.ACT
        (ACTNO, ACTKWD, ACTDESC)
    VALUES (:hva1:ind1, :hva2:ind2, :hva3:ind3)
    FOR 10 ROWS;

```

Db2 ignora los valores de la matriz "hva3" y asigna los valores de la columna "ARTDESC" a nulo para las 10 filas que se insertan.

Tareas relacionadas

[Declaración de variables host y variables de indicación](#)

Puede utilizar variables host y variables de indicación en sentencias SQL del programa para pasar datos entre Db2 y la aplicación.

Recuperación de una única fila de datos en una estructura de host

Si sabe que la consulta devuelve varios valores de columna para una única fila, puede especificar una estructura de host para contener los valores de columna.

Acerca de esta tarea

En el siguiente ejemplo, supongamos que su programa COBOL incluye la siguiente instrucción SQL:

```

EXEC SQL
    SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
        INTO :EMPNO, :FIRSTNME, :MIDINIT, :LASTNAME, :WORKDEPT
        FROM DSN8C10.VEMP
        WHERE EMPNO = :EMPID
END-EXEC.

```

Si desea evitar enumerar variables de host, puede sustituir el nombre de una estructura, por ejemplo :PEMP, que contiene :EMPNO, :FIRSTNME, :MIDINIT, :LASTNAME y :WORKDEPT. El ejemplo dice entonces:

```

EXEC SQL
    SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
        INTO :PEMP
        FROM DSN8C10.VEMP
        WHERE EMPNO = :EMPID
END-EXEC.

```

Puede declarar una estructura de host usted mismo, o puede utilizar DCLGEN para generar una descripción de registro COBOL, una declaración de estructura PL/I o una declaración de estructura C que se corresponda con las columnas de una tabla.

Conceptos relacionados

[DCLGEN \(generador de declaraciones\)](#)

El programa debe declarar las tablas y vistas a las que accede. El generador de declaraciones de Db2, DCLGEN, crea estas sentencias DECLARE para programas C, COBOL y PL/I programs, y así no necesita codificar las sentencias él mismo. DCLGEN genera también las estructuras de variable de lenguaje principal correspondientes.

Estructuras host

Utilice las estructuras host para pasar un grupo de variables host entre Db2 y su aplicación.

[Ejemplo: Añadir declaraciones de DCLGEN a una biblioteca](#)

Puede utilizar DCLGEN para generar declaraciones de tabla y variable para programas C, COBOL y PL/I.

Si almacena estas declaraciones en una biblioteca, puede integrarlas posteriormente en el programa con una sola sentencia SQL INCLUDE.

Inclusión de SQL dinámico en el programa

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

Antes de empezar

Antes de utilizar SQL dinámico, considere si SQL estático o dinámico es la mejor técnica para su aplicación, y considere el tipo de SQL dinámico que desea utilizar. También hay que tener en cuenta las implicaciones de rendimiento del uso de SQL dinámico en los programas de aplicación. Para obtener información sobre los métodos que puede utilizar para mejorar el rendimiento de las sentencias SQL dinámicas, consulte [Mejora del rendimiento del SQL dinámico \(Db2 Performance\)](#).

Acerca de esta tarea

Conceptos introductorios

[Envío de sentencias SQL a Db2 \(Introducción a Db2 para z/OS\)](#)

[Aplicaciones de SQL dinámico \(Introducción a Db2 para z/OS\)](#)

El SQL dinámico prepara y ejecuta las sentencias SQL dentro de un programa, mientras el programa se está ejecutando.

Puede emitir sentencias de SQL dinámico en los siguientes contextos:

SQL interactivo

Un usuario introduce instrucciones SQL a través de SPUFI, el procesador de línea de comandos, o una herramienta interactiva, como QMF for Workstation. Db2 prepara y ejecuta esas sentencias como sentencias SQL dinámicas.

SQL dinámico incorporado

Su aplicación pone el origen SQL en variables host e incluye sentencias PREPARE y EXECUTE que dicen a Db2 que se prepare y ejecute el contenido de dichas variables host en el tiempo de ejecución. Debe precompilar y vincular los programas que incluyen SQL dinámico incorporado.

SQL incorporado diferido

Las sentencias de SQL incorporado diferido no son completamente estáticas ni completamente dinámicas. Al igual que las sentencias estáticas, las sentencias de SQL incorporado diferido se incorporan dentro de aplicaciones; sin embargo, al igual que las sentencias dinámicas, se preparan durante el tiempo de ejecución. Db2 procesa las sentencias de SQL incorporado diferido con normas de hora de enlace. Por ejemplo, Db2 utiliza el ID de autorización y calificador (determinados en la hora de enlace) como propietario del paquete o plan.

SQL dinámico ejecutado mediante funciones ODBC y JDBC

La aplicación contiene llamadas a funciones ODBC que pasan sentencias de SQL dinámico como argumentos. No es necesario precompilar y vincular los programas que utilizan llamadas a funciones ODBC.

El soporte de aplicaciones JDBC le permite escribir aplicaciones de SQL dinámico en Java.

Para la mayoría de los usuarios de Db2 , *el SQL estático*, que está integrado en un programa de lenguaje host y vinculado antes de que se ejecute el programa, proporciona una ruta directa y eficiente a los datos de Db2 . Puede utilizar SQL estático cuando sepa antes del tiempo de ejecución qué instrucciones SQL necesita ejecutar su aplicación.

Tareas relacionadas

[Establecimiento de límites para el uso de recursos de sistema utilizando el recurso de límite de recursos \(Db2 Performance\)](#)

[Habilitar la caché de instrucciones dinámicas para mejorar el rendimiento de SQL dinámico \(Db2 Performance\)](#)

Información relacionada

[Memoria caché de sentencias dinámicas \(libro blanco\)](#)

Diferencias entre SQL estático y dinámico

El SQL estático y el SQL dinámico son adecuados para distintas circunstancias. Es necesario tener en cuenta las diferencias entre los dos a la hora de determinar si a la aplicación le conviene más el SQL estático o el SQL dinámico.

Flexibilidad de SQL estático con variables de host

Conceptos introductorios

- [Instrucciones SQL estáticas \(Introducción a Db2 for z/OS \)](#)
- [Aplicaciones de SQL estático \(Introducción a DB2 para z/OS\)](#)
- [Envío de sentencias SQL a Db2 \(Introducción a Db2 para z/OS\)](#)
- [Aplicaciones de SQL dinámico \(Introducción a Db2 para z/OS\)](#)

Cuando utiliza SQL estático, no puede cambiar la forma de las sentencias de SQL a menos que realice cambios en el programa. Sin embargo, puede aumentar la flexibilidad de las sentencias estáticas utilizando variables de host.

Ejemplo : En el siguiente ejemplo, la instrucción UPDATE puede actualizar el salario de cualquier empleado. En el momento de la vinculación, sabes que los salarios deben actualizarse, pero no sabes hasta el momento de la ejecución qué salarios deben actualizarse y en qué cuantía.

```
01 IOAREA.  
02 EMPID          PIC X(06).  
02 NEW-SALARY     PIC S9(7)V9(2) COMP-3.  
: (Other declarations)  
READ CARDIN RECORD INTO IOAREA  
    AT END MOVE 'N' TO INPUT-SWITCH.  
: (Other COBOL statements)  
EXEC SQL  
    UPDATE DSN8C10.EMP  
        SET SALARY = :NEW-SALARY  
        WHERE EMPNO = :EMPID  
END-EXEC.
```

La sentencia (UPDATE) no cambia, ni tampoco su estructura básica, pero la entrada puede cambiar los resultados de la sentencia UPDATE.

Flexibilidad de SQL dinámico

¿Qué ocurre si un programa debe utilizar diferentes tipos y estructuras de sentencias SQL? Si hay tantos tipos y estructuras que no puede contener un modelo de cada uno, su programa podría necesitar SQL dinámico.

Puede utilizar uno de los siguientes programas para ejecutar SQL dinámico:

Db2 Query Management Facility (QMF)

Proporciona una interfaz alternativa a Db2 que acepta casi cualquier instrucción SQL

SPUFI

Acepta instrucciones SQL de un conjunto de datos de entrada y, a continuación, las procesa y ejecuta dinámicamente

Db2 command line processor

Acepta instrucciones SQL de un entorno de servicios del sistema UNIX.

Limitaciones de SQL dinámico

No puede utilizar algunas de las sentencias SQL de forma dinámica.

Procesamiento dinámico de SQL

Un programa que proporciona SQL dinámico acepta como entrada, o genera, una instrucción SQL en forma de cadena de caracteres. Puede simplificar la programación si planifica el programa para no utilizar

sentencias SELECT, o para utilizar solo aquellas que devuelvan un número conocido de valores de tipos conocidos. En el caso más general, en el que no se conocen de antemano las sentencias SQL que se ejecutarán, el programa suele seguir estos pasos:

1. Traduce los datos de entrada, incluidos los marcadores de parámetros, en una instrucción SQL
2. Prepara la instrucción SQL para su ejecución y adquiere una descripción de la tabla de resultados
3. Obtiene, para las sentencias SELECT, suficiente almacenamiento principal para contener los datos recuperados
4. Ejecuta la instrucción o recupera las filas de datos
5. Procesa la información devuelta
6. Maneja códigos de retorno de SQL.

Ejecución de SQL estático y dinámico

Para acceder a datos de Db2 , una instrucción SQL requiere una ruta de acceso. Dos factores importantes en el rendimiento de una instrucción SQL son la cantidad de tiempo que utiliza Db2 para determinar la ruta de acceso en tiempo de ejecución y si la ruta de acceso es eficiente. Db2 determina la ruta de acceso para un extracto en cualquiera de estos momentos:

- Cuando vinculas el plan o paquete que contiene la instrucción SQL
- Cuando se ejecuta la instrucción SQL

El momento en el que Db2 determina la ruta de acceso depende de estos factores:

- Si el estado se ejecuta de forma estática o dinámica
- Si el enunciado contiene variables de host de entrada
- Si el estado contiene una tabla temporal global declarada.

Instrucciones SQL estáticas sin variables de host de entrada

Para las sentencias SQL estáticas que no contienen variables de host de entrada, Db2 determina la ruta de acceso cuando vinculas el plan o el paquete. Esta combinación ofrece el mejor rendimiento porque la ruta de acceso ya está determinada cuando se ejecuta el programa.

Instrucciones SQL estáticas con variables de host de entrada

Para las sentencias SQL estáticas que tienen variables de host de entrada, el momento en el que Db2 determina la ruta de acceso depende de la opción de enlace REOPT que especifique: REOPT(NONE) o REOPT(ALWAYS). REOPT(NONE) es el valor predeterminado. No especifique REOPT(AUTO) o REOPT(ONCE); estas opciones solo son aplicables a las sentencias dinámicas. Db2 ignora REOPT(ONCE) y REOPT(AUTO) para sentencias SQL estáticas, porque Db2 almacena en caché solo sentencias SQL dinámicas.

Si especifica REOPT(NONE), Db2 determina la ruta de acceso en el momento de la vinculación, tal como lo hace cuando no hay variables de entrada.

Si especifica REOPT(ALWAYS), Db2 determina la ruta de acceso en el momento de la vinculación y de nuevo en el momento de la ejecución, utilizando los valores de los siguientes tipos de variables de entrada:

- Variables de host
- Marcadores de parámetro
- Registros especiales

Db2 debe dedicar tiempo adicional a determinar la ruta de acceso para las declaraciones en tiempo de ejecución. Sin embargo, si Db2 determina una ruta de acceso significativamente mejor utilizando los valores de las variables, es posible que se produzca una mejora general del rendimiento. Con REOPT(ALWAYS), Db2 optimiza las sentencias utilizando valores literales conocidos. Conocer los valores

literales puede ayudar a Db2 a elegir una ruta de acceso más eficiente cuando las columnas contienen datos sesgados. Db2 también puede reconocer qué particiones cumplen los requisitos si hay condiciones de búsqueda con variables de host en las claves de límite de los espacios de tabla particionados.

Con REOPT(ALWAYS) Db2 , la optimización no se reinicia desde el principio. Por ejemplo, Db2 no realiza transformaciones de consultas basadas en los valores literales. En consecuencia, las sentencias SQL estáticas que utilizan variables de host optimizadas con REOPT(ALWAYS) y sentencias SQL similares que utilizan valores literales explícitos pueden dar lugar a diferentes rutas de acceso.

Sentencias de SQL dinámico

Para sentencias de SQL dinámico, Db2 determina la vía de acceso en tiempo de ejecución, cuando se prepara la sentencia. El coste repetitivo de preparar una sentencia dinámica puede hacer que el rendimiento sea peor que el de las sentencias SQL estáticas. Sin embargo, si ejecuta la misma instrucción SQL a menudo, puede utilizar la caché de instrucciones dinámicas para disminuir el número de veces que deben prepararse esas instrucciones dinámicas.

Instrucciones SQL dinámicas con variables de host de entrada

Cuando vincule aplicaciones que contengan sentencias SQL dinámicas con variables de host de entrada, considere la posibilidad de utilizar las opciones de vinculación REOPT(ALWAYS), REOPT(ONCE) o REOPT(AUTO), en lugar de la opción REOPT(NONE).

Utilice REOPT(ALWAYS) cuando no esté utilizando la caché de sentencias dinámicas. Db2 determina la ruta de acceso para las sentencias en cada EXECUTE u OPEN de la sentencia. Esta opción garantiza la mejor ruta de acceso para una instrucción, pero el uso de REOPT(ALWAYS) puede aumentar el coste de las instrucciones SQL dinámicas utilizadas con frecuencia.

Por lo tanto, la opción REOPT(ALWAYS) no es una buena elección para consultas de gran volumen en menos de un segundo. Para consultas de gran volumen y ejecución rápida, el coste repetitivo de preparación puede superar el coste de ejecución de la instrucción. Las sentencias que se procesan con la opción REOPT(ALWAYS) se excluyen de la caché de sentencias dinámicas, incluso si el almacenamiento en caché de sentencias dinámicas está habilitado, porque Db2 no puede reutilizar las rutas de acceso cuando se especifica REOPT(ALWAYS).

Utilice REOPT(ONCE) o REOPT(AUTO) cuando utilice la caché de sentencias dinámicas:

- Si especifica REOPT(ONCE), Db2 determina y la ruta de acceso para las sentencias solo en el primer EXECUTE u OPEN de la sentencia. Guarda esa ruta de acceso en la caché de sentencias dinámicas y la utiliza hasta que la sentencia se invalida o se elimina de la caché. Esta reutilización de la ruta de acceso reduce el coste de preparación de las sentencias SQL dinámicas de uso frecuente que contienen variables de host de entrada; sin embargo, no tiene en cuenta los cambios en los valores de los marcadores de parámetros para las sentencias dinámicas.

La opción REOPT(ONCE) es ideal para aplicaciones de consulta ad-hoc como SPUFI, DSNTEP2, DSNTEP4, DSNTIAUL y QMF Db2 pueden optimizar mejor las sentencias conociendo los valores literales de registros especiales como CURRENT DATE y CURRENT TIMESTAMP, en lugar de utilizar estimaciones de factores de filtro predeterminados.

- Si especifica REOPT(AUTO), Db2 determina la ruta de acceso en tiempo de ejecución. Db2 , genera una nueva ruta de acceso para cada ejecución de una declaración con marcadores de parámetros, si determina que una nueva ruta de acceso puede mejorar el rendimiento.

Codificación de sentencias PREPARE para una optimización eficiente

Debe codificar sus declaraciones PREPARE para minimizar los gastos generales. Con REOPT(AUTO), REOPT(ALWAYS) y REOPT(ONCE), Db2 prepara una instrucción SQL al mismo tiempo que procesa OPEN o EXECUTE para la instrucción. Es decir, Db2 procesa la declaración como si especificara DEFER(PREPARE). Sin embargo, Db2 prepara el estado de cuenta dos veces en las siguientes situaciones:

- Su programa emite la sentencia DESCRIBE antes de la sentencia OPEN

- Usted emite la sentencia PREPARE con el parámetro INTO

Para la primera preparación, Db2 determina la ruta de acceso sin utilizar valores de variables de entrada. Para la segunda preparación, Db2 utiliza los valores de las variables de entrada para determinar la ruta de acceso. Esta preparación adicional puede disminuir el rendimiento.

Si especifica REOPT(ALWAYS), Db2 prepara el estado dos veces cada vez que se ejecuta.

Si especifica REOPT(ONCE), Db2 prepara el estado de cuenta dos veces solo cuando el estado de cuenta nunca se ha guardado en la memoria caché. Si la instrucción se ha preparado y guardado en la memoria caché, Db2 utilizará la versión guardada de la instrucción para completar la instrucción DESCRIBE.

Si especifica REOPT(AUTO), Db2 prepara inicialmente el estado sin utilizar valores de variables de entrada. Si la declaración se ha guardado en la memoria caché, para el OPEN o EXECUTE posterior, Db2 determina si se necesita una nueva ruta de acceso de acuerdo con los valores de las variables de entrada.

Para una sentencia que utiliza un cursor, puede evitar la doble preparación colocando la sentencia DESCRIBE después de la sentencia OPEN en su programa.

Si utiliza un gobierno predictivo y una instrucción SQL dinámica vinculada con REOPT(ALWAYS) o REOPT(ONCE) supera un umbral de advertencia de gobierno predictivo, su aplicación no recibe un SQLCODE de advertencia. Sin embargo, recibirá un error SQLCODE de la instrucción OPEN o EXECUTE.

Tareas relacionadas

[Reoptimización de sentencias de SQL en tiempo de ejecución \(Db2 Performance\)](#)

[Habilitar la caché de instrucciones dinámicas para mejorar el rendimiento de SQL dinámico \(Db2 Performance\)](#)

Referencia relacionada

[Acciones permitidas en instrucciones SQL \(Db2 SQL\)](#)

[REOPT opción bind \(comandos Db2 \)](#)

Possibles idiomas de host para aplicaciones SQL dinámicas

Los programas que utilizan SQL dinámico suelen estar escritos en ensamblador, C, PL/I, REXX y COBOL. Todas las sentencias SQL en programas REXX se consideran SQL dinámico.

Puede escribir sentencias SELECT no SELECT y de lista fija en cualquiera de los lenguajes compatibles con Db2 . Un programa que contenga una instrucción SELECT de lista variable es más difícil de escribir en Fortran, porque el programa no puede ejecutarse sin la ayuda de una subrutina para gestionar las variables de dirección (punteros) y la asignación de memoria.

La mayoría de los ejemplos de este tema están en PL/I. En las aplicaciones de muestra hay ejemplos más largos en forma de programas completos:

DSNTEP2

Procesa dinámicamente tanto las sentencias SELECT como las no SELECT. (PL/I).

DSNTIAD

Procesa solo las sentencias no SELECT de forma dinámica. (Montador).

DSNTIAUL

Procesa las sentencias SELECT de forma dinámica. (Montador).

La biblioteca *prefix.SDSNSAMP* contiene los programas de muestra. Puede ver los programas en línea o imprimirllos utilizando ISPF, IEBPTPCH o su propio programa de impresión.

Puede utilizar todas las formas de SQL dinámico en todas las versiones compatibles de COBOL.

Conceptos relacionados

[Programa de SQL dinámico COBOL de ejemplo](#)

Puede codificar sentencias SELECT de lista variable dinámicas en un programa COBOL. *Las sentencias SELECT de lista variable* son sentencias para las que no se conoce el número o los tipos de datos de las columnas que se van a devolver al escribir el programa.

IIInclusión de SQL dinámico para sentencias no SELECT en el programa

La forma más sencilla de utilizar SQL dinámico es utilizar sentencias no SELECT. Puesto que no es necesario asignar dinámicamente almacenamiento principal, puede escribir su programa en cualquier lenguaje host, incluyendo Fortran.

Procedimiento

Su programa debe seguir los siguientes pasos:

1. Incluya un SQLCA. Los requisitos para un área de comunicaciones SQL (SQLCA) son los mismos que para las sentencias SQL estáticas. Para REXX, Db2 incluye SQLCA automáticamente.
2. Cargar la instrucción SQL de entrada en un área de datos. El procedimiento para crear o leer la instrucción SQL de entrada no se trata aquí; la instrucción depende de su entorno y de las fuentes de información. Puede leer en instrucciones SQL completas, u obtener información para construir la instrucción a partir de conjuntos de datos, un usuario en un terminal, variables de programa previamente establecidas o tablas en la base de datos.
Si intenta ejecutar una instrucción SQL de forma dinámica que Db2 no permite, obtendrá un error SQL.
3. Ejecutar la declaración. Puede utilizar cualquiera de estos métodos:
 - EXECUTE IMMEDIATE
 - Preparar y ejecutar
4. Gestionar cualquier error que pueda producirse. Los requisitos son los mismos que para las sentencias SQL estáticas. El código de retorno de la instrucción SQL ejecutada más recientemente aparece en las variables de host SQLCODE y SQLSTATE o en los campos correspondientes de la SQLCA.

Conceptos relacionados

[SQL estático y dinámico de ejemplo en un programa C](#)

Los programas que acceden a Db2 pueden contener SQL estático, SQL dinámico o ambos.

[Aplicaciones ensambladoras que emiten sentencias SQL](#)

Puede codificar sentencias SQL en programas ensambladores siempre que pueda utilizar sentencias ejecutables.

[Aplicaciones C y C++ que emiten sentencias SQL](#)

Puede codificar sentencias SQL en un programa C o C++ siempre que pueda utilizar sentencias ejecutables.

[Aplicaciones COBOL que emiten sentencias SQL](#)

Puede codificar sentencias SQL en determinadas secciones del programa COBOL.

[Aplicaciones Fortran que emiten sentencias SQL](#)

Puede codificar sentencias SQL en un programa Fortran siempre que pueda colocar sentencias ejecutables. Si la sentencia SQL está dentro de una sentencia IF, el precompilador genera cualquier sentencia THEN y END IF necesaria.

[Aplicaciones PL/I que emiten sentencias SQL](#)

Puede codificar sentencias SQL en un programa PL/I siempre que pueda utilizar sentencias ejecutables.

[Aplicaciones REXX que emiten sentencias SQL](#)

Puede codificar instrucciones SQL en un programa REXX siempre que pueda utilizar comandos REXX.

Tareas relacionadas

[Comprobación de la ejecución de sentencias SQL](#)

Después de ejecutar una sentencia SQL, el programa debe comprobar si hay errores antes de confirmar los datos y manejar los errores que representan.

[Ejecución dinámica de una sentencia SQL utilizando EXECUTE IMMEDIATE](#)

En determinadas situaciones, es posible que desee que el programa prepare y ejecute dinámicamente una sentencia inmediatamente después de leerla.

[Ejecución dinámica de una sentencia SQL utilizando PREPARE y EXECUTE](#)

Como alternativa a la ejecución de una sentencia SQL inmediatamente después de que se lee, puede preparar y ejecutar una sentencia SQL en dos pasos. Este método de dos pasos es útil cuando necesita ejecutar una sentencia SQL varias veces con diferentes valores.

Incluir SQL dinámico para instrucciones SELECT de lista fija en su programa

Una instrucción SELECT de lista fija devuelve filas que contienen un número conocido de valores de un tipo conocido. Cuando se utiliza este tipo de declaración, se sabe de antemano exactamente qué tipos de variables de host hay que declarar para almacenar los resultados.

Acerca de esta tarea

El término "lista fija" no implica que deba saber de antemano cuántas filas de datos se devolverán. Sin embargo, debe conocer el número de columnas y los tipos de datos de esas columnas. Una instrucción SELECT de lista fija devuelve una tabla de resultados que puede contener cualquier número de filas; su programa examina esas filas de una en una, utilizando la instrucción FETCH. Cada búsqueda sucesiva devuelve el mismo número de valores que la anterior, y los valores tienen los mismos tipos de datos cada vez. Por lo tanto, puede especificar variables de host como lo hace para SQL estático.

Una ventaja de la lista fija SELECT es que puede escribirla en cualquiera de los lenguajes de programación que admite Db2 . Las sentencias SELECT dinámicas de lista variable requieren ensamblador, C, PL/I y COBOL.

Por ejemplo, supongamos que su programa recupera apellidos y números de teléfono ejecutando dinámicamente sentencias SELECT de esta forma:

```
SELECT LASTNAME, PHONENO FROM DSN8C10.EMP  
WHERE ... ;
```

El programa lee los estados desde un terminal y el usuario determina la cláusula WHERE.

Al igual que con las sentencias no SELECT, su programa coloca las sentencias en una variable de caracteres de longitud variable; llámela DSTRING. Finalmente, prepara una declaración de DSTRING, pero primero debe declarar un cursor para la declaración y darle un nombre.

Procedimiento

Para ejecutar una instrucción SELECT de lista fija de forma dinámica, su programa debe:

1. Incluya un SQLCA.
2. Cargar la instrucción SQL de entrada en un área de datos.

Los dos pasos anteriores son exactamente iguales, incluido el SQL dinámico para sentencias no SELECT en su programa.

3. Declare un cursor para el nombre de la declaración.

Las sentencias SELECT dinámicas no pueden utilizar INTO. Por lo tanto, debe utilizar un cursor para poner los resultados en variables de host.

Por ejemplo, cuando declare el cursor, utilice el nombre de la sentencia (llámela STMT) y asigne un nombre al propio cursor (por ejemplo, C1):

```
EXEC SQL DECLARE C1 CURSOR FOR STMT;
```

4. Prepare el extracto.

Preparar una declaración (STMT) de DSTRING. Esta es una posible declaración PREPARE:

```
EXEC SQL PREPARE STMT FROM :DSTRING ATTRIBUTES :ATTRVAR;
```

ATTRVAR contiene atributos que desea agregar a la instrucción SELECT, como FETCH FIRST 10 ROWS ONLY u OPTIMIZE for 1 ROW. En general, si la sentencia SELECT tiene atributos que entran en conflicto con los atributos de la sentencia PREPARE, los atributos de la sentencia SELECT tienen prioridad sobre los atributos de la sentencia PREPARE. Sin embargo, en este ejemplo, la instrucción

SELECT en DSTRING no tiene atributos especificados, por lo que Db2 utiliza los atributos en ATTRVAR para la instrucción SELECT.

Al igual que con las sentencias no SELECT, la SELECT de lista fija podría contener marcadores de parámetros. Sin embargo, este ejemplo no los necesita.

5. Abre el cursor.

La sentencia OPEN evalúa la sentencia SELECT denominada STMT.

Por ejemplo, sin marcadores de parámetros, utilice esta declaración:

```
EXEC SQL OPEN C1;
```

Si STMT contiene marcadores de parámetros, debe utilizar la cláusula USING de OPEN para proporcionar valores para todos los marcadores de parámetros en STMT. Si hay cuatro marcadores de parámetros en STMT, necesita la siguiente declaración:

```
EXEC SQL OPEN C1 USING :PARM1, :PARM2, :PARM3, :PARM4;
```

6. Recuperar filas de la tabla de resultados.

Por ejemplo, su programa podría ejecutar repetidamente una instrucción como esta:

```
EXEC SQL FETCH C1 INTO :NAME, :PHONE;
```

La característica clave de esta declaración es el uso de una lista de variables de host para recibir los valores devueltos por FETCH. La lista tiene un número conocido de elementos (en este caso, dos elementos, :NAME y :PHONE) de tipos de datos conocidos (ambos son cadenas de caracteres, de longitudes 15 y 4, respectivamente).

Puede utilizar esta lista en la sentencia FETCH solo porque ha programado el programa para que utilice solo SELECT de lista fija. Cada fila a la que apunte el cursor C1 debe contener exactamente dos valores de caracteres de la longitud adecuada. Si el programa va a manejar cualquier otra cosa, debe utilizar las técnicas para incluir SQL dinámico para instrucciones SELECT de lista variable en su programa.

7. Cierre el cursor.

Este paso es el mismo que para SQL estático.

Una declaración WHENEVER NOT FOUND en su programa puede nombrar una rutina que contenga esta declaración:

```
EXEC SQL CLOSE C1;
```

8. Gestionar los errores resultantes.

Este paso es el mismo que para el SQL estático, excepto por el número y los tipos de errores que pueden producirse.

Conceptos relacionados

SQL estático y dinámico de ejemplo en un programa C

Los programas que acceden a Db2 pueden contener SQL estático, SQL dinámico o ambos.

Aplicaciones ensambladoras que emiten sentencias SQL

Puede codificar sentencias SQL en programas ensambladores siempre que pueda utilizar sentencias ejecutables.

Aplicaciones C y C++ que emiten sentencias SQL

Puede codificar sentencias SQL en un programa C o C++ siempre que pueda utilizar sentencias ejecutables.

Aplicaciones COBOL que emiten sentencias SQL

Puede codificar sentencias SQL en determinadas secciones del programa COBOL.

Aplicaciones Fortran que emiten sentencias SQL

Puede codificar sentencias SQL en un programa Fortran siempre que pueda colocar sentencias ejecutables. Si la sentencia SQL está dentro de una sentencia IF, el precompilador genera cualquier sentencia THEN y END IF necesaria.

Aplicaciones PL/I que emiten sentencias SQL

Puede codificar sentencias SQL en un programa PL/I siempre que pueda utilizar sentencias ejecutables.

Aplicaciones REXX que emiten sentencias SQL

Puede codificar instrucciones SQL en un programa REXX siempre que pueda utilizar comandos REXX.

Tareas relacionadas

IInclusión de SQL dinámico para sentencias no SELECT en el programa

La forma más sencilla de utilizar SQL dinámico es utilizar sentencias no SELECT. Puesto que no es necesario asignar dinámicamente almacenamiento principal, puede escribir su programa en cualquier lenguaje host, incluyendo Fortran.

Inclusión de SQL dinámico para sentencias SELECT de lista variable en el programa

Una sentencia SELECT de lista variable devuelve filas que contienen un número desconocido de valores de tipo desconocido. Cuando utiliza este tipo de sentencia, no sabe de antemano exactamente qué clase de variables host debe declarar para almacenar los resultados.

Inclusión de SQL dinámico para sentencias SELECT de lista variable en el programa

Una sentencia SELECT de lista variable devuelve filas que contienen un número desconocido de valores de tipo desconocido. Cuando utiliza este tipo de sentencia, no sabe de antemano exactamente qué clase de variables host debe declarar para almacenar los resultados.

Acerca de esta tarea

Debido a que la instrucción SELECT de lista variable requiere variables puntero para el área descriptora SQL, no puede emitirse desde un programa de Fortran. Un programa de lenguaje de programación orientado a objetos (Fortran) puede llamar a una subrutina escrita en un lenguaje que admite variables de puntero (como PL/I o ensamblador), si necesita utilizar una instrucción SELECT de lista variable.

Procedimiento

Para ejecutar una instrucción SELECT de lista variable de forma dinámica, su programa debe seguir estos pasos:

1. Incluya un SQLCA.

Db2 realiza este paso para un programa REXX.

2. Cargar la instrucción SQL de entrada en un área de datos.

3. Preparar y ejecutar la declaración. Este paso es más complejo que para las SELECCIONES de lista fija.

Incluye los siguientes pasos:

a) Incluir un SQLDA (área de descriptor SQL).

Db2 realiza este paso para un programa REXX.

b) Declare un cursor y prepare la declaración de variables.

c) Obtener información sobre el tipo de datos de cada columna de la tabla de resultados.

d) Determinar el almacenamiento principal necesario para contener una fila de datos recuperados.

No se realiza este paso para un programa REXX.

e) Ponga direcciones de almacenamiento en el SQLDA para indicar dónde colocar cada elemento de los datos recuperados.

f) Abre el cursor.

g) Recuperar una fila.

h) Finalmente, cierre el cursor y libere la memoria principal.

Existen complicaciones adicionales para las declaraciones con marcadores de parámetros.

4. Gestionar cualquier error que pueda producirse.

ejemplos

Preparación de una sentencia SELECT de lista variable

Supongamos que su programa ejecuta dinámicamente sentencias SQL, pero esta vez sin ningún límite en su forma. Su programa lee los estados de una terminal y usted no sabe nada de ellos de antemano. Puede que ni siquiera sean sentencias SELECT.

Al igual que con las sentencias no SELECT, su programa coloca las sentencias en una variable de caracteres de longitud variable; llámela DSTRING. Su programa continúa preparando un estado de cuenta a partir de la variable y, a continuación, le da un nombre al estado de cuenta; llámelo STMT.

Ahora, el programa debe averiguar si la sentencia es un SELECT. Si lo es, el programa también debe averiguar cuántos valores hay en cada fila y cuáles son sus tipos de datos. La información procede de un área descriptora SQL (SQLDA).

Área de descriptores de SQL (SQLDA)

El SQLDA es una estructura que se utiliza para comunicarse con su programa, y el almacenamiento para él se suele asignar dinámicamente en tiempo de ejecución.

Para incluir el SQLDA en un programa PL/I o C, utilice:

```
EXEC SQL INCLUDE SQLDA;
```

Para ensamblador, utilícelo en el área de definición de almacenamiento de un CSECT:

```
EXEC SQL INCLUDE SQLDA
```

Para COBOL, utilice:

```
EXEC SQL INCLUDE SQLDA END-EXEC.
```

No puede incluir un SQLDA en un programa de lenguaje de scripting (Fortran) o REXX.

Obtención de información sobre la instrucción SQL

Un SQLDA puede contener un número variable de apariciones de SQLVAR, cada una de las cuales es un conjunto de cinco campos que describen una columna en la tabla de resultados de una instrucción SELECT.

El número de apariciones de SQLVAR depende de los siguientes factores:

- El número de columnas de la tabla de resultados que desea describir.
- Si desea que PREPARE o DESCRIBE ponga tanto los nombres de las columnas como las etiquetas en su SQLDA. Esta es la opción USAR AMBAS en la declaración PREPARAR o DESCRIBIR.
- Si alguna columna de la tabla de resultados es de tipo LOB o de tipo distinto.

La siguiente tabla muestra el número mínimo de instancias SQLVAR que necesita para una tabla de resultados que contenga n columnas.

Tabla 86. Número mínimo de SQLVAR para una tabla de resultados con n columnas

Tipo de DESCRIBE y contenido de la tabla de resultados	No USAR AMBOS	USO DE AMBOS
No hay tipos o LOB distintos	n	$2*n$
Tipos distintos pero sin LOB	$2*n$	$3*n$
LOB, pero sin tipos distintos	$2*n$	$2*n$
LOB y tipos distintos	$2*n$	$3*n$

Un SQLDA con n ocurrencias de SQLVAR se denomina *SQLDA simple*, un SQLDA con $2*n$ ocurrencias de SQLVAR *un SQLDA doble*, un SQLDA con $3*n$ ocurrencias de SQLVAR *un SQLDA triple*.

Un programa que admite sentencias SQL de todo tipo para su ejecución dinámica tiene dos opciones:

- Proporcionar la mayor base de datos SQLDA que pueda necesitar. El número máximo de columnas en una tabla de resultados es 750, por lo que un SQLDA para 750 columnas ocupa 33 016 bytes para un SQLDA simple, 66 016 bytes para un SQLDA doble o 99 016 bytes para un SQLDA triple. La mayoría de las sentencias SELECT no recuperan 750 columnas, por lo que el programa no suele utilizar la mayor parte de ese espacio.
- Proporcione un SQLDA más pequeño, con menos ocurrencias de SQLVAR. A partir de esto, el programa puede averiguar si la sentencia era un SELECT y, si lo era, cuántas columnas hay en su tabla de resultados. Si el resultado contiene más columnas de las que puede contener SQLDA, Db2 no devuelve ninguna descripción. Cuando esto ocurre, el programa debe adquirir almacenamiento para un segundo SQLDA que sea lo suficientemente largo como para contener las descripciones de las columnas, y volver a solicitar las descripciones a Db2. Aunque esta técnica es más complicada de programar que la primera, es más general.

¿Cuántas columnas debería permitir? Debe elegir un número que sea lo suficientemente grande para la mayoría de sus sentencias SELECT, pero que no desperdicie demasiado espacio; 40 es un buen compromiso. Para ilustrar lo que debe hacer en el caso de los estados que devuelven más columnas de las permitidas, el ejemplo de este análisis utiliza un SQLDA que está asignado para al menos 100 columnas.

Declaración de un cursor para el extracto

Como antes, necesita un cursor para el SELECT dinámico. Por ejemplo, escriba:

```
EXEC SQL  
  DECLARE C1 CURSOR FOR STMT;
```

Preparación del estado utilizando el SQLDA mínimo

Supongamos que su programa declara una estructura SQLDA con el nombre MINSQLDA, que tiene 100 apariciones de SQLVAR y SQLN establecido en 100. Para preparar una declaración a partir de la cadena de caracteres en DSTRING e introducir también su descripción en MINSQLDA, escriba lo siguiente:

```
EXEC SQL PREPARE STMT FROM :DSTRING;  
EXEC SQL DESCRIBE STMT INTO :MINSQLDA;
```

De forma equivalente, puede utilizar la cláusula INTO en la sentencia PREPARE:

```
EXEC SQL  
  PREPARE STMT INTO :MINSQLDA FROM :DSTRING;
```

No utilice la cláusula USING en ninguno de estos ejemplos. Por el momento, solo se utiliza el SQLDA mínimo. La siguiente figura muestra el contenido del SQLDA mínimo en uso.

Cabecera →	SQLDAID	SQLDABC	100	SQLD
------------	---------	---------	-----	------

Figura 32. La estructura mínima de SQLDA

SQLN determina lo que obtiene SQLVAR

El campo SQLN, que debe configurar antes de usar DESCRIBE (o PREPARE INTO), indica cuántas veces se asigna SQLVAR al SQLDA. Si DESCRIBE necesita más que eso, los resultados de DESCRIBE dependen del contenido de la tabla de resultados. Sea n el número de columnas de la tabla de resultados. Entonces:

- Si la tabla de resultados contiene al menos una columna de tipo distinto pero ninguna columna LOB, no especifica USING BOTH y $n \leq \text{SQLN} < 2 * n$, entonces Db2 devuelve información base de SQLVAR en las primeras n ocurrencias de SQLVAR, pero ninguna información de tipo distinta. La información básica de SQLVAR incluye:
 - Código de tipo de datos
 - Atributo de longitud (excepto para LOB)

- Nombre de columna o etiqueta
- Dirección de la variable de host
- Dirección de la variable de indicador
- De lo contrario, si SQLN es menor que el número mínimo de SQLVAR especificado en la tabla anterior, entonces Db2 no devuelve información en los SQLVAR.

Independientemente de si su SQLDA es lo suficientemente grande, siempre que ejecute DESCRIBE, Db2 devuelve los siguientes valores, que puede utilizar para crear una SQLDA del tamaño correcto:

- SQLD es 0 si la instrucción SQL no es un SELECT. De lo contrario, SQLD es el número de columnas de la tabla de resultados. El número de apariciones de SQLVAR que necesita para el SELECT depende del valor del séptimo byte de SQLDAID.
- El séptimo byte de SQLDAID es 2 si cada columna de la tabla de resultados requiere dos entradas SQLVAR. El séptimo byte de SQLDAID es 3 si cada columna de la tabla de resultados requiere tres entradas SQLVAR.

Si el enunciado no es un SELECT

Para averiguar si el estado es SELECT, su programa puede consultar el campo SQLD en MINSQLDA. Si el campo contiene 0, la instrucción no es un SELECT, la instrucción ya está preparada y su programa puede ejecutarla. Si no hay marcadores de parámetros en la declaración, puede utilizar:

```
EXEC SQL EXECUTE STMT;
```

(Si la declaración contiene marcadores de parámetros, debe utilizar un área de descriptor SQL)

Adquirir almacenamiento para un segundo SQLDA si es necesario

Ahora puede asignar almacenamiento para un segundo SQLDA de tamaño completo; llámelo FULSQLDA. La siguiente figura muestra su estructura.

encabezado fijo de 16 bytes	SQLDAID		SQLDABC	SQLN	SQLD
Conjunto de campos SQLVAR (elemento 1, 44 bytes)	tiposql	SQLLEN	SQLDATA	SQLIND	n*
Otros conjuntos de campos SQLVAR (elementos 2, 3, etc., 44 bytes cada uno)					
SQLVAR2 conjunto de campos (elemento 1, 44 bytes)	SQLLONGL	Reservado	SQLDATAL	m**	SQLNAME
Otros conjuntos SQLVAR2 campos SQLVAR2 (elementos 2, 3, etc., 44 bytes cada uno)					

* La longitud de la cadena de caracteres en SQLNAME.
SQLNAME es un área de 30 bytes que sigue inmediatamente al campo de longitud

** La longitud de la cadena de caracteres en SQLTNAME.
SQLTNAME es un área de 30 bytes que sigue inmediatamente al campo de longitud

Figura 33. La estructura SQLDA a tamaño completo

FULSQLDA tiene un encabezado de longitud fija de 16 bytes, seguido de una sección de longitud variable que consta de estructuras con el formato SQLVAR. Si la tabla de resultados contiene columnas LOB o columnas de tipo distinto, una sección de longitud variable que consta de estructuras con el formato SQLVAR2, sigue a las estructuras con formato SQLVAR. Todas las estructuras SQLVAR y las estructuras de SQLVAR2 tienen 44 bytes de longitud. El número de elementos SQLVAR y

SQLVAR2 que necesita está en el campo SQLD de MINSQLDA, y la longitud total que necesita para FULSQLDA ($16 + \text{SQLD} * 44$) está en el campo SQLDABC de MINSQLDA. Asignar esa cantidad de almacenamiento.

Describir de nuevo la sentencia SELECT

Después de asignar suficiente espacio para FULSQLDA, su programa debe seguir estos pasos:

1. Ponga el número total de ocurrencias de SQLVAR y SQLVAR2 en FULSQLDA en el campo SQLN de FULSQLDA. Este número aparece en el campo SQLD de MINSQLDA.
2. Describe la sentencia de nuevo en el nuevo SQLDA:

```
EXEC SQL DESCRIBE STMT INTO :FULSQLDA;
```

Después de que se ejecute la instrucción DESCRIBE, cada aparición de SQLVAR en el SQLDA de tamaño completo (FULSQLDA en nuestro ejemplo) contiene una descripción de una columna de la tabla de resultados en cinco campos. Si una ocurrencia SQLVAR describe una columna LOB o una columna de tipo distinto, la ocurrencia correspondiente SQLVAR2 contiene información adicional específica del LOB o del tipo distinto.

La siguiente figura muestra un SQLDA que describe dos columnas que no son columnas LOB ni columnas de tipo distinto.

Encabezado SQLDA	SQLDA			8816	200	200
Elemento SQLVAR 1 (44 bytes)	452	3	Indefinido	0	8	WORKDEPT
Elemento SQLVAR 2 (44 bytes)	453	4	Indefinido	0	7	PHONENO

Figura 34. Contenido de FULSQLDA después de ejecutar DESCRIBE

Adquirir almacenamiento para mantener una fila

Antes de recuperar filas de la tabla de resultados, su programa debe:

1. Analice cada descripción SQLVAR para determinar cuánto espacio necesita para el valor de la columna.
2. Obtener la dirección de algún área de almacenamiento del tamaño requerido.
3. Ponga esta dirección en el campo SQLDATA.

Si el campo SQLTYPE indica que el valor puede ser nulo, el programa también debe poner la dirección de una variable indicadora en el campo SQLIND. Las siguientes figuras muestran el área del descriptor SQL después de realizar ciertas acciones.

En la figura anterior, la sentencia DESCRIBE insertó todos los valores excepto la primera aparición del número 200. El programa insertó el número 200 antes de ejecutar DESCRIBE para indicar cuántas apariciones de SQLVAR se permiten. Si la tabla de resultados del SELECT tiene más columnas que esta, los campos SQLVAR no describen nada.

El primer SQLVAR corresponde a la primera columna de la tabla de resultados (la columna WORKDEPT). El elemento SQLVAR 1 contiene cadenas de caracteres de longitud fija y no permite valores nulos (SQLTYPE=452); el atributo de longitud es 3.

La siguiente figura muestra el SQLDA después de que su programa adquiere almacenamiento para los valores de columna y sus indicadores, y coloca las direcciones en los campos SQLDATA del SQLDA.

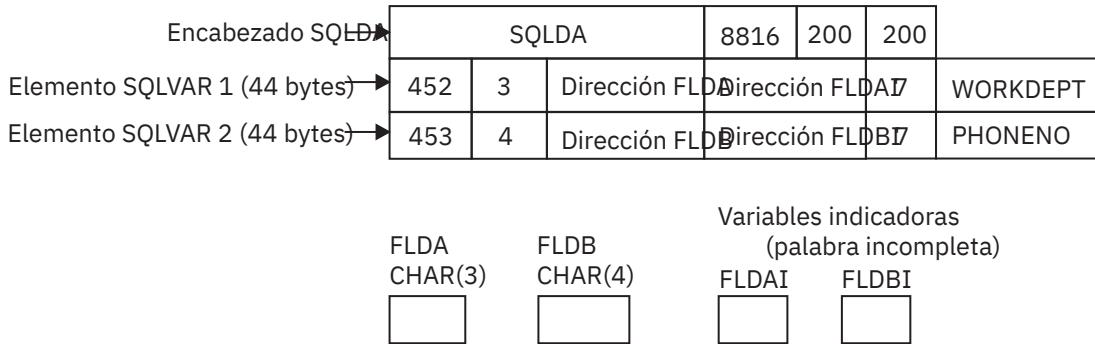


Figura 35. Área descriptora SQL tras analizar descripciones y adquirir almacenamiento

La siguiente figura muestra el SQLDA después de que su programa ejecuta una instrucción FETCH.

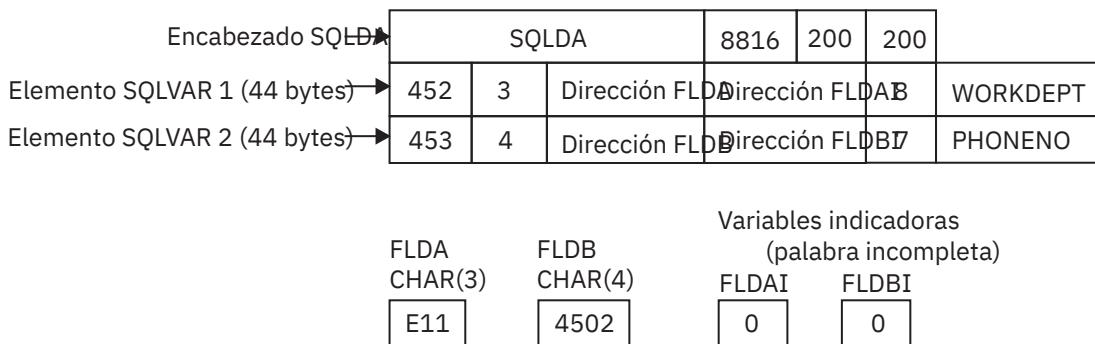


Figura 36. Área descriptora SQL después de ejecutar FETCH

La siguiente tabla describe los valores del área descriptora.

Tabla 87. Valores insertados en el SQLDA

Valor	Campo	Descripción
SQLDA	SQLDAID	"Un atractivo para la vista"
8816	SQLDABC	El tamaño del SQLDA en bytes (16 + 44 * 200)
200	SQLN	El número de apariciones de SQLVAR, establecido por el programa
200	SQLD	El número de apariciones de SQLVAR realmente utilizado por la sentencia DESCRIBE
452	SQLTYPE	El valor de SQLTYPE en la primera aparición de SQLVAR. Indica que la primera columna contiene cadenas de caracteres de longitud fija y no permite valores nulos.
3	SQLLEN	El atributo de longitud de la columna
Valor no definido o CCSID	SQLDATA	Los bytes 3 y 4 contienen el CCSID de una columna de cadena. No definido para otros tipos de columnas.
No definido	SQLIND	
8	SQLNAME	El número de caracteres en el nombre de la columna
WORKDEPT	SQLNAME+2	El nombre de la primera columna

Introducir direcciones de almacenamiento en el SQLDA

Después de analizar la descripción de cada columna, su programa debe reemplazar el contenido de cada campo SQLDATA con la dirección de un área de almacenamiento lo suficientemente grande

como para contener los valores de esa columna. De manera similar, para cada columna que permita valores nulos, el programa debe reemplazar el contenido del campo SQLIND. El contenido debe ser la dirección de una semicadena que pueda utilizar como variable indicadora para la columna. El programa puede adquirir almacenamiento para este fin, por supuesto, pero las áreas de almacenamiento utilizadas no tienen que ser contiguas.

[Figura 35 en la página 537](#) muestra el contenido del área del descriptor antes de que el programa obtenga cualquier fila de la tabla de resultados. Las direcciones de los campos y las variables indicadoras ya están en el SQLVAR.

Cambiar el CCSID para los datos recuperados

Todos los datos de cadena de caracteres (Db2) tienen un esquema de codificación y un CCSID asociados. Cuando selecciona datos de cadena de una tabla, los datos seleccionados generalmente tienen el mismo esquema de codificación y CCSID que la tabla. Si la aplicación utiliza algún método, como la emisión de la instrucción DECLARE VARIABLE, para cambiar el CCSID de los datos seleccionados, los datos se convierten del CCSID de la tabla al CCSID especificado por la aplicación.

Puede establecer el esquema de codificación de la aplicación predeterminado para un plan o paquete especificando el valor en el campo CODIFICACIÓN DE LA APLICACIÓN del panel VALORES PREDETERMINADOS PARA EL PAQUETE DE ENCUADERNACIÓN o VALORES PREDETERMINADOS PARA EL PLAN DE ENCUADERNACIÓN. El esquema de codificación de aplicación predeterminado para el subsistema de Db2 es el valor que se especificó en el campo APPLICATION ENCODING del panel de instalación DSNTIPF.

Si desea recuperar los datos en un esquema de codificación y CCSID distintos de los valores predeterminados, puede utilizar una de las siguientes técnicas:

- Para SQL dinámico, establezca el registro especial CURRENT APPLICATION ENCODING SCHEME antes de ejecutar las instrucciones SELECT. Por ejemplo, para establecer el CCSID y el esquema de codificación de los datos recuperados en el CCSID predeterminado para Unicode, ejecute esta instrucción SQL:

```
EXEC SQL SET CURRENT APPLICATION ENCODING SCHEME = 'UNICODE';
```

El valor inicial de este registro especial es el esquema de codificación de la aplicación que se determina mediante la opción BIND.

- Para instrucciones SQL estáticas y dinámicas que utilizan variables de host y matrices de variables de host, utilice la instrucción DECLARE VARIABLE para asociar CCSID con las variables de host en las que recupera los datos. Visite [“Establecimiento del CCSID para variables host” en la página 508](#) para obtener información sobre esta técnica.
- Para las sentencias SQL estáticas y dinámicas que utilizan un descriptor, establezca el CCSID para los datos recuperados en el SQLDA. El siguiente texto describe esa técnica.

Para cambiar el esquema de codificación de las sentencias SQL que utilizan un descriptor, configure el SQLDA y, a continuación, realice estos cambios adicionales en el SQLDA:

1. Ponga el carácter + en el sexto byte del campo SQLDAID.
2. Para cada entrada SQLVAR:
 - a. Establezca el campo de longitud de SQLNAME en 8.
 - b. Establezca los dos primeros bytes del campo de datos de SQLNAME en X'0000'.
 - c. Establezca el tercer y cuarto byte del campo de datos de SQLNAME en el CCSID, en hexadecimal, en el que desea que se muestren los resultados, o en X'0000'. X'0000' indica que Db2 debe utilizar el CCSID predeterminado. Si especifica un CCSID distinto de cero, debe cumplir una de las siguientes condiciones:
 - Una fila de la tabla de catálogo SYSSTRINGS tiene un valor coincidente para OUTCCSID.
 - Los servicios de conversión Unicode admiten la conversión a ese CCSID. Visite [Creación y uso de bibliotecas de enlace dinámico \(DLL\) \(XL C/C++ Programming Guide\)](#) para obtener información sobre las conversiones admitidas.

Si está modificando el CCSID para recuperar el contenido de una tabla ASCII, EBCDIC o Unicode en un sistema de base de datos relacional (Db2 for z/OS) y previamente ejecutó una instrucción DESCRIBE en la instrucción SELECT que está utilizando para recuperar los datos, los campos SQLDATA en el SQLDA que utilizó para el DESCRIBE contienen el ASCII o el Unicode CCSID para esa tabla. Para establecer la parte de datos de los campos SQLNAME para el SELECT, mueva el contenido de cada campo SQLDATA en el SQLDA desde el DESCRIBE a cada campo SQLNAME en el SQLDA para el SELECT. Si utiliza el mismo SQLDA para DESCRIBE y SELECT, asegúrese de mover el contenido del campo SQLDATA a SQLNAME antes de modificar el campo SQLDATA para SELECT.

Para REXX, se establece el CCSID en el campo SQLUSECCSID de la instrucción stem.n, en lugar de establecer los campos SQLDAID y SQLNAME.

Por ejemplo, supongamos que la tabla que contiene WORKDEPT y PHONENO se define con CCSID ASCII. Para recuperar datos de las columnas WORKDEPT y PHONENO en ASCII CCSID 437 ('X'01B5'), cambie el SQLDA como se muestra en la siguiente figura.

Encabezado SQLDA	SQLDA			8816	200	200
Elemento SQLVAR 1 (44 bytes) →	452	3	Dirección FLDA	Dirección FLDAB	X 000001B500000000	
Elemento SQLVAR 2 (44 bytes) →	453	4	Dirección FLDB	Dirección FLDBB	X 000001B500000000	
	FLDA CHAR(3)	FLDB CHAR(4)		Variables indicadoras (palabra incompleta)	FLDAI	FLDBI

Figura 37. Área descriptora SQL para recuperar datos en ASCII CCSID 437

Especificación que DESCRIBE utilice etiquetas de columna en el campo SQLNAME

Por defecto, DESCRIBE describe cada columna en el campo SQLNAME por el nombre de la columna. Puede indicarle que utilice etiquetas de columna en su lugar.

Restricción: No puede utilizar etiquetas de columna con operadores de conjuntos (UNION, INTERSECT y EXCEPT).

Para especificar que DESCRIBE utilice etiquetas de columna en el campo SQLNAME, especifique una de las siguientes opciones al emitir la instrucción DESCRIBE:

USO DE ETIQUETAS

Especifica que SQLNAME debe contener etiquetas. Si una columna no tiene etiqueta, SQLNAME no contiene nada.

USO DE CUALQUIER

Especifica que SQLNAME debe contener etiquetas dondequiera que existan. Si una columna no tiene etiqueta, SQLNAME contiene el nombre de la columna.

USO DE AMBOS

Especifica que SQLNAME debe contener tanto etiquetas como nombres de columna, cuando ambos existen.

En este caso, FULSQLDA debe contener un segundo conjunto de apariciones de SQLVAR. El primer conjunto contiene descripciones de todas las columnas con nombres de columna; el segundo conjunto contiene descripciones con etiquetas de columna.

Si elige esta opción, realice las siguientes acciones:

- Asigne un SQLDA más largo para la segunda sentencia DESCRIBE ((16 + SQLD * 88 bytes) en lugar de (16 + SQLD * 44))
- Ponga el doble de columnas (SQLD * 2) en el campo SQLN del segundo SQLDA.

Estas acciones garantizan que haya suficiente espacio disponible. De lo contrario, si no hay suficiente espacio disponible, DESCRIBE no introduce descripciones de ninguna de las columnas.

```
EXEC SQL  
DESCRIBE STMT INTO :FULSQLDA USING LABELS;
```

Algunas columnas, como las derivadas de funciones o expresiones, no tienen nombre ni etiqueta; SQLNAME no contiene nada para esas columnas. Por ejemplo, si utiliza UNION para combinar dos columnas que no tienen el mismo nombre y no utilizan una etiqueta, SQLNAME contiene una cadena de longitud cero.

Describir tablas con LOB y columnas de tipo distinto

En general, los pasos que se realizan al preparar un SQLDA para seleccionar filas de una tabla con LOB y columnas de tipo distinto son similares a los pasos que se realizan si la tabla no tiene columnas de este tipo. La única diferencia es que necesita analizar algunos campos adicionales en el SQLDA para columnas LOB o de tipo distinto.

Por ejemplo, supongamos que desea ejecutar esta instrucción SELECT:

```
SELECT USER, A_DOC FROM DOCUMENTS;
```

La columna USUARIO no puede contener valores nulos y es de tipo ID distinto, definido de la siguiente manera:

```
CREATE DISTINCT TYPE SCHEMA1.ID AS CHAR(20);
```

La columna A_DOC puede contener valores nulos y es de tipo CLOB (1M).

La tabla de resultados de esta sentencia tiene dos columnas, pero necesita cuatro apariciones de SQLVAR en su SQLDA porque la tabla de resultados contiene un tipo LOB y un tipo distinto. Supongamos que prepara y describe esta declaración en FULSQLDA, que es lo suficientemente grande como para contener cuatro ocurrencias de SQLVAR. FULSQLDA tiene el aspecto de la siguiente figura.

Encabezado SQLDA	SQLDA 2			192	4	4
Elemento SQLVAR 1 (44 bytes) →	452	20	Indefinido	0	4	USER
Elemento SQLVAR 2 (44 bytes) →	409	0	Indefinido	0	5	A_DOC
SQLVAR2 elemento 1 (44 bytes) →					7	SCH1.ID
SQLVAR2 elemento 2 (44 bytes) →	1.048.576				11	SYSIBM.CLOB

Figura 38. Área descriptora SQL después de describir un CLOB y un tipo distinto

Los siguientes pasos son los mismos que para las tablas de resultados sin LOB o tipos distintos:

1. Analice cada descripción SQLVAR para determinar la cantidad máxima de espacio que necesita para el valor de la columna.

Para un tipo LOB, recupere la longitud del campo SQLLONGL en lugar del campo SQLLEN.

2. Obtener la dirección de algún área de almacenamiento del tamaño requerido.

Para un tipo de datos LOB, también necesita un área de almacenamiento de 4 bytes para la longitud de los datos LOB. Puede asignar esta área de 4 bytes al principio de los datos LOB o en una ubicación diferente.

3. Ponga esta dirección en el campo SQLDATA.

Para un tipo de datos LOB, si ha asignado un área separada para contener la longitud de los datos LOB, ponga la dirección del campo de longitud en SQLDATA. Si el campo de longitud está al principio del área de datos LOB, ponga 0 en SQLDATA. Cuando se utiliza una variable de referencia de archivo para una columna LOB, la variable indicadora indica si los datos del archivo son nulos, no si los datos a los que apunta SQLDATA son nulos.

4. Si el campo SQLTYPE indica que el valor puede ser nulo, el programa también debe poner la dirección de una variable indicadora en el campo SQLIND.

La siguiente figura muestra el contenido de FULSQLDA después de introducir punteros a las ubicaciones de almacenamiento.

Encabezado SQLDA	SQLDA 2			192	4	4
Elemento SQLVAR 1 (44 bytes) →	452	20	Dirección FLDA0	4	USER	
Elemento SQLVAR 2 (44 bytes) →	409	0	Dirección FLDB0	A_DOC		
SQLVAR2 elemento 1 (44 bytes) →				7	SCH1.ID	
SQLVAR2 elemento 2 (44 bytes) →	1.048.576			11	SYSIBM.CLOB	
	FLDA CHAR(20)	FLDB CLOB (1M)	Variable indicadora (palabra incompleta) FLDBI			
			4 bytes de campo de longitud			

Figura 39. Área descriptora SQL después de analizar descripciones de tipo CLOB y distinto y adquirir almacenamiento

La siguiente figura muestra el contenido de FULSQLDA después de ejecutar una instrucción FETCH.

Encabezado SQLDA	SQLDA 2			192	4	4
Elemento SQLVAR 1 (44 bytes) →	452	20	Dirección FLDA0	4	USER	
Elemento SQLVAR 2 (44 bytes) →	409	0	Dirección FLDB0	A_DOC		
SQLVAR2 elemento 1 (44 bytes) →				7	SCH1.ID	
SQLVAR2 elemento 2 (44 bytes) →	1.048.576			11	SYSIBM.CLOB	
	FLDA CHAR(20)	FLDB CLOB (1M)	Variable indicadora (palabra incompleta) FLDBI			
			longitud de 4 bytes el campo contiene longitud real de datos			

Figura 40. Área descriptora SQL después de ejecutar FETCH en una tabla con columnas de tipo CLOB y distinto

Configuración de una variable de host XML en un SQLDA

En lugar de especificar variables de host para almacenar valores XML de una tabla, puede crear un SQLDA para que apunte a las áreas de datos donde Db2 coloca los datos recuperados. El SQLDA necesita describir el tipo de datos para cada área de datos.

Para establecer una variable de host XML en un SQLDA:

1. Asignar un SQLDA apropiado.
2. Emite una sentencia DESCRIBE para la sentencia SQL cuyo conjunto de resultados deseas almacenar. La sentencia DESCRIBE rellena el SQLDA basándose en las definiciones de las columnas. En el SQLDA, se rellena una entrada SQLVAR para cada columna del conjunto de resultados. (Se llenan varias entradas SQLVAR para columnas LOB y columnas con tipos

distintos) Para columnas de tipo XML, la entrada SQLVAR asociada se rellena de la siguiente manera:

Tabla 88. Valores de campo SQLVAR para columnas XML

Campo SQLVAR	Valor para una columna XML
sqltype SQLTYPE	988 para una columna que no es anulable o 989 para una columna anulable
sqlllen SQLLEN	0
sqldata SQLDATA	0
sqlind SQLIND	0
sqlname SQLNAME	El nombre o la etiqueta no cualificados de la columna

3. Compruebe el campo SQLTYPE de cada entrada SQLVAR. Si el campo SQLTYPE es 988 o 989, la columna en el conjunto de resultados es una columna XML.

4. Para cada columna XML, realice los siguientes cambios en la entrada SQLVAR asociada:

- Cambie el campo SQLTYPE para indicar el tipo de datos de la variable host para recibir los datos XML. Puede recuperar los datos XML en una variable host de tipo XML AS BLOB, XML AS CLOB o XML AS DBCLOB, o un tipo de datos de cadena compatible.

Si el tipo de variable de host de destino es XML AS BLOB, XML AS CLOB o XML AS DBCLOB, establezca el campo SQLTYPE en uno de los siguientes valores:

404

XML AS BLOB

405

xML anulable COMO BLOB

408

XML AS CLOB

409

xML anulable COMO CLOB

412

XML COMO DBCLOB

413

xML anulable COMO DBCLOB

Si el tipo de variable de host de destino es un tipo de datos de cadena, establezca el campo SQLTYPE en un valor de cadena válido.

Restricción: No puede utilizar el tipo XML (988/989) como tipo de variable de host de destino.

- Si el tipo de variable de host de destino es XML AS BLOB, XML AS CLOB o XML AS DBCLOB, cambie los dos primeros bytes del campo SQLNAME a 'X'0000' y el quinto y sexto bytes a 'X'0100'. Estos bytes indican que el valor que se va a recibir es un valor XML.

5. Rellene los campos SQLVAR extendidos para cada columna XML como lo haría para una columna LOB, como se indica en la siguiente tabla.

Tabla 89. Campos para una entrada SQLVAR ampliada para una variable de host XML

Campo SQLVAR	Valor para una variable de host XML
len.sqllonglen SQLLONGL SQLLONGLEN	atributo length para la variable host XML
*	Reservado
sqldatalen SQLDATAL sqldatalen	puntero a la longitud de la variable host XML
sqldatatype_name SQLNAME SQLDATATYPENAME	no utilizado

Ahora puede utilizar SQLDA para recuperar los datos XML en una variable host de tipo XML AS BLOB, XML AS CLOB o XML AS DBCLOB, o un tipo de datos de cadena compatible.

Ejecutar una sentencia SELECT de lista variable de forma dinámica

Puede recuperar fácilmente filas de la tabla de resultados utilizando una instrucción SELECT de lista variable. Las declaraciones difieren solo un poco de las del ejemplo de la lista fija.

1. Abre el cursor. Si la instrucción SELECT no contiene ningún marcador de parámetro, este paso es bastante sencillo. Por ejemplo:

```
EXEC SQL OPEN C1;
```

2. Recuperar filas de la tabla de resultados. Esta declaración difiere de la correspondiente en el caso de una selección de lista fija. Escribir:

```
EXEC SQL
  FETCH C1 USING DESCRIPTOR :FULSQLDA;
```

La característica clave de esta declaración es la cláusula USING DESCRIPTOR: FULSQLDA. Esta cláusula nombra un área descriptora SQL en la que las apariciones de SQLVAR apuntan a otras áreas. Esas otras áreas reciben los valores que devuelve FETCH. Es posible utilizar esa cláusula solo porque configuró previamente FULSQLDA para que se pareciera a [Figura 34 en la página 536](#).

[Figura 36 en la página 537](#) muestra el resultado de FETCH. Las áreas de datos identificadas en los campos SQLVAR reciben los valores de una sola fila de la tabla de resultados.

Las ejecuciones sucesivas de la misma sentencia FETCH ponen valores de filas sucesivas de la tabla de resultados en estas mismas áreas.

3. Cierre el cursor. Este paso es el mismo que para el caso de la lista fija. Cuando no haya más filas que procesar, ejecute la siguiente instrucción:

```
EXEC SQL CLOSE C1;
```

Cuando COMMIT finaliza la unidad de trabajo que contiene OPEN, la instrucción en STMT vuelve al estado no preparado. A menos que haya definido el cursor utilizando la opción CON RETENCIÓN, debe preparar el extracto de nuevo antes de poder reabrir el cursor.

Ejecución de sentencias arbitrarias con marcadores de parámetros

Considere, por ejemplo, un programa que ejecuta sentencias SQL dinámicas de varios tipos, incluidas sentencias SELECT de lista variable, cualquiera de las cuales podría contener un número variable de marcadores de parámetros. Este programa puede presentar a sus usuarios listas de opciones: opciones de operación (actualizar, seleccionar, eliminar); opciones de nombres de tablas; opciones de columnas para seleccionar o actualizar. El programa también permite a los usuarios introducir

listas de números de empleados para aplicarlas a la operación elegida. A partir de esto, el programa construye sentencias SQL de varias formas, una de las cuales tiene este aspecto:

```
SELECT .... FROM DSN8C10.EMP  
WHERE EMPNO IN (?,?,?,?,...?);
```

A continuación, el programa ejecuta estas instrucciones de forma dinámica.

Cuando se conocen el número y los tipos de parámetros

En el ejemplo anterior, no se conoce de antemano el número de marcadores de parámetros, y tal vez los tipos de parámetros que representan. Puede utilizar las técnicas descritas anteriormente si conoce el número y los tipos de parámetros, como en los siguientes ejemplos:

- Si la instrucción SQL **no es** SELECT, nombre una lista de variables de host en la instrucción EXECUTE:

WRONG:	EXEC SQL EXECUTE STMT;
RIGHT:	EXEC SQL EXECUTE STMT USING :VAR1, :VAR2, :VAR3;

- Si la instrucción SQL **es** SELECT, nombre una lista de variables de host en la instrucción OPEN:

WRONG:	EXEC SQL OPEN C1;
RIGHT:	EXEC SQL OPEN C1 USING :VAR1, :VAR2, :VAR3;

En **ambos** casos, el número y los tipos de variables de host nombradas deben coincidir con el número de marcadores de parámetros en STMT y los tipos de parámetros que representan. La primera variable (VAR1 en los ejemplos) debe tener el tipo esperado para el primer marcador de parámetro en la sentencia, la segunda variable debe tener el tipo esperado para el segundo marcador, y así sucesivamente. Debe haber al menos tantas variables como marcadores de parámetros.

Cuando se desconoce el número y los tipos de parámetros

Cuando no se conozca el número y los tipos de parámetros, se puede adaptar el área del descriptor SQL. Su programa puede incluir un número ilimitado de SQLDA, y puede utilizarlos para diferentes propósitos. Supongamos que un SQLDA, llamado arbitrariamente DPARM, describe un conjunto de parámetros.

La estructura de DPARM es la misma que la de cualquier otro SQLDA. El número de apariciones de SQLVAR puede variar, como en los ejemplos anteriores. En este caso, cada marcador de parámetro debe tener un SQLVAR. Cada aparición de SQLVAR describe una variable de host que reemplaza un marcador de parámetro en tiempo de ejecución. Db2 sustituye a los marcadores de parámetros cuando se ejecuta una sentencia no SELECT o cuando se abre un cursor para una sentencia SELECT.

Debe introducir ciertos campos en DPARM **antes de** usar EXECUTE u OPEN; puede ignorar los demás campos.

Campo

Utilícelo al describir variables de host para marcadores de parámetros

SQLDAID

El séptimo byte indica si se utiliza más de una entrada SQLVAR para cada marcador de parámetro. Si este byte no está en blanco, al menos un marcador de parámetro representa un tipo distinto o un valor LOB, por lo que el SQLDA tiene más de un conjunto de entradas SQLVAR.

No se establece este campo para un REXX SQLDA.

SQLDABC

La longitud del SQLDA, que es igual a SQLN * 44 + 16. No se establece este campo para un REXX SQLDA.

SQLN

El número de ocurrencias de SQLVAR asignado para DPARM. No se establece este campo para un REXX SQLDA.

SQLD

El número de veces que se ha utilizado SQLVAR. Este número no debe ser inferior al número de marcadores de parámetros. En cada aparición de SQLVAR, introduzca información en los siguientes campos: SQLTYPE, SQLLEN, SQLDATA, SQLIND.

SQLTYPE

El código para el tipo de variable y si permite valores nulos.

SQLLEN

La longitud de la variable de host.

SQLDATA

La dirección de la variable de host.

Para REXX, este campo contiene el valor de la variable host.

SQLIND

La dirección de una variable indicadora, si es necesario.

Para REXX, este campo contiene un número negativo si el valor en SQLDATA es nulo.

SQLNAME

Ignorar.

Uso de SQLDA con EXECUTE u OPEN

Para indicar que el SQLDA llamado DPARM describe las variables del host sustituidas por los marcadores de parámetros en tiempo de ejecución, utilice una cláusula USING DESCRIPTOR con EXECUTE u OPEN.

- Para una declaración no SELECT, escriba:

```
EXEC SQL EXECUTE STMT USING DESCRIPTOR :DPARM;
```

- Para una sentencia SELECT, escriba:

```
EXEC SQL OPEN C1 USING DESCRIPTOR :DPARM;
```

Cómo afectan las opciones de enlace REOPT(ALWAYS), REOPT(AUTO) y REOPT(ONCE) al SQL dinámico

Cuando especifica la opción de enlace REOPT(ALWAYS), Db2 , reoptimiza la ruta de acceso en tiempo de ejecución para las sentencias SQL que contienen variables de host, marcadores de parámetros o registros especiales. La opción REOPT(ALWAYS) tiene los siguientes efectos en las sentencias SQL dinámicas:

- Cuando especifica la opción REOPT(ALWAYS), Db2 utiliza automáticamente DEFER(PREPARE), lo que significa que Db2 espera para preparar una sentencia hasta que encuentra una sentencia OPEN o EXECUTE.
- Cuando ejecutas una sentencia DESCRIBE y luego una sentencia EXECUTE en una sentencia que no es SELECT, el procesador de sentencias (Db2) prepara la sentencia dos veces: una para la sentencia DESCRIBE y otra para la sentencia EXECUTE. Db2 utiliza los valores de las variables de entrada solo durante el segundo PREPARE. Estos múltiples PREPAREs pueden hacer que el rendimiento se degrade si su programa contiene muchas sentencias dinámicas no SELECT. Para mejorar el rendimiento, considere la posibilidad de poner el código que contiene esas sentencias en un paquete separado y luego vincular ese paquete con la opción REOPT(NONE).
- Si ejecuta una instrucción DESCRIBE antes de abrir un cursor para esa instrucción, el procesador de instrucciones (Db2) prepara la instrucción dos veces. Sin embargo, si ejecuta una instrucción DESCRIBE después de abrir el cursor, Db2 prepara la instrucción solo una vez. Para mejorar el rendimiento de un programa vinculado con la opción REOPT(ALWAYS), ejecute la sentencia

DESCRIBE **después de** abrir el cursor. Para evitar que se abra automáticamente un DESCRIBE antes de un cursor, no utilice una sentencia PREPARE con la cláusula INTO.

- Si utiliza el gobierno predictivo para aplicaciones vinculadas con REOPT(ALWAYS), Db2 no devuelve un SQLCODE de advertencia cuando las sentencias SQL dinámicas superan el umbral de advertencia de gobierno predictivo. Db2 devuelve un error SQLCODE cuando las sentencias SQL dinámicas superan el umbral de error regulador predictivo. Db2 devuelve el error SQLCODE para una instrucción EXECUTE u OPEN.

Cuando especifica la opción de enlace REOPT(AUTO), Db2 optimiza la ruta de acceso para las instrucciones SQL en el primer EXECUTE u OPEN. Db2 , determina cada vez que se ejecuta una instrucción si se necesita una nueva ruta de acceso para mejorar el rendimiento de la instrucción. Si una nueva ruta de acceso mejora el rendimiento, Db2 genera una. La opción REOPT(AUTO) tiene los siguientes efectos en las sentencias SQL dinámicas:

- Cuando especifica la opción de enlace REOPT(AUTO), Db2 optimiza la ruta de acceso para las instrucciones SQL en el primer EXECUTE u OPEN. Db2 , determina cada vez que se ejecuta una instrucción si se necesita una nueva ruta de acceso para mejorar el rendimiento de la instrucción. Si una nueva ruta de acceso mejora el rendimiento, Db2 genera una.
- Cuando especifica la opción REOPT(ONCE), Db2 utiliza automáticamente DEFER(PREPARE), lo que significa que Db2 espera para preparar una sentencia hasta que encuentra una sentencia OPEN o EXECUTE.
- Cuando Db2 prepara una sentencia utilizando REOPT(AUTO), guarda la ruta de acceso en la caché de sentencias dinámicas. Esta ruta de acceso se utiliza cada vez que se ejecuta la instrucción, hasta que Db2 determina que se necesita una nueva ruta de acceso para mejorar el rendimiento o la instrucción que está en la caché se invalida (o se elimina de la caché) y necesita ser reasignada.
- La sentencia DESCRIBE tiene los siguientes efectos en las sentencias dinámicas que están vinculadas con REOPT(AUTO):
 - Cuando ejecutas una sentencia DESCRIBE antes de una sentencia EXECUTE en una sentencia que no es SELECT, Db2 prepara la sentencia una vez más si aún no está guardada en la caché: Una vez para la sentencia DESCRIBE y otra para la sentencia EXECUTE. Db2 utiliza los valores de las variables de entrada solo durante la segunda vez que se prepara la instrucción. A continuación, guarda el extracto en la caché. Si ejecuta una sentencia DESCRIBE antes de una sentencia EXECUTE en una sentencia no SELECT que ya se ha guardado en la caché, Db2 siempre preparará la sentencia no SELECT para la sentencia DESCRIBE, y preparará la sentencia de nuevo en EXECUTE solo si Db2 determina que una nueva ruta de acceso diferente a la ya guardada en la caché puede mejorar el rendimiento.
 - Si ejecuta DESCRIBE en un extracto antes de abrir un cursor para ese extracto, Db2 siempre prepara el extracto en DESCRIBE. Sin embargo, Db2 no volverá a preparar el extracto en OPEN si el extracto ya se ha guardado en la caché y Db2 no cree que sea necesario un nuevo acceso en el momento de OPEN. Si ejecuta DESCRIBE en un extracto después de abrir un cursor para ese extracto, Db2 preparó el extracto solo una vez si aún no está guardado en la caché. Si la sentencia ya está guardada en la caché y ejecutas DESCRIBE después de abrir un cursor para esa sentencia, Db2 no prepara la sentencia, sino que utiliza la sentencia que está guardada en la caché.
- Si utiliza el gobierno predictivo para aplicaciones vinculadas con REOPT(AUTO), Db2 no devuelve un SQLCODE de advertencia cuando las sentencias SQL dinámicas superan el umbral de advertencia de gobierno predictivo. Db2 devuelve un error SQLCODE cuando las sentencias SQL dinámicas superan el umbral de error regulador predictivo. Db2 devuelve el error SQLCODE para una instrucción EXECUTE u OPEN.

Cuando especifica la opción de enlace REOPT(ONCE), Db2 optimiza la ruta de acceso solo una vez, en el primer EXECUTE u OPEN, para las sentencias SQL que contienen variables de host, marcadores de parámetros o registros especiales. La opción REOPT(ONCE) tiene los siguientes efectos en las sentencias SQL dinámicas:

- Cuando especifica la opción REOPT(ONCE), Db2 utiliza automáticamente DEFER(PREPARE), lo que significa que Db2 espera para preparar una sentencia hasta que encuentra una sentencia OPEN o EXECUTE.
- Cuando Db2 prepara una sentencia utilizando REOPT(ONCE), guarda la ruta de acceso en la caché de sentencias dinámicas. Esta ruta de acceso se utiliza cada vez que se ejecuta la instrucción, hasta que la instrucción que está en la caché se invalida (o se elimina de la caché) y es necesario volver a enlazarla.
- La sentencia DESCRIBE tiene los siguientes efectos en las sentencias dinámicas que están vinculadas con REOPT(ONCE):
 - Cuando ejecutas una sentencia DESCRIBE antes de una sentencia EXECUTE en una sentencia que no es SELECT, el procesador de consultas de Oracle (Db2) prepara la sentencia dos veces si aún no está guardada en la caché: una vez para la sentencia DESCRIBE y otra para la sentencia EXECUTE. Db2 utiliza los valores de las variables de entrada solo durante la segunda vez que se prepara la instrucción. A continuación, guarda el extracto en la caché. Si ejecuta una sentencia DESCRIBE antes de una sentencia EXECUTE en una sentencia no SELECT que ya se ha guardado en la caché, Db2 prepara la sentencia no SELECT solo para la sentencia DESCRIBE.
 - Si ejecuta DESCRIBE en un extracto **antes de** abrir un cursor para ese extracto, Db2 siempre prepara el extracto en DESCRIBE. Sin embargo, Db2 no volverá a preparar el extracto en ABIERTO si el extracto ya se ha guardado en la caché. Si ejecuta DESCRIBE en un extracto **después de** abrir un cursor para ese extracto, Db2 preparó el extracto solo una vez si aún no está guardado en la memoria caché. Si la sentencia ya está guardada en la caché y ejecutas DESCRIBE después de abrir un cursor para esa sentencia, Db2 no prepara la sentencia, sino que utiliza la sentencia que está guardada en la caché.

Para mejorar el rendimiento de un programa que está vinculado con REOPT(ONCE), ejecute la sentencia DESCRIBE después de abrir un cursor. Para evitar que se abra automáticamente un DESCRIBE antes de un cursor, no utilice una sentencia PREPARE con la cláusula INTO.

- Si utiliza el gobierno predictivo para aplicaciones vinculadas con REOPT(ONCE), Db2 no devuelve un SQLCODE de advertencia cuando las sentencias SQL dinámicas superan el umbral de advertencia de gobierno predictivo. Db2 devuelve un error SQLCODE cuando las sentencias SQL dinámicas superan el umbral de error regulador predictivo. Db2 devuelve el error SQLCODE para una instrucción EXECUTE u OPEN.

Conceptos relacionados

Aplicaciones ensambladoras que emiten sentencias SQL

Puede codificar sentencias SQL en programas ensambladores siempre que pueda utilizar sentencias ejecutables.

Aplicaciones C y C++ que emiten sentencias SQL

Puede codificar sentencias SQL en un programa C o C++ siempre que pueda utilizar sentencias ejecutables.

Aplicaciones COBOL que emiten sentencias SQL

Puede codificar sentencias SQL en determinadas secciones del programa COBOL.

Aplicaciones Fortran que emiten sentencias SQL

Puede codificar sentencias SQL en un programa Fortran siempre que pueda colocar sentencias ejecutables. Si la sentencia SQL está dentro de una sentencia IF, el precompilador genera cualquier sentencia THEN y END IF necesaria.

Aplicaciones PL/I que emiten sentencias SQL

Puede codificar sentencias SQL en un programa PL/I siempre que pueda utilizar sentencias ejecutables.

Aplicaciones REXX que emiten sentencias SQL

Puede codificar instrucciones SQL en un programa REXX siempre que pueda utilizar comandos REXX.

Referencia relacionada

DESCRIBE OUTPUT declaración (Db2 SQL)

Área de descriptor de SQL (SQLDA) (Db2 SQL)

Campos SQLTYPE y SQLLEN del SQLDA (Db2 SQL)

Ejecución dinámica de una sentencia SQL utilizando EXECUTE IMMEDIATE

En determinadas situaciones, es posible que desee que el programa prepare y ejecute dinámicamente una sentencia inmediatamente después de leerla.

Acerca de esta tarea

Supongamos que diseña un programa para leer instrucciones SQL DELETE, similares a estas, desde un terminal:

```
DELETE FROM DSN8C10.EMP WHERE EMPNO = '000190'  
DELETE FROM DSN8C10.EMP WHERE EMPNO = '000220'
```

Después de leer una declaración, el programa debe ejecutarla inmediatamente.

Recuerde que debe preparar (precompilar y vincular) sentencias SQL estáticas antes de poder utilizarlas. No se pueden preparar instrucciones SQL dinámicas por adelantado. La instrucción SQL EXECUTE IMMEDIATE hace que una instrucción SQL se prepare y execute, dinámicamente, en tiempo de ejecución.

Antes de preparar y ejecutar una instrucción SQL, puede leerla en una variable de host. Si la longitud máxima de la instrucción SQL es de 32 KB, declare la variable de host como una variable de host de caracteres o gráfica de acuerdo con las siguientes reglas para los lenguajes de host:

- En ensamblador, PL/I, COBOL y C, debe declarar una variable de host de cadena como una cadena de longitud variable.
- En Fortran, debe ser una variable de cadena de longitud fija.

Si la longitud es superior a 32 KB, debe declarar la variable host como CLOB o DBCLOB, y el máximo es de 2 MB.

ejemplos

Ejemplo: Uso de una variable de host de caracteres de longitud variable

Este fragmento es de un programa en C que lee una instrucción DELETE en la variable de host *dstring* y ejecuta la instrucción:

```
EXEC SQL BEGIN DECLARE SECTION;  
...  
struct VARCHAR {  
    short len;  
    char s[40];  
} dstring;  
EXEC SQL END DECLARE SECTION;  
...  
/* Read a DELETE statement into the host variable dstring. */  
gets(dstring);  
EXEC SQL EXECUTE IMMEDIATE :dstring;  
...
```

EXECUTE IMMEDIATE hace que la instrucción DELETE se prepare y execute inmediatamente.

Declaración de una variable de host CLOB o DBCLOB

Usted declara las variables de host CLOB y DBCLOB de acuerdo con ciertas reglas.

El precompilador genera una estructura que contiene dos elementos, un campo de longitud de 4 bytes y un campo de datos de la longitud especificada. Los nombres de estos campos varían en función del idioma del host:

- En PL/I, ensamblador y C Fortran, los nombres son *variable_LENGTH* y *variable_DATA*.
- En COBOL, los nombres son *variable-LENGTH* y *variable-DATA*.
- En C, los nombres son *variable.LENGTH* y *variable.DATOS*.

Ejemplo: Uso de una variable de host CLOB

Este fragmento es de un programa en C que copia una instrucción UPDATE en la variable host *string1* y ejecuta la instrucción:

```
EXEC SQL BEGIN DECLARE SECTION;
...
SQL TYPE IS CLOB(4k) string1;
EXEC SQL END DECLARE SECTION;
...
/* Copy a statement into the host variable string1. */
strcpy(string1.data, "UPDATE DSN8610.EMP SET SALARY = SALARY * 1.1");
string1.length = 44;
EXEC SQL EXECUTE IMMEDIATE :string1;
...
```

EXECUTE IMMEDIATE hace que la instrucción UPDATE se prepare y execute inmediatamente.

Conceptos relacionados

[Declaraciones de variable de referencia de archivo LOB, variable host LOB y localizador LOB](#)

Cuando escribe aplicaciones para manipular datos LOB, necesita declarar variables hosto para contener los datos LOB o el localizador LOB. De lo contrario, debe declarar variables de referencia de archivo LOB para apuntar a datos LOB.

[Aplicaciones ensambladoras que emiten sentencias SQL](#)

Puede codificar sentencias SQL en programas ensambladores siempre que pueda utilizar sentencias ejecutables.

[Aplicaciones C y C++ que emiten sentencias SQL](#)

Puede codificar sentencias SQL en un programa C o C++ siempre que pueda utilizar sentencias ejecutables.

[Aplicaciones COBOL que emiten sentencias SQL](#)

Puede codificar sentencias SQL en determinadas secciones del programa COBOL.

[Aplicaciones Fortran que emiten sentencias SQL](#)

Puede codificar sentencias SQL en un programa Fortran siempre que pueda colocar sentencias ejecutables. Si la sentencia SQL está dentro de una sentencia IF, el precompilador genera cualquier sentencia THEN y END IF necesaria.

[Aplicaciones PL/I que emiten sentencias SQL](#)

Puede codificar sentencias SQL en un programa PL/I siempre que pueda utilizar sentencias ejecutables.

[Aplicaciones REXX que emiten sentencias SQL](#)

Puede codificar instrucciones SQL en un programa REXX siempre que pueda utilizar comandos REXX.

Ejecución dinámica de una sentencia SQL utilizando PREPARE y EXECUTE

Como alternativa a la ejecución de una sentencia SQL inmediatamente después de que se lee, puede preparar y ejecutar una sentencia SQL en dos pasos. Este método de dos pasos es útil cuando necesita ejecutar una sentencia SQL varias veces con diferentes valores.

Acerca de esta tarea

Supongamos que desea ejecutar instrucciones DELETE repetidamente utilizando una lista de números de empleados. Piense en cómo lo haría si pudiera escribir la instrucción DELETE como una instrucción SQL estática:

```
< Read a value for EMP from the list. >
DO UNTIL (EMP = 0);
  EXEC SQL
    DELETE FROM DSN8C10.EMP WHERE EMPNO = :EMP ;
  < Read a value for EMP from the list. >
END;
```

El bucle se repite hasta que se lee un valor EMP de 0.

Si sabe de antemano que solo utilizará la instrucción DELETE y solo la tabla DSN8C10.EMP, puede utilizar el SQL estático, que es más eficiente. Supongamos además que varias tablas diferentes tienen filas que se identifican por números de empleado, y que los usuarios introducen un nombre de tabla, así como una lista de números de empleado para eliminar. Aunque las variables pueden representar los números de empleado, no pueden representar el nombre de la tabla, por lo que debe construir y ejecutar la sentencia completa de forma dinámica.

Procedimiento

Para construir y ejecutar sentencias de forma dinámica, su programa debe hacer ahora estas cosas de manera diferente:

- Utilice marcadores de parámetros en lugar de variables de host.

Las sentencias SQL dinámicas no pueden utilizar variables de host. Por lo tanto, no puede ejecutar dinámicamente una instrucción SQL que contenga variables de host. En su lugar, sustituya *un marcador de parámetro*, indicado por un signo de interrogación (?), por cada variable de host en la instrucción.

Puede indicar a Db2 que un marcador de parámetro representa una variable de host de un determinado tipo de datos especificando el marcador de parámetro como argumento de una especificación CAST. Cuando se ejecuta la instrucción, Db2 convierte la variable host al tipo de datos en la especificación CAST. Un marcador de parámetro que se incluye en una especificación CAST se denomina marcador de parámetro *tipado*. Un marcador de parámetro sin una especificación CAST se denomina marcador de parámetro *sin tipo*.

Recomendación: Db2 , dado que puede evaluar una instrucción SQL con marcadores de parámetros tipificados de manera más eficiente que una instrucción con marcadores de parámetros no tipificados, utiliza marcadores de parámetros tipificados siempre que sea posible. En determinadas circunstancias, debe utilizar marcadores de parámetros mecanografiados.

Por ejemplo, supongamos que desea preparar esta declaración:

```
DELETE FROM DSN8C10.EMP WHERE EMPNO = :EMP;
```

Debe preparar una cadena como esta:

```
DELETE FROM DSN8C10.EMP WHERE EMPNO = CAST(? AS CHAR(6))
```

Asocia la variable de host :EMP con el marcador de parámetro cuando ejecutas la sentencia preparada. Supongamos que S1 es la declaración preparada. Entonces, la instrucción EXECUTE tiene este aspecto:

```
EXECUTE S1 USING :EMP;
```

- Utilice la instrucción PREPARE.

Antes de preparar una instrucción SQL, puede asignarla a una variable de host. Si la longitud de la declaración es superior a 32 KB, debe declarar la variable host como CLOB o DBCLOB.

Puede pensar en PREPARAR y EJECUTAR como un EJECUTAR INMEDIATAMENTE realizado en dos pasos. El primer paso, PREPARAR, convierte una cadena de caracteres en una instrucción SQL y, a continuación, le asigna un nombre de su elección.

Por ejemplo, supongamos que la variable de host de carácter :DSTRING tiene el valor «DELETE FROM DSN8C10 ».EMP WHERE EMPNO =?». Para preparar una instrucción SQL a partir de esa cadena y asignarle el nombre S1, escriba:

```
EXEC SQL PREPARE S1 FROM :DSTRING;
```

La sentencia preparada sigue conteniendo un marcador de parámetro, para el que debe proporcionar un valor cuando se ejecute la sentencia. Después de preparar la instrucción, el nombre de la tabla queda fijado, pero el marcador de parámetro le permite ejecutar la misma instrucción muchas veces con diferentes valores del número de empleado.

- Utilice EXECUTE en lugar de EXECUTE IMMEDIATE.

La instrucción EXECUTE ejecuta una instrucción SQL preparada nombrando una lista de una o más variables de host, una o más matrices de variables de host o una estructura de host. Esta lista proporciona valores para todos los marcadores de parámetros.

Después de preparar un estado de cuenta, puede ejecutarlo muchas veces dentro de la misma unidad de trabajo. En la mayoría de los casos, COMMIT o ROLLBACK destruyen las sentencias preparadas en una unidad de trabajo. Entonces, debe prepararlos de nuevo antes de poder ejecutarlos de nuevo. Sin embargo, si declara un cursor para una sentencia dinámica y utiliza la opción WITH HOLD, una operación de confirmación no destruye la sentencia preparada si el cursor sigue abierto. Puede ejecutar la instrucción en la siguiente unidad de trabajo sin tener que volver a prepararla.

Por ejemplo, para ejecutar la sentencia preparada S1 una sola vez, utilizando un valor de parámetro contenido en la variable de host :EMP, escriba:

```
EXEC SQL EXECUTE S1 USING :EMP;
```

ejemplos

Preparación y ejecución del ejemplo de instrucción DELETE

```
< Read a value for EMP from the list. >
DO UNTIL (EMP = 0);
  EXEC SQL
    DELETE FROM DSN8C10.EMP WHERE EMPNO = :EMP ;
  < Read a value for EMP from the list. >
END;
```

Ahora puede escribir un ejemplo equivalente para una sentencia SQL dinámica:

```
< Read a statement containing parameter markers into DSTRING.>
EXEC SQL PREPARE S1 FROM :DSTRING;
< Read a value for EMP from the list. >
DO UNTIL (EMPNO = 0);
  EXEC SQL EXECUTE S1 USING :EMP;
  < Read a value for EMP from the list. >
END;
```

La sentencia PREPARE prepara la sentencia SQL y la llama S1. La sentencia EXECUTE ejecuta un S1 a repetidamente, utilizando diferentes valores para EMP.

Uso de más de un marcador de parámetros

La declaración preparada (S1 en el ejemplo) puede contener más de un marcador de parámetro. Si lo hace, la cláusula USING de EXECUTE especifica una lista de variables o una estructura de host. Las variables deben contener valores que coincidan con el número y los tipos de datos de los parámetros en S1 en el orden correcto. Debe conocer el número y los tipos de parámetros de antemano y declarar las variables en su programa, o puede utilizar un SQLDA (área de descriptor SQL).

Conceptos relacionados

[Aplicaciones ensambladoras que emiten sentencias SQL](#)

Puede codificar sentencias SQL en programas ensambladores siempre que pueda utilizar sentencias ejecutables.

[Aplicaciones C y C++ que emiten sentencias SQL](#)

Puede codificar sentencias SQL en un programa C o C++ siempre que pueda utilizar sentencias ejecutables.

[Aplicaciones COBOL que emiten sentencias SQL](#)

Puede codificar sentencias SQL en determinadas secciones del programa COBOL.

[Aplicaciones Fortran que emiten sentencias SQL](#)

Puede codificar sentencias SQL en un programa Fortran siempre que pueda colocar sentencias ejecutables. Si la sentencia SQL está dentro de una sentencia IF, el precompilador genera cualquier sentencia THEN y END IF necesaria.

[Aplicaciones PL/I que emiten sentencias SQL](#)

Puede codificar sentencias SQL en un programa PL/I siempre que pueda utilizar sentencias ejecutables.

Aplicaciones REXX que emiten sentencias SQL

Puede codificar instrucciones SQL en un programa REXX siempre que pueda utilizar comandos REXX.

Tareas relacionadas

Ejecución dinámica de una sentencia SQL utilizando EXECUTE IMMEDIATE

En determinadas situaciones, es posible que desee que el programa prepare y ejecute dinámicamente una sentencia inmediatamente después de leerla.

Referencia relacionada

PREPARE declaración (Db2 SQL)

Ejecución dinámica de una sentencia de cambio de datos

La ejecución dinámica de sentencias de cambio de datos con matrices de variables de host es útil si desea especificar filas de datos en tablas diferentes. También resulta de utilidad si desea introducir un número diferente de filas. El proceso es similar para ambas sentencias INSERT y MERGE.

Acerca de esta tarea

Por ejemplo, supongamos que desea ejecutar repetidamente una instrucción INSERT de varias filas con una lista de ID de actividad, palabras clave de actividad y descripciones de actividad proporcionadas por el usuario. Puede utilizar la siguiente instrucción SQL INSERT estática para insertar varias filas de datos en la tabla de actividades:

```
EXEC SQL
  INSERT INTO DSN8C10.ACT
    VALUES (:hva_actno, :hva_actkwd, :hva_actdesc)
  FOR :num_rows ROWS;
```

Sin embargo, si desea introducir las filas de datos en tablas diferentes o introducir un número diferente de filas, puede construir la instrucción INSERT de forma dinámica.

Procedimiento

Para ejecutar una declaración de cambio de datos de forma dinámica, utilice uno de los siguientes métodos:

- Utilice matrices de variables de host que contengan los datos que se van a insertar, completando las siguientes acciones en su programa:

a) Asigne la instrucción INSERT o MERGE adecuada a una variable de host. Si es necesario, utilice la especificación CAST para asignar explícitamente tipos a marcadores de parámetros que representen matrices de variables de host.

Para la tabla de actividades, la siguiente cadena contiene una instrucción INSERT que debe prepararse:

```
INSERT INTO DSN8C10.ACT
  VALUES (CAST(?) AS SMALLINT), CAST(?) AS CHAR(6)), CAST(?) AS VARCHAR(20)))
```

b) Asigne cualquier atributo para la instrucción SQL a una variable de host.

c) Incluya una instrucción PREPARE para la instrucción SQL.

d) Incluya una instrucción EXECUTE con la cláusula *FOR n ROWS*.

Cada variable de host en la cláusula USING de la sentencia EXECUTE representa una matriz de valores para la columna correspondiente del objetivo de la sentencia SQL. Puede variar el número de filas sin necesidad de preparar de nuevo la instrucción SQL.

Por ejemplo, el siguiente código prepara y ejecuta una instrucción INSERT:

```
/* Copy the INSERT string into the host variable sqlstmt */
strcpy(sqlstmt, "INSERT INTO DSN8C10.ACT VALUES (CAST(?) AS SMALLINT),");
strcat(sqlstmt, " CAST(?) AS CHAR(6)), CAST(?) AS VARCHAR(20)))";
```

```

/* Copy the INSERT attributes into the host variable attrvar */
strcpy(attrvar, "FOR MULTIPLE ROWS");

/* Prepare and execute my_insert using the host-variable arrays */
EXEC SQL PREPARE my_insert ATTRIBUTES :attrvar FROM :sqlstmt;
EXEC SQL EXECUTE my_insert USING :hva1, :hva2, :hva3 FOR :num_rows ROWS;

```

- Utilice el descriptor para describir las matrices de variables de host que contienen los datos, completando las siguientes acciones en su programa:
 - Establezca los siguientes campos en la estructura SQLDA para especificar los tipos de datos y otra información sobre las matrices de variables de host que contienen los valores que se van a insertar en la instrucción INSERT.
 - SQLN
 - SQLABC
 - SQLD
 - SQLVAR
 - SQLNAME

Asuma que su programa incluye la declaración de estructura SQLDA estándar y las declaraciones para las variables del programa que apuntan a la estructura SQLDA. Para los programas de aplicación C, el siguiente código de ejemplo establece los campos SQLDA:

```

strcpy(sqladaptr->sqlaid,"SQLDA");
sqladaptr->sqldbc = 192; /* number of bytes of storage allocated
for the SQLDA */
sqladaptr->sqln = 4; /* number of SQLVAR
occurrences */
sqladaptr->sqld = 4;
varptr = (struct sqlvar *) (&(sqladaptr->sqlvar[0])); /* Point
to first SQLVAR */
varptr->sqltype = 500; /* data
type SMALLINT */
varptr->sqllen = 2;
varptr->sqldata = (char *) hva1;
varptr->sqlname.length = 8;
memcpy(varptr->sqlname.data, "\x00\x00\x00\x00\x00\x01\x00\x14", varptr->sqlname.length);
varptr = (struct sqlvar *) (&(sqladaptr->sqlvar[0]) + 1); /* Point
to next SQLVAR */
varptr->sqltype = 452; /* data
type CHAR(6) */
varptr->sqllen = 6;
varptr->sqldata = (char *) hva2;
varptr->sqlname.length = 8;
memcpy(varptr->sqlname.data, "\x00\x00\x00\x00\x00\x01\x00\x14", varptr->sqlname.length);
varptr = (struct sqlvar *) (&(sqladaptr->sqlvar[0]) + 2); /* Point
to next SQLVAR */
varptr->sqltype = 448; /* data type
VARCHAR(20) */
varptr->sqllen = 20;
varptr->sqldata = (char *) hva3;
varptr->sqlname.length = 8;
memcpy(varptr->sqlname.data, "\x00\x00\x00\x00\x00\x01\x00\x14", varptr->sqlname.length);

```

La estructura SQLDA tiene los siguientes campos:

- SQLDABC indica el número de bytes de almacenamiento que se asignan para el SQLDA. El almacenamiento incluye un encabezado de 16 bytes y 44 bytes para cada campo SQLVAR. El valor es SQLN x 44 + 16, o 192 para este ejemplo.
- SQLN es el número de apariciones de SQLVAR, más uno para uso de Db2 para la variable de host que contiene el número *n* en la cláusula *FOR n ROWS*.
- SQLD es el número de variables en el SQLDA que son utilizadas por el Db2 cuando procesa la instrucción INSERT.
- Una ocurrencia SQLVAR especifica los atributos de un elemento de una matriz de variables de host que corresponde a un valor proporcionado para una columna de destino del INSERT. Dentro de cada SQLVAR:
 - SQLTYPE indica el tipo de datos de los elementos de la matriz de variables de host.

- SQLLEN indica la longitud de un único elemento de la matriz de variables de host.
 - SQLDATA apunta a la matriz de variables de host correspondiente. Suponga que su programa asigna las matrices de variables dinámicas hva1, hva2 y hva3.
 - SQLNAME tiene dos partes: LENGTH y DATA. La LONGITUD es de 8. Los dos primeros bytes del campo DATA son X'0000'. Los bytes 5 y 6 del campo DATA son un indicador que indica si la variable es una matriz o un valor FOR n ROWS. Los bytes 7 y 8 son una representación entera binaria de dos bytes de la dimensión de la matriz.
- b) Asigne la instrucción INSERT o MERGE adecuada a una variable de host.
Por ejemplo, la siguiente cadena contiene una instrucción INSERT que debe prepararse:

```
INSERT INTO DSN8C10.ACT VALUES (?, ?, ?)
```

- c) Asigne cualquier atributo para la instrucción SQL a una variable de host.
- d) Incluya una instrucción PREPARE para la instrucción SQL.
- e) Incluya una instrucción EXECUTE con la cláusula *FOR n ROWS*. La variable host en la cláusula USING de la sentencia EXECUTE nombra el SQLDA que describe los marcadores de parámetros en la sentencia INSERT.

Por ejemplo, el siguiente código prepara y ejecuta una instrucción INSERT:

```
/* Copy the INSERT string into the host variable sqlstmt */
strcpy(sqlstmt, "INSERT INTO DSN8C10.ACT VALUES (?, ?, ?)");

/* Copy the INSERT attributes into the host variable attrvar */
strcpy(attrvar, "FOR MULTIPLE ROWS");

/* Prepare and execute my_insert using the descriptor */
EXEC SQL PREPARE my_insert ATTRIBUTES :attrvar FROM :sqlstmt;
EXEC SQL EXECUTE my_insert USING DESCRIPTOR :*sqlaptr FOR :num_rows ROWS;
```

Conceptos relacionados

Aplicaciones ensambladoras que emiten sentencias SQL

Puede codificar sentencias SQL en programas ensambladores siempre que pueda utilizar sentencias ejecutables.

Aplicaciones C y C++ que emiten sentencias SQL

Puede codificar sentencias SQL en un programa C o C++ siempre que pueda utilizar sentencias ejecutables.

Aplicaciones COBOL que emiten sentencias SQL

Puede codificar sentencias SQL en determinadas secciones del programa COBOL.

Aplicaciones Fortran que emiten sentencias SQL

Puede codificar sentencias SQL en un programa Fortran siempre que pueda colocar sentencias ejecutables. Si la sentencia SQL está dentro de una sentencia IF, el precompilador genera cualquier sentencia THEN y END IF necesaria.

Aplicaciones PL/I que emiten sentencias SQL

Puede codificar sentencias SQL en un programa PL/I siempre que pueda utilizar sentencias ejecutables.

Tareas relacionadas

Inclusión de SQL dinámico para sentencias SELECT de lista variable en el programa

Una sentencia SELECT de lista variable devuelve filas que contienen un número desconocido de valores de tipo desconocido. Cuando utiliza este tipo de sentencia, no sabe de antemano exactamente qué clase de variables host debe declarar para almacenar los resultados.

Referencia relacionada

Campos SQLTYPE y SQLLEN del SQLDA (Db2 SQL)

Ejecución dinámica de una sentencia con marcadores de parámetros mediante el uso de SQLDA

Tu programa puede obtener información sobre el tipo de datos de los marcadores de parámetros solicitando a Db2 que establezca los campos en el SQLDA.

Antes de empezar

Antes de ejecutar dinámicamente una sentencia con marcadores de parámetros, asigne un SQLDA con suficientes instancias de SQLVAR para representar todos los marcadores de parámetros en la sentencia SQL.

Procedimiento

Para ejecutar dinámicamente una instrucción con marcadores de parámetros utilizando el SQLDA:

1. Incluya en su programa una instrucción DESCRIBE INPUT que especifique la instrucción SQL preparada y el nombre de un SQLDA apropiado.

Db2 pone la información del marcador de parámetro solicitado en el SQLDA.
2. Codifique la aplicación de la misma manera que cualquier otra aplicación en la que ejecute una instrucción preparada mediante el uso de un SQLDA. Primero, obtenga las direcciones de las variables de host de entrada y sus variables indicadoras e inserte esas direcciones en los campos SQLDATA y SQLIND. A continuación, ejecute la instrucción SQL preparada.

Ejemplo

Supongamos que desea ejecutar la siguiente instrucción de forma dinámica:

```
DELETE FROM DSN8C10.EMP WHERE EMPNO = ?
```

Puede utilizar el siguiente código para configurar un SQLDA, obtener información de parámetros mediante la instrucción DESCRIBE INPUT y ejecutar la instrucción:

```
SQLAPTR=ADDR(INSQLDA);           /* Get pointer to SQLDA          */
SQLDAID='SQLDA';                 /* Fill in SQLDA eye-catcher    */
SQLDABC=LENGTH(INSQLDA);         /* Fill in SQLDA length          */
SQLN=1;                          /* Fill in number of SQLVARs    */
SQLD=0;                          /* Initialize # of SQLVARs used */
DO IX=1 TO SQLN;                /* Initialize the SQLVAR         */
  SQLTYPE(IX)=0;
  SQLLEN(IX)=0;
  SQLNAME(IX)='';
END;
SQLSTMT='DELETE FROM DSN8C10.EMP WHERE EMPNO = ?';
EXEC SQL PREPARE SQLOBJ FROM SQLSTMT;
EXEC SQL DESCRIBE INPUT SQLOBJ INTO :INSQLDA;
SQLDATA(1)=ADDR(HVEMP);          /* Get input data address        */
SQLIND(1)=ADDR(HVEMPIND);        /* Get indicator address         */
EXEC SQL EXECUTE SQLOBJ USING DESCRIPTOR :INSQLDA;
```

Conceptos relacionados

[Aplicaciones ensambladoras que emiten sentencias SQL](#)

Puede codificar sentencias SQL en programas ensambladores siempre que pueda utilizar sentencias ejecutables.

[Aplicaciones C y C++ que emiten sentencias SQL](#)

Puede codificar sentencias SQL en un programa C o C++ siempre que pueda utilizar sentencias ejecutables.

[Aplicaciones COBOL que emiten sentencias SQL](#)

Puede codificar sentencias SQL en determinadas secciones del programa COBOL.

[Aplicaciones Fortran que emiten sentencias SQL](#)

Puede codificar sentencias SQL en un programa Fortran siempre que pueda colocar sentencias ejecutables. Si la sentencia SQL está dentro de una sentencia IF, el precompilador genera cualquier sentencia THEN y END IF necesaria.

Aplicaciones PL/I que emiten sentencias SQL

Puede codificar sentencias SQL en un programa PL/I siempre que pueda utilizar sentencias ejecutables.

Aplicaciones REXX que emiten sentencias SQL

Puede codificar instrucciones SQL en un programa REXX siempre que pueda utilizar comandos REXX.

Tareas relacionadas

Definición de áreas de descriptor de SQL (SQLDA)

Si su programa incluye ciertas sentencias SQL, debe definir al menos *un área de descriptor SQL (SQLDA)*. Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Referencia relacionada

DESCRIBE INPUT declaración (Db2 SQL)

Comprobación de la ejecución de sentencias SQL

Después de ejecutar una sentencia SQL, el programa debe comprobar si hay errores antes de confirmar los datos y manejar los errores que representan.

Acerca de esta tarea

Puede comprobar la ejecución de las sentencias SQL de una de las siguientes maneras:

- Al mostrar campos específicos en el SQLCA.
- Comprobando SQLCODE o SQLSTATE para valores específicos.
- Utilizando la sentencia WHENEVER en su programa de aplicación.
- Probando variables indicadoras para detectar errores numéricos.
- Mediante el uso de la instrucción GET DIAGNOSTICS en su programa de aplicación para devolver toda la información de condición que resulte de la ejecución de una instrucción SQL.
- Llamando a DSNTIAR para mostrar el contenido de SQLCA.

Conceptos relacionados

Errores aritméticos y de conversión

Puede realizar un seguimiento de los errores aritméticos y de conversión utilizando variables indicadoras. Una variable indicadora contiene un valor entero pequeño que indica alguna información sobre la variable host asociada.

Tareas relacionadas

Definición del área de comunicación SQL, SQLSTATE y SQLCODE en el ensamblador

Los programas de ensamblador que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicaciones SQL, SQLSTATE y SQLCODE en C y C++

Los programas en C y C++ que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en COBOL

Los programas en COBOL que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicación SQL, SQLSTATE y SQLCODE en Fortran

Los programas en Fortran que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en PL/I

Los programas en PL/I que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en REXX

Cuando Db2 prepara un programa REXX que contiene sentencias SQL, Db2 incluye automáticamente un SQLCA en el programa.

Visualización de campos SQLCA invocando DSNTIAR

Si utiliza el SQLCA para comprobar si una sentencia SQL se ha ejecutado correctamente, el programa necesita leer los datos de los campos SQLCA adecuados. Una forma sencilla de leer estos campos es utilizar la subrutina de ensamblador DSNTIAR.

Comprobación de la ejecución de sentencias SQL utilizando SQLCA

Un modo de comprobar si una sentencia SQL se ha ejecutado correctamente es utilizar el área de comunicación SQL (SQLCA). Esta área se distingue de la comunicación con Db2.

Acerca de esta tarea

Si utiliza el SQLCA, incluya las instrucciones necesarias para mostrar la información que contiene el SQLCA en su programa de aplicación. Como alternativa, puede utilizar la instrucción GET DIAGNOSTICS, que es un estándar SQL, para diagnosticar problemas.

- Cuando Db2 procesa una instrucción SQL, coloca códigos de retorno que indican el éxito o el fracaso de la ejecución de la instrucción en SQLCODE y SQLSTATE.
- Cuando Db2 procesa una sentencia FETCH y la FETCH tiene éxito, el contenido de SQLERRD(3) en el SQLCA se establece en el número de filas devueltas.
- Cuando Db2 procesa una sentencia FETCH de varias filas, el contenido de SQLCODE se establece en +100 si la última fila de la tabla se ha devuelto con el conjunto de filas.
- Cuando Db2 procesa una instrucción UPDATE, INSERT o DELETE, y la ejecución de la instrucción se realiza correctamente, el contenido de SQLERRD(3) en el SQLCA se establece en el número de filas que se actualizan, insertan o eliminan.
- Cuando Db2 procesa una instrucción TRUNCATE y la ejecución de la instrucción se realiza correctamente, SQLERRD(3) en el SQLCA se establece en -1. No se devuelve el número de filas que se eliminan.
- Si SQLWARN0 contiene **W**, Db2 ha establecido al menos una de las banderas de advertencia SQL (SQLWARN1 a través de SQLWARNA):
 - SQLWARN1 contiene **N** para cursos no desplazables y **S** para cursos desplazables después de una instrucción OPEN CURSOR o ALLOCATE CURSOR.
 - SQLWARN4 contiene **I** para cursos deslizables insensibles, **S** para cursos deslizables estáticos sensibles y **D** para cursos deslizables dinámicos sensibles, después de una instrucción OPEN CURSOR o ALLOCATE CURSOR, o en blanco si el cursor no es deslizable.
 - SQLWARN5 contiene un valor de carácter de **1** (solo lectura), **2** (leer y eliminar) o **4** (leer, eliminar y actualizar) para indicar la operación que está permitida en la tabla de resultados del cursor.

Tareas relacionadas

Acceso a datos utilizando un cursor colocado por conjunto de filas

Un cursor colocado por conjunto de filas es un cursor que puede devolver una o varias filas para una única operación de captación. El cursor se coloca en el conjunto de filas que se van a captar.

Comprobación de la ejecución de sentencias SQL utilizando SQLCODE y SQLSTATE

Siempre que se ejecuta una sentencia SQL, los campos SQLCODE y SQLSTATE de SQLCA reciben un código de retorno.

Definición del área de comunicación SQL, SQLSTATE y SQLCODE en el ensamblador

Los programas de ensamblador que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicaciones SQL, SQLSTATE y SQLCODE en C y C++

Los programas en C y C++ que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en COBOL

Los programas en COBOL que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicación SQL, SQLSTATE y SQLCODE en Fortran

Los programas en Fortran que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en PL/I

Los programas en PL/I que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en REXX

Cuando Db2 prepara un programa REXX que contiene sentencias SQL, Db2 incluye automáticamente un SQLCA en el programa.

Referencia relacionada

[Descripción de campos de SQLCA \(Db2 SQL\)](#)

Visualización de campos SQLCA invocando DSNTIAR

Si utiliza el SQLCA para comprobar si una sentencia SQL se ha ejecutado correctamente, el programa necesita leer los datos de los campos SQLCA adecuados. Una forma sencilla de leer estos campos es utilizar la subrutina de ensamblador DSNTIAR.

Acerca de esta tarea

Debe comprobar los códigos de error antes de enviar los datos y gestionar los errores que representan. La subrutina de ensamblador DSNTIAR le ayuda a obtener una forma formateada del SQLCA y un mensaje de texto basado en el campo SQLCODE del SQLCA. Puede recuperar este mismo texto de mensaje utilizando el campo de elemento de condición MESSAGE_TEXT de la instrucción GET DIAGNOSTICS. Los programas que requieren un soporte de mensajes token largos deben codificar la instrucción GET DIAGNOSTICS en lugar de DSNTIAR.

DSNTIAR toma datos de SQLCA, los formatea en un mensaje y coloca el resultado en un área de salida de mensajes que usted proporciona en su programa de aplicación. Cada vez que utilice DSNTIAR, sobrescribirá cualquier mensaje anterior en el área de salida de mensajes. Debe mover o imprimir los mensajes antes de volver a utilizar DSNTIAR y antes de que cambie el contenido de SQLCA, para obtener una vista precisa de SQLCA.

DSNTIAR espera que el SQLCA tenga un formato determinado. Si su aplicación modifica el formato SQLCA antes de llamar a DSNTIAR, los resultados son impredecibles.

DSNTIAR

La subrutina de ensamblador DSNTIAR le ayuda a obtener una forma formateada de SQLCA y un mensaje de texto que se basa en el campo SQLCODE de SQLCA.

DSNTIAR puede ejecutarse por encima o por debajo de la línea de 16 MB de almacenamiento virtual. El módulo de objetos DSNTIAR que viene con Db2 tiene los atributos AMODE(31) y RMODE(ANY). En el momento de la instalación, DSNTIAR se vincula como AMODE(31) y RMODE(CUALQUIERA). DSNTIAR se ejecuta en modo de 31 bits si se cumple alguna de las siguientes condiciones:

- DSNTIAR está vinculado con otros módulos que también tienen los atributos AMODE(31) y RMODE(ANY).
- DSNTIAR está vinculado a una aplicación que especifica los atributos AMODE(31) y RMODE(ANY) en su JCL de edición de vínculos.
- Una aplicación carga DSNTIAR.

Al cargar DSNTIAR desde otro programa, tenga cuidado al pasar a DSNTIAR. Por ejemplo, si el programa de llamada está en modo de direccionamiento de 24 bits y DSNTIAR está cargado por encima de la línea de 16 MB, no se puede utilizar la instrucción ensambladora BALR ni la macro CALL para llamar a DSNTIAR, porque asumen que DSNTIAR está en modo de 24 bits. En su lugar, debe utilizar una instrucción que sea capaz de bifurcarse en modo de 31 bits, como BASSM.

Puede vincular (cargar) y llamar dinámicamente a DSNTIAR directamente desde un idioma que no maneje direccionamiento de 31 bits. Para ello, vincule una segunda versión de DSNTIAR con los atributos AMODE(24) y RMODE(24) en otra biblioteca de módulos de carga. Como alternativa, puede escribir un programa de lenguaje ensamblador intermedio que llame a DSNTIAR en modo de 31 bits y, a continuación, llamar a ese programa intermedio en modo de 24 bits desde su aplicación.

Para obtener más información sobre los ajustes AMODE y RMODE permitidos y predeterminados para un idioma concreto, consulte la guía de programación de aplicaciones para ese idioma. Para obtener más información sobre cómo se determinan los atributos AMODE y RMODE de una aplicación, consulte la guía del usuario del editor y cargador de enlaces para el lenguaje en el que haya escrito la aplicación.

Definición de un área de salida de mensajes

Si un programa llama a DSNTIAR, el programa debe asignar suficiente almacenamiento en el área de salida de mensajes para contener todo el texto del mensaje que devuelve DSNTIAR.

Acerca de esta tarea

Probablemente no necesitará más de 10 líneas, de 80 bytes cada una, para el área de salida de su mensaje. Un programa de aplicación solo puede tener un área de salida de mensajes.

Debe definir el área de salida del mensaje en formato VARCHAR. En este formato de carácter variable, un campo de longitud de 2 bytes precede a los datos. El campo de longitud indica a DSNTIAR cuántos bytes totales hay en el área del mensaje de salida; la longitud mínima del área de salida es de 240 bytes.

La siguiente figura muestra el formato del área de salida del mensaje, donde *length* es el campo de longitud total de 2 bytes, y la longitud de cada línea coincide con la longitud de registro lógico (*lrecl*) que especifique a DSNTIAR.

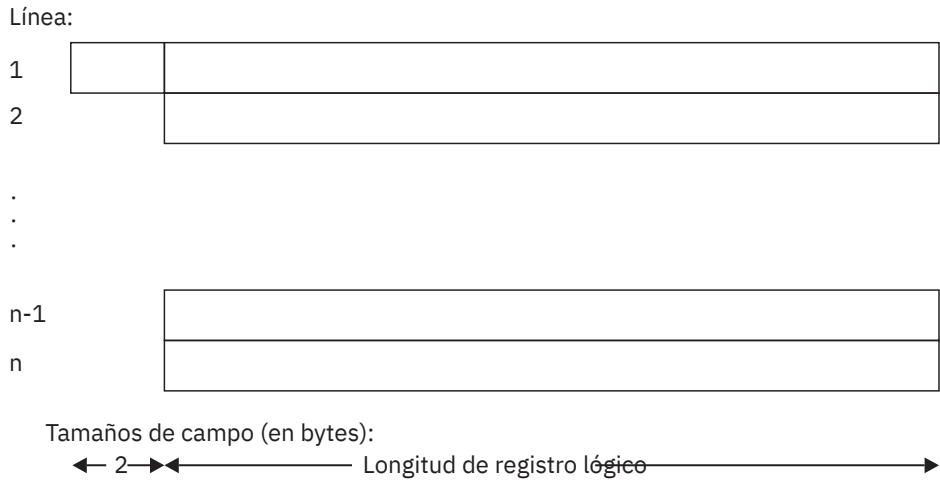


Figura 41. Formato del área de salida del mensaje

Cuando llame a DSNTIAR, debe nombrar un SQLCA y un área de mensajes de salida en los parámetros DSNTIAR. También debe proporcionar la longitud de registro lógico (*lrecl*) como un valor en el rango de 72 a 240 bytes. DSNTIAR asume que el área de mensajes contiene registros de longitud fija de longitud *lrecl*.

DSNTIAR coloca hasta 10 líneas en el área de mensaje. Si el texto de un mensaje es más largo que la longitud de registro que especifique en DSNTIAR, el mensaje de salida se divide en varios registros, en los límites de palabra si es posible. Los registros divididos están sangrados. Todos los registros comienzan con un carácter en blanco para el control de transporte. Si tiene más líneas de las que puede contener el área de salida de mensajes, DSNTIAR emite un código de retorno de 4. Un registro completamente en blanco marca el final del área de salida del mensaje.

Códigos de retorno posibles desde DSNTIAR

La subrutina de ensamblador DSNTIAR ayuda a su programa a leer la información del SQLCA. La subrutina también devuelve su propio código de retorno.

Código	Significado
--------	-------------

- | | |
|-----------|---|
| 0 | Ejecución satisfactoria. |
| 4 | Hay más datos disponibles de los que caben en el área de mensaje proporcionada. |
| 8 | La longitud del registro lógico no está en el rango de 72 a 240, ambos inclusive. |
| 12 | El área de mensaje no es lo suficientemente grande. La longitud del mensaje era de 240 o más. |
| 16 | Error en la rutina de mensajes TSO. |
| 20 | No se pudo cargar el módulo DSNTIA1. |
| 24 | Error de datos SQLCA. |

Un escenario para usar DSNTIAR

Puede utilizar la subrutina ensambladora DSNTIAR para generar el texto del mensaje de error en el SQLCA.

Supongamos que desea que su aplicación COBOL de Db2 compruebe si hay bloqueos y tiempos de espera, y quiere asegurarse de que sus cursores están cerrados antes de continuar. Utiliza la sentencia

WHENEVER SQLERROR para transferir el control a una rutina de error cuando su aplicación recibe un SQLCODE negativo.

En su rutina de error, escriba una sección que compruebe si hay SQLCODE -911 o -913. Puede recibir cualquiera de estos códigos SQLCODE cuando se produce un bloqueo o un tiempo de espera. Cuando se produce uno de estos errores, la rutina de error cierra los cursos emitiendo la instrucción:

```
EXEC SQL CLOSE cursor-name
```

Un SQLCODE de 0 o -501 resultante de esa declaración indica que el cierre se realizó correctamente.

Para utilizar DSNTIAR para generar el texto del mensaje de error, siga primero estos pasos:

1. Elija una longitud de registro lógica (*lrecl*) de las líneas de salida. Para este ejemplo, supongamos que *lrecl* es 72 (para que quepa en la pantalla de un terminal) y se almacena en la variable denominada ERROR-TEXT-LEN.
2. Defina un área de mensajes en su aplicación COBOL. Suponiendo que desea un área de hasta 10 líneas de longitud 72, debe definir un área de 720 bytes, más un área de 2 bytes que especifique la longitud total del área de salida del mensaje.

```
01  ERROR-MESSAGE.
    02  ERROR-LEN    PIC S9(4)  COMP VALUE +720.
    02  ERROR-TEXT   PIC X(72)  OCCURS 10 TIMES
                                INDEXED BY ERROR-INDEX.
77  ERROR-TEXT-LEN      PIC S9(9)  COMP VALUE +72.
```

Para este ejemplo, el nombre del área de mensajes es ERROR-MESSAGE.

3. Asegúrese de que tiene un SQLCA. Para este ejemplo, supongamos que el nombre de la SQLCA es SQLCA.

Para mostrar el contenido de SQLCA cuando SQLCODE es 0 o -501, llame a DSNTIAR después de la instrucción SQL que produce SQLCODE 0 o -501:

```
CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
```

A continuación, puede imprimir el área de salida del mensaje como lo haría con cualquier otra variable. Tu mensaje podría tener este aspecto:

```
DSNT408I SQLCODE = -501, ERROR: THE CURSOR IDENTIFIED IN A FETCH OR
CLOSE STATEMENT IS NOT OPEN
DSNT418I SQLSTATE   = 24501 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXERT SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = -315 0 0 -1 0 0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'FFFFFECE' X'00000000' X'00000000'
X'FFFFFF' X'00000000' X'00000000' SQL DIAGNOSTIC
INFORMATION
```

Comprobación de la ejecución de sentencias SQL utilizando SQLCODE y SQLSTATE

Siempre que se ejecuta una sentencia SQL, los campos SQLCODE y SQLSTATE de SQLCA reciben un código de retorno.

Procedimiento

Puede declarar SQLCODE y SQLSTATE (SQLCOD y SQLSTA en Fortran) como variables de host independientes. Si especifica la opción de precompilador STDSQL(YES), estas variables de host reciben los códigos de retorno, y no debe incluir un SQLCA en su programa.

Las aplicaciones portables deberían utilizar SQLSTATE en lugar de SQLCODE, aunque los valores de SQLCODE pueden proporcionar información específica de Db2 sobre un error o advertencia de SQL.

Una ventaja de utilizar el campo SQLCODE es que puede proporcionar información más específica que el SQLSTATE. Muchos de los SQLCODES tienen tokens asociados en el SQLCA que indican, por ejemplo, qué objeto incurrió en un error SQL. Sin embargo, una aplicación estándar de SQL utiliza solo SQLSTATE.

SQLCODE

Db2 devuelve los siguientes códigos en SQLCODE:

- Si SQLCODE = 0, la ejecución se realizó correctamente.
- Si SQLCODE > 0, la ejecución se realizó correctamente con una advertencia.
- Si SQLCODE < 0, la ejecución no ha sido satisfactoria.

El SQLCODE 100 indica que no se han encontrado datos.

El significado de los códigos SQL distintos de 0 y 100 varía según el producto concreto que implementa SQL.

SQLSTATE

El SQLSTATE permite a un programa de aplicación comprobar errores de la misma manera para diferentes IBM sistemas de gestión de bases de datos.

Tareas relacionadas

[Definición del área de comunicación SQL, SQLSTATE y SQLCODE en el ensamblador](#)

Los programas de ensamblador que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

[Definición del área de comunicaciones SQL, SQLSTATE y SQLCODE en C y C++](#)

Los programas en C y C++ que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

[Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en COBOL](#)

Los programas en COBOL que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

[Definición del área de comunicación SQL, SQLSTATE y SQLCODE en Fortran](#)

Los programas en Fortran que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

[Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en PL/I](#)

Los programas en PL/I que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

[Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en REXX](#)

Cuando Db2 prepara un programa REXX que contiene sentencias SQL, Db2 incluye automáticamente un SQLCA en el programa.

Referencia relacionada

[Valores de SQLSTATE y códigos de error comunes \(Db2 Codes\)](#)

Comprobación de la ejecución de sentencias SQL mediante la sentencia WHENEVER

La sentencia WHENEVER hace que Db2 compruebe el SQLCA y continúe procesando su programa. Si se produce un error, una excepción o una advertencia, Db2 se dirige a otra área de su programa. El área de gestión de condiciones de su programa puede entonces examinar el SQLCODE o SQLSTATE para reaccionar específicamente al error o excepción.

Acerca de esta tarea

La sentencia WHENEVER no es compatible con REXX.

La sentencia WHENEVER le permite especificar qué hacer si se cumple una condición general. Puede especificar más de una instrucción WHENEVER en su programa. Cuando haces esto, la primera instrucción WHENEVER se aplica a todas las instrucciones SQL posteriores en el programa fuente hasta la siguiente instrucción WHENEVER.

La declaración WHENEVER tiene este aspecto:

```
EXEC SQL
  WHENEVER condition action
END-EXEC
```

La condición de la sentencia WHENEVER es uno de estos tres valores:

SQLWARNING

Indica qué hacer cuando SQLWARN0 = W o SQLCODE contiene un valor positivo distinto de 100. Db2 puede establecer SQLWARN0 por varias razones, por ejemplo, si el valor de una columna se trunca al trasladarlo a una variable de host. Es posible que su programa no considere esto como un error.

SQLERROR

Indica qué hacer cuando Db2 devuelve un código de error como resultado de una instrucción SQL (SQLCODE < 0).

NOT FOUND

Indica qué hacer cuando Db2 no puede encontrar una fila que satisfaga su instrucción SQL o cuando no hay más filas que recuperar (SQLCODE = 100).

La acción de la sentencia WHENEVER es uno de estos dos valores:

CONTINUE

Especifica la siguiente instrucción secuencial del programa fuente.

GOTO o IR A etiqueta de host

Especifica la declaración identificada por *host-label*. Para *la etiqueta del host*, sustituya un solo token, precedido por dos puntos opcionales. El formato del símbolo depende del lenguaje principal. En COBOL, por ejemplo, puede ser *un nombre de sección* o *un nombre de párrafo* sin calificar.

La instrucción WHENEVER debe preceder a la primera instrucción SQL a la que afecte. Sin embargo, si su programa comprueba SQLCODE directamente, debe comprobar SQLCODE después de cada instrucción SQL.

Conceptos relacionados

[Aplicaciones REXX que emiten sentencias SQL](#)

Puede codificar instrucciones SQL en un programa REXX siempre que pueda utilizar comandos REXX.

Referencia relacionada

[WHENEVER declaración \(Db2 SQL\)](#)

Comprobación de la ejecución de sentencias SQL utilizando la instrucción GET DIAGNOSTICS

Una forma de comprobar si una sentencia SQL se ha ejecutado correctamente es solicitar a Db2 que devuelva información de diagnóstico sobre la última sentencia SQL ejecutada.

Procedimiento

Puede utilizar la instrucción GET DIAGNOSTICS para devolver información de diagnóstico sobre la última instrucción SQL que se ejecutó.

Puede solicitar elementos individuales de información de diagnóstico de los siguientes grupos de elementos:

- Elementos de la instrucción, que contienen información sobre la instrucción SQL en su conjunto
- Elementos de condición, que contienen información sobre cada error o advertencia que se produjo durante la ejecución de la instrucción SQL
- Elementos de conexión, que contienen información sobre la instrucción SQL si se trataba de una instrucción CONNECT

Además de solicitar elementos individuales, puede solicitar que GET DIAGNOSTICS devuelva todos los elementos de diagnóstico que se establecen durante la ejecución de la última instrucción SQL como una sola cadena.

En los procedimientos SQL, también puede recuperar información de diagnóstico mediante el uso de controladores. Los controladores indican al procedimiento qué hacer si se produce un error concreto.

Utilice la sentencia GET DIAGNOSTICS para gestionar múltiples errores SQL que podrían resultar de la ejecución de una sola sentencia SQL. En primer lugar, compruebe SQLSTATE (o SQLCODE) para determinar si la información de diagnóstico debe recuperarse mediante GET DIAGNOSTICS. Este método es especialmente útil para diagnosticar problemas que resultan de una instrucción INSERT de varias filas que se especifica como NOT ATOMIC CONTINUE ON SQLEXCEPTION de instrucciones MERGE de varias filas.

Incluso si solo utiliza la instrucción GET DIAGNOSTICS en su programa de aplicación para comprobar las condiciones, debe incluir las instrucciones necesarias para utilizar SQLCA o debe declarar SQLSTATE (o SQLCODE) por separado en su programa.

Cuando utiliza la instrucción GET DIAGNOSTICS, asigna la información de diagnóstico solicitada a variables de host. Declare cada variable de host de destino con un tipo de datos que sea compatible con el tipo de datos del elemento solicitado.

Para recuperar información de estado, primero debe recuperar el número de elementos de estado (es decir, el número de errores y advertencias que Db2 detectó durante la ejecución de la última instrucción SQL). El número de elementos de condición es al menos uno. Si la última instrucción SQL devolvió SQLSTATE '00000' (o SQLCODE 0), el número de elementos de condición es uno.

Ejemplo: Uso de GET DIAGNOSTICS con INSERT de varias filas

Desea mostrar información de diagnóstico para cada condición que pueda producirse durante la ejecución de una instrucción INSERT de varias filas en su programa de aplicación. Usted especifica la sentencia INSERT como NOT ATOMIC CONTINUE ON SQLEXCEPTION, lo que significa que la ejecución continúa independientemente del fallo de cualquier inserción de una sola fila. Db2 no inserta la fila que se procesó en el momento del error.

En el siguiente ejemplo, la primera sentencia GET DIAGNOSTICS devuelve el número de filas insertadas y el número de condiciones devueltas. La segunda sentencia GET DIAGNOSTICS devuelve los siguientes elementos para cada condición: SQLCODE, SQLSTATE y el número de la fila (en el conjunto de filas que se estaba insertando) para la que se produjo la condición.

```

EXEC SQL BEGIN DECLARE SECTION;
  long row_count, num_condns, i;
  long ret_sqlcode, row_num;
  char ret_sqlstate[6];
  ...
EXEC SQL END DECLARE SECTION;
  ...
EXEC SQL
  INSERT INTO DSN8C10.ACT
    (ACTNO, ACTKWD, ACTDESC)
  VALUES (:hva1, :hva2, :hva3)
  FOR 10 ROWS
  NOT ATOMIC CONTINUE ON SQLEXCEPTION;

EXEC SQL GET DIAGNOSTICS
  :row_count = ROW_COUNT, :num_condns = NUMBER;
  printf("Number of rows inserted = %d\n", row_count);

  for (i=1, i<=num_condns; i++) {
    EXEC SQL GET DIAGNOSTICS CONDITION :i
      :ret_sqlcode = DB2_RETURNED_SQLCODE,

```

```

:ret_sqlstate = RETURNED_SQLSTATE,
:row_num      = DB2_ROW_NUMBER;
printf("SQLCODE = %d, SQLSTATE = %s, ROW NUMBER = %d\n",
       ret_sqlcode, ret_sqlstate, row_num);
}

```

En la tabla de actividades, la columna ACTNO se define como SMALLINT. Supongamos que declara la matriz de variables de host hva1 como una matriz con tipo de datos long, y rellena la matriz de modo que el valor del cuarto elemento sea 32768.

Si comprueba los valores SQLCA después de la instrucción INSERT, el valor de SQLCODE es igual a 0, el valor de SQLSTATE es '00000' y el valor de SQLERRD(3) es 9 para el número de filas que se insertaron. Sin embargo, la instrucción INSERT especificaba que se debían insertar 10 filas.

La sentencia GET DIAGNOSTICS le proporciona la información que necesita para corregir los datos de la fila que no se insertó. El resultado impreso de su programa tiene este aspecto:

```

Number of rows inserted = 9
SQLCODE = -302, SQLSTATE = 22003, ROW NUMBER = 4

```

El valor 32768 para la variable de entrada es demasiado grande para la columna de destino ACTNO. Puede imprimir el elemento de condición MESSAGE_TEXT.

Qué hacer a continuación

Cuando utiliza la instrucción GET DIAGNOSTICS, asigna la información de diagnóstico solicitada a variables de host. Declare cada variable de host de destino con un tipo de datos que sea compatible con el tipo de datos del elemento solicitado. Para obtener más información, consulte ["Tipos de datos para los elementos GET DIAGNOSTICS" en la página 565](#).

Para recuperar información de estado, primero debe recuperar el número de elementos de estado (es decir, el número de errores y advertencias que Db2 detectó durante la ejecución de la última instrucción SQL). El número de elementos de condición es al menos uno. Si la última instrucción SQL devolvió SQLSTATE '00000' (o SQLCODE 0), el número de elementos de condición es uno.

Conceptos relacionados

[Manejadores en un procedimiento SQL](#)

Si se produce un error cuando se ejecuta el procedimiento SQL, el procedimiento finaliza a menos que incluya sentencias para indicar al procedimiento que realice alguna otra acción. Estas sentencias se denominan manejadores.

Referencia relacionada

[Tipos de datos para los elementos GET DIAGNOSTICS](#)

Puede utilizar la sentencia GET DIAGNOSTICS para solicitar información sobre la sentencia, condición y conexión para la última sentencia SQL que se ha ejecutado. Debe declarar cada variable host de destino con un tipo de datos que sea compatible con el tipo de datos del elemento solicitado.

[GET DIAGNOSTICS declaración \(Db2 SQL\)](#)

Información relacionada

[-302 \(Db2 Codes\)](#)

Tipos de datos para los elementos GET DIAGNOSTICS

Puede utilizar la sentencia GET DIAGNOSTICS para solicitar información sobre la sentencia, condición y conexión para la última sentencia SQL que se ha ejecutado. Debe declarar cada variable host de destino con un tipo de datos que sea compatible con el tipo de datos del elemento solicitado.

La siguiente tabla resume los tipos de datos para los elementos de información de diagnóstico que puede solicitar mediante la instrucción GET DIAGNOSTICS.

OBTENER DIAGNÓSTICO artículo	Tipo de datos	Descripción
DB2_GET_DIAGNOSTICS_DIAGNOSTICS	VARCHAR(32672)	Después de una sentencia GET DIAGNOSTICS, todos los diagnósticos como una sola cadena si se produjo algún error o advertencia
DB2_LAST_ROW	ENTERO	Después de una sentencia FETCH de varias filas, el valor de +100 si la última fila de la tabla está en el conjunto de filas devuelto
DB2_NUMBER_PARAMETER_MARKERS	ENTERO	Después de una sentencia PREPARE, el número de marcadores de parámetros en la sentencia preparada
DB2_NUMBER_RESULT_SETS	ENTERO	Después de una sentencia CALL que invoca un procedimiento almacenado, el número de conjuntos de resultados que devuelve el procedimiento
DB2_NUMBER_ROWS	DECIMAL(31,0)	Después de una instrucción OPEN o FETCH para la que se conoce el tamaño de la tabla de resultados, el número de filas de la tabla de resultados Después de una sentencia PREPARE, el número estimado de filas en la tabla de resultados para la sentencia preparada Para cursos DINÁMICOS SENSIBLES, el número aproximado de filas De lo contrario, o si el servidor solo devuelve un SQLCA, el valor cero
DB2_RETURN_STATUS	ENTERO	Después de una sentencia CALL que invoque un procedimiento SQL, el estado de retorno si el procedimiento contiene una sentencia RETURN
DB2_SQL_ATTR_CURSOR_HOLD	CHAR(1)	Después de una instrucción ALLOCATE o OPEN, si el cursor puede mantenerse abierto a través de múltiples unidades de trabajo (S o N)
DB2_SQL_ATTR_CURSOR_ROWSHEET	CHAR(1)	Después de una sentencia ALLOCATE u OPEN, si el cursor puede utilizar el posicionamiento de conjuntos de filas (Y o N)
DB2_SQL_ATTR_CURSOR_SCROLLABLE	CHAR(1)	Después de una instrucción ALLOCATE o OPEN, si el cursor se puede desplazar (Y o N)
DB2_SQL_ATTR_CURSOR_SENSITIVITY	CHAR(1)	Después de una sentencia ALLOCATE u OPEN, si el cursor muestra actualizaciones realizadas por otros procesos (sensibilidad I o S)
DB2_SQL_ATTR_CURSOR_TYPE	CHAR(1)	Después de una sentencia ALLOCATE u OPEN, si el cursor está hacia adelante (F), declarado estático (S para INSENSITIVE o SENSITIVE STATIC), o dinámico (D para SENSITIVE DYNAMIC).
MÁS	CHAR(1)	Después de cualquier instrucción SQL, este elemento indica si algunos elementos de condiciones se descartaron debido a almacenamiento insuficiente (S o N).

OBTENER DIAGNÓSTICO artículo	Tipo de datos	Descripción
NUMBER	ENTERO	Después de cualquier instrucción SQL, este elemento contiene el número de elementos de condición. Si no se ha producido ningún aviso o error, o si no se ha ejecutado ninguna instrucción SQL anterior, el número que se devuelve es 1.
ROW_COUNT	DECIMAL(31,0)	Después de insertar, actualizar, eliminar o recuperar, este elemento contiene el número de filas que se eliminan, insertan, actualizan o recuperan. Después de PREPARE, este elemento contiene el número estimado de filas de resultados en la sentencia preparada. Después de TRUNCATE, contiene -1.
DB2_SQL_NESTING_LEVEL	ENTERO	Después de una sentencia CALL, este elemento identifica el nivel actual de anidamiento o recursión en vigor cuando se ejecutó la sentencia GET DIAGNOSTICS. Cada nivel de anidación corresponde a una invocación anidada o recursiva de una función SQL empaquetada, un procedimiento SQL empaquetado o un desencadenador. Si la sentencia GET DIAGNOSTICS se ejecuta fuera de un nivel de anidamiento, se devuelve el valor cero. Cuando una aplicación se conecta a otro servidor, el valor se restablece a cero.
CATALOG_NAME	VARCHAR(128)	El nombre de servidor de la tabla que posee una restricción que causó un error, o que causó una regla de acceso o una violación de verificación
CONDITION_NUMBER	ENTERO	El número de la condición
CURSOR_NAME	VARCHAR(128)	El nombre de un cursor en un estado de cursor no válido
DB2_ERROR_CODE1	ENTERO	Un código de error interno
DB2_ERROR_CODE2	ENTERO	Un código de error interno
DB2_ERROR_CODE3	ENTERO	Un código de error interno
DB2_ERROR_CODE4	ENTERO	Un código de error interno
DB2_INTERNAL_ERROR_POINT ER	ENTERO	Para algunos errores, un valor negativo que es un indicador de error interno
DB2_LINE_NUMBER	ENTERO	El número de línea donde se encuentra un error al analizar una sentencia dinámica, o al analizar, vincular o ejecutar una sentencia CREATE o ALTER para un procedimiento SQL nativo, una función SQL compilada o un disparador El número de línea cuando una instrucción CALL invoca un procedimiento SQL nativo y el procedimiento devuelve un error
DB2_MESSAGE_ID	CHAR (10)	El ID de mensaje que corresponde al mensaje que está contenido en el elemento de diagnóstico MESSAGE_TEXT.

OBTENER DIAGNÓSTICO artículo	Tipo de datos	Descripción
DB2_MODULE_DETECTING_ERR OR	CHAR(8)	El módulo que detectó el error
DB2_ORDINAL_TOKEN_n	VARCHAR(515)	<i>La enésima ficha</i> , donde n es un valor de 1 a 100
DB2_REASON_CODE	ENTERO	El código de motivo para errores que tienen un código de motivo en el texto del mensaje
DB2_RETURNED_SQLCODE	ENTERO	El SQLCODE para la condición
DB2_ROW_NUMBER	DECIMAL(31,0)	Después de cualquier instrucción SQL que implique varias filas, este elemento contiene el número de fila en el que Db2 detectó la condición
DB2_SQLERRD1	ENTERO	El valor sqlerrd(1) de la SQLCA que devuelve el servidor, o cero
DB2_SQLERRD2	ENTERO	El valor sqlerrd(2) de la SQLCA que devuelve el servidor, o cero
DB2_SQLERRD3	ENTERO	El valor sqlerrd(3) de la SQLCA que devuelve el servidor, o cero
DB2_SQLERRD4	ENTERO	El valor sqlerrd(4) de la SQLCA que devuelve el servidor, o cero
DB2_SQLERRD5	ENTERO	El valor sqlerrd(5) de SQLCA devuelto por el servidor, o cero
DB2_SQLERRD6	ENTERO	El valor sqlerrd(6) de SQLCA que devuelve el servidor, o cero
DB2_TOKEN_COUNT	ENTERO	El número de fichas disponibles para la condición
MESSAGE_TEXT	VARCHAR(32672)	El texto del mensaje asociado con el SQLCODE
RETURNED_SQLSTATE	CHAR (5)	El SQLSTATE para la condición
server_name	VARCHAR(128)	Después de una instrucción CONNECT, DISCONNECT o SET CONNECTION, el nombre del servidor especificado en la instrucción
DB2_AUTHENTICATION_TYPE	CHAR(1)	El tipo de autenticación (S, C, D, E o en blanco)
DB2_AUTHORIZATION_ID	VARCHAR(128)	El ID de autorización que utiliza el servidor conectado
DB2_CONNECTION_STATE	ENTERO	Si la conexión es desconectada (-1), local (0) o remota (1)
DB2_CONNECTION_STATUS	ENTERO	Si se pueden realizar actualizaciones para la unidad de trabajo actual (1 para Sí, 2 para No)
DB2_ENCRYPTION_TYPE	CHAR(1)	El nivel de encriptación de la conexión: (A) solo se cifran los tokens de autenticación (ID de autenticación y contraseña) (D) todos los datos de la conexión están cifrados
DB2_PRODUCT_ID	VARCHAR(8)	La firma del producto de Tiffany & Co. (Db2)

OBTENER DIAGNÓSTICO artículo	Tipo de datos	Descripción
DB2_SERVER_CLASS_NAME	CHAR(128)	Después de una instrucción CONNECT o SET CONNECTION, el nombre de clase del servidor Db2
ALL	VARCHAR(32672)	Todos los elementos de diagnóstico establecidos para la última instrucción SQL combinados en una cadena, en forma de una lista separada por punto y coma de toda la información de diagnóstico disponible

Tareas relacionadas

Comprobación de la ejecución de sentencias SQL utilizando la instrucción GET DIAGNOSTICS
Una forma de comprobar si una sentencia SQL se ha ejecutado correctamente es solicitar a Db2 que devuelva información de diagnóstico sobre la última sentencia SQL ejecutada.

Referencia relacionada

[GET DIAGNOSTICS declaración \(Db2 SQL \)](#)

Manejo de códigos de error de SQL

Los programas de aplicación pueden solicitar más información sobre los códigos de error SQL en Db2.

Acerca de esta tarea

Db2 , como se muestra en la siguiente tabla, establece el valor SQLCODE después de ejecutar cada instrucción.

Valor SQLCODE	Significado	Descripciones de SQLCODE
SQLCODE = 0	Ejecución correcta, si el campo " SQLWARNO " está en blanco. Si SQLWARN0 = 'W', ejecución correcta con advertencia.	000
SQLCODE = 100	No se han encontrado datos. Por ejemplo, una sentencia FETCH no devolvió datos porque el cursor se situó después de la última fila de la tabla de resultados.	+100
SQLCODE > 0 y no = 100	Ejecución correcta con una advertencia.	+ sqlcode-num
CÓDIGO SQL < 0	La ejecución no se ha realizado correctamente.	- sqlcode-num

Para ver las descripciones en formato PDF de los códigos SQL que podría emitir Db2 12 , consulte  [Códigos](#).

Procedimiento

Para manejar los códigos de error SQL de los programas de aplicación en el lenguaje del host, tome medidas basadas en el lenguaje de programación que utilice, como se describe en los siguientes temas.

- “Gestión de códigos de error de SQL en aplicaciones de ensamblador” en la página 602
- “Gestión de códigos de error de SQL en aplicaciones C y C++” en la página 652
- “Gestión de códigos de error de SQL en aplicaciones Cobol” en la página 721
- “Gestión de códigos de error de SQL en aplicaciones Fortran” en la página 730

- “Gestión de códigos de error de SQL en aplicaciones PL/I” en la página 741
- “Gestión de códigos de error de SQL en aplicaciones REXX” en la página 793

Tareas relacionadas

Visualización de campos SQLCA invocando DSNTIAR

Si utiliza el SQLCA para comprobar si una sentencia SQL se ha ejecutado correctamente, el programa necesita leer los datos de los campos SQLCA adecuados. Una forma sencilla de leer estos campos es utilizar la subrutina de ensamblador DSNTIAR.

Comprobación de la ejecución de sentencias SQL utilizando la instrucción GET DIAGNOSTICS

Una forma de comprobar si una sentencia SQL se ha ejecutado correctamente es solicitar a Db2 que devuelva información de diagnóstico sobre la última sentencia SQL ejecutada.

Referencia relacionada

[GET DIAGNOSTICS declaración \(Db2 SQL \)](#)

Errores aritméticos y de conversión

Puede realizar un seguimiento de los errores aritméticos y de conversión utilizando variables indicadoras. Una variable indicadora contiene un valor entero pequeño que indica alguna información sobre la variable host asociada.

Los errores de conversión numérica o de caracteres o los errores de expresión aritmética pueden establecer una variable indicadora en -2. Por ejemplo, la división por cero y el desbordamiento aritmético no detienen necesariamente la ejecución de una instrucción SELECT. Si utiliza variables de indicador y se produce un error en la lista SELECT, la instrucción puede continuar ejecutándose y devolver datos correctos para las filas en las que no se produce el error.

Para las filas en las que se produce un error de conversión o de expresión aritmética, la variable indicadora indica que uno o más elementos seleccionados no tienen un valor significativo. La variable indicadora marca este error con un código de estado de consulta (-2) para la variable de host afectada y un código de error de consulta (SQL CODE) de +802 (SQLSTATE '01519') en el área de control de consultas (SQLCA).

Escritura de aplicaciones que permiten al usuario crear y modificar tablas

Puede escribir una aplicación Db2 que permite a los usuarios crear tablas nuevas, añadirles columnas, aumentar la longitud de las columnas, reorganizar las columnas y descartarlas.

Procedimiento

Para crear nuevas tablas:

- Utilice la instrucción CREATE TABLE.

Para añadir columnas o aumentar la longitud de las columnas:

- Utilice la instrucción ALTER TABLE con la cláusula ADD COLUMN o la cláusula ALTER COLUMN.

Las columnas añadidas contienen inicialmente el valor nulo o un valor predeterminado. Tanto CREATE TABLE como ALTER TABLE, como cualquier declaración de definición de datos, son relativamente costosos de ejecutar. Tenga en cuenta también los efectos de los bloqueos.

Para soltar columnas:

- Utilice la instrucción ALTER TABLE con la cláusula DROP COLUMN.

Eliminar una columna de una tabla es un cambio pendiente de definición, a menos que el espacio de la tabla se defina con la opción DEFINE NO. La columna no se elimina de la tabla hasta que se ejecuta la utilidad REORG en el espacio de la tabla. Si planea eliminar una columna de una tabla además de realizar otros cambios en la tabla, haga todos los cambios que surtan efecto inmediatamente, antes de emitir la instrucción ALTER TABLE con la cláusula DROP COLUMN.

Para reorganizar las columnas:

- Elimine la tabla y vuelva a crearla con las columnas que desee y en el orden que desee. Consideré la posibilidad de crear una vista de la tabla que incluya solo las columnas que desee, en el orden que desee, como alternativa a redefinir la tabla.

Tareas relacionadas

[Inclusión de SQL dinámico en el programa](#)

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

Referencia relacionada

[ALTER TABLE declaración \(Db2 SQL\)](#)

[CREATE TABLE declaración \(Db2 SQL\)](#)

[CREATE VIEW declaración \(Db2 SQL\)](#)

Guardar sentencias SQL que se convierten en solicitudes de usuario

Si su programa convierte las solicitudes de los usuarios en sentencias SQL y permite que los usuarios guarden sus solicitudes, su programa puede mejorar el rendimiento si guarda dichas sentencias convertidas.

Acerca de esta tarea

Un programa traduce las solicitudes de los usuarios en instrucciones SQL antes de ejecutarlas, y los usuarios pueden guardar una solicitud.

Procedimiento

Guarde las sentencias SQL correspondientes en una tabla con una columna que tenga un tipo de datos *VARCHAR(n)*, donde *n* es la longitud máxima de cualquier sentencia SQL.

Debe guardar las instrucciones SQL de origen, no las versiones preparadas. Esto significa que debe recuperar y preparar cada estado antes de ejecutar la versión almacenada en la tabla. En esencia, su programa prepara una instrucción SQL a partir de una cadena de caracteres y la ejecuta dinámicamente.

Tareas relacionadas

[Inclusión de SQL dinámico en el programa](#)

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

Datos XML en aplicaciones SQL incorporadas

Las aplicaciones de SQL incorporado que se escriben en lenguaje ensamblador, C, C++, COBOL o PL/I puede actualizar y recuperar datos en columnas XML.

En las aplicaciones SQL incrustadas, puede:

- Almacenar un documento XML completo en una columna XML mediante sentencias INSERT o UPDATE.
- Recuperar un documento XML completo de una columna XML utilizando sentencias SELECT.
- Recuperar una secuencia de un documento en una columna XML utilizando la función SQL XMLQUERY dentro de una instrucción SELECT o FETCH, para recuperar la secuencia en una cadena de texto XML en la base de datos, y luego recuperar los datos en una variable de aplicación.

Recomendación : Siga estas pautas cuando escriba aplicaciones SQL incrustadas:

- Evite utilizar las funciones XMLPARSE y XMLSERIALIZE.

Deje que Db2 realice las conversiones entre los formatos XML externos e internos de forma implícita.

- Utilice variables de host XML para la entrada y la salida.

Al hacerlo, se permite que Db2 procese valores como datos XML en lugar de datos de caracteres o cadenas binarias. Si la aplicación no puede utilizar variables de host XML, debe utilizar variables de host de cadena binaria para minimizar los problemas de conversión de caracteres.

- Evite la conversión de caracteres utilizando variables de host de XML (UTF-8) para la entrada y salida de valores XML siempre que sea posible.

Tipos de datos de variable host para datos XML en aplicaciones de SQL incorporado

Db2 proporciona tipos de variables de host XML para el ensamblador, C, C++, COBOL y PL/I.

Estos tipos son:

- XML AS BLOB
- XML AS CLOB
- XML COMO DBCLOB
- XML COMO BLOB_FILE (C, C++ o PL/I) o XML COMO BLOB-FILE (COBOL)
- XML COMO CLOB_FILE (C, C++ o PL/I) o XML COMO CLOB-FILE (COBOL)
- XML AS DBCLOB_FILE (C, C++ o PL/I) o XML AS DBCLOB-FILE (COBOL)

Los tipos de variables de host XML solo son compatibles con el tipo de datos de columna XML.

Puede utilizar variables de host BLOB, CLOB, DBCLOB, CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, BINARY o VARBINARY para actualizar columnas XML. Puede convertir los tipos de datos de variables de host al tipo XML utilizando la función XMLPARSE, o puede dejar que el servidor de base de datos Db2 realice la conversión de forma implícita.

Puede utilizar variables de host BLOB, CLOB, DBCLOB, CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, BINARY o VARBINARY para recuperar datos de columnas XML. Puede convertir los datos XML al tipo de variable de host utilizando la función XMLSERIALIZE, o puede dejar que el servidor de base de datos Db2 realice la conversión de forma implícita.

Los siguientes ejemplos muestran cómo declarar variables de host XML en cada lenguaje compatible. En cada tabla, la columna de la izquierda contiene la declaración que codifica en su programa de aplicación. La columna de la derecha contiene la declaración que genera Db2 .

Declaraciones de variables de host XML en ensamblador

La siguiente tabla muestra las declaraciones de lenguaje ensamblador para algunos tipos XML típicos.

Tabla 90. Ejemplo de declaraciones de variables XML de ensamblador

Usted declara esta variable	Db2 genera esta variable
BLOB_XML SQL TYPE IS XML AS BLOB 1M	<pre>BLOB_XML DS 0FL4 BLOB_XML_LENGTH DS FL4 BLOB_XML_DATA DS CL65535"1" en la página 573 ORG **+(983041)</pre>
CLOB_XML SQL TYPE IS XML AS CLOB 40000K	<pre>CLOB_XML DS 0FL4 CLOB_XML_LENGTH DS FL4 CLOB_XML_DATA DS CL65535"1" en la página 573 ORG **+(40894465)</pre>
DBCLOB_XML SQL TYPE IS XML AS DBCLOB 4000K	<pre>DBCLOB_XML DS 0FL4 DBCLOB_XML_LENGTH DS FL4 DBCLOB_XML_DATA DS GL65534"2" en la página 573 ORG **+(4030466)</pre>

Tabla 90. Ejemplo de declaraciones de variables XML de ensamblador (continuación)

Usted declara esta variable	Db2 genera esta variable
BLOB_XML_FILE SQL TYPE IS XML AS BLOB_FILE	<pre>BLOB_XML_FILE DS OFL4 BLOB_XML_FILE_NAME_LENGTH DS FL4 BLOB_XML_FILE_DATA_LENGTH DS FL4 BLOB_XML_FILE_FILE_OPTIONS DS FL4 BLOB_XML_FILE_NAME DS CL255</pre>
CLOB_XML_FILE SQL TYPE IS XML AS CLOB_FILE	<pre>CLOB_XML_FILE DS OFL4 CLOB_XML_FILE_NAME_LENGTH DS FL4 CLOB_XML_FILE_DATA_LENGTH DS FL4 CLOB_XML_FILE_FILE_OPTIONS DS FL4 CLOB_XML_FILE_NAME DS CL255</pre>
DBCLOB_XML_FILE SQL TYPE IS XML AS DBCLOB_FILE	<pre>DBCLOB_XML_FILE DS OFL4 DBCLOB_XML_FILE_NAME_LENGTH DS FL4 DBCLOB_XML_FILE_DATA_LENGTH DS FL4 DBCLOB_XML_FILE_FILE_OPTIONS DS FL4 DBCLOB_XML_FILE_NAME DS CL255</pre>

Notas:

1. Debido a que el lenguaje ensamblador permite declaraciones de caracteres de no más de 65535 bytes, Db2 separa en dos partes las declaraciones de lenguaje de host para las variables de host XML AS BLOB y XML AS CLOB que tienen más de 65535 bytes.
2. Debido a que el lenguaje ensamblador permite declaraciones gráficas de no más de 65534 bytes, Db2 separa en dos partes las declaraciones del lenguaje host para las variables host XML AS DBCLOB que tienen más de 65534 bytes.

Declaraciones de variables de host XML en C y C++

La siguiente tabla muestra las declaraciones de lenguaje C y C++ que genera el precompilador Db2 para algunos tipos XML típicos. Las declaraciones que genera el coprocesador de Db2 podrían ser diferentes.

Tabla 91. Ejemplos de declaraciones de variables en lenguaje C

Usted declara esta variable	Db2 genera esta variable
SQL TYPE IS XML AS BLOB (1M) blob_xml;	<pre>struct { unsigned long length; char data??(1048576??); } blob_xml;</pre>
SQL TYPE IS XML AS CLOB(40000K) clob_xml;	<pre>struct { unsigned long length; char data??(40960000??); } clob_xml;</pre>
SQL TYPE IS XML AS DBCLOB (4000K) dbblob_xml;	<pre>struct { unsigned long length; unsigned short data??(4096000??); } dbblob_xml;</pre>
SQL TYPE IS XML AS BLOB_FILE blob_xml_file;	<pre>struct { unsigned long name_length; unsigned long data_length; unsigned long file_options; char name??(255??); } blob_xml_file;</pre>

Tabla 91. Ejemplos de declaraciones de variables en lenguaje C (continuación)

Usted declara esta variable	Db2 genera esta variable
SQL TYPE IS XML AS CLOB_FILE clob_xml_file;	<pre>struct { unsigned long name_length; unsigned long data_length; unsigned long file_options; char name??(255??); } clob_xml_file;</pre>
SQL TYPE IS XML AS DBCLOB_FILE dbblob_xml_file;	<pre>struct { unsigned long name_length; unsigned long data_length; unsigned long file_options; char name??(255??); } dbblob_xml_file;</pre>

Declaraciones de variables de host XML en COBOL

Las declaraciones que se generan para COBOL difieren, dependiendo de si se utiliza el precompilador Db2 o el coprocesador Db2 .

La siguiente tabla muestra las declaraciones COBOL que el precompilador Db2 genera para algunos tipos XML típicos.

Tabla 92. Ejemplos de declaraciones de variables COBOL por el precompilador de Db2

Usted declara esta variable	Db2 el precompilador genera esta variable
01 BLOB-XML USAGE IS SQL TYPE IS XML AS BLOB(1M).	<pre>01 BLOB-XML. 02 BLOB-XML-LENGTH PIC 9(9) COMP. 02 BLOB-XML-DATA: 49 FILLER PIC X(32767). "1" en la página 575 49 FILLER PIC X(32767). Repeat 30 times : 49 FILLER PIC X(1048576-32*32767).</pre>
01 CLOB-XML USAGE IS SQL TYPE IS XML AS CLOB(40000K).	<pre>01 CLOB-XML. 02 CLOB-XML-LENGTH PIC 9(9) COMP. 02 CLOB-XML-DATA: 49 FILLER PIC X(32767). "1" en la página 575 49 FILLER PIC X(32767). Repeat 1248 times : 49 FILLER PIC X(40960000-1250*32767).</pre>
01 DBCLOB-XML USAGE IS SQL TYPE IS XML AS DBCLOB(4000K).	<pre>01 DBCLOB-XML. 02 DBCLOB-XML-LENGTH PIC 9(9) COMP. 02 DBCLOB-XML-DATA: 49 FILLER PIC G(32767) USAGE DISPLAY-1. "2" en la página 575 49 FILLER PIC G(32767) USAGE DISPLAY-1. Repeat 123 times : 49 FILLER PIC G(4096000-125*32767) USAGE DISPLAY-1.</pre>

Tabla 92. Ejemplos de declaraciones de variables COBOL por el precompilador de Db2 (continuación)

Usted declara esta variable	Db2 el precompilador genera esta variable
01 BLOB-XML-FILE USAGE IS SQL TYPE IS XML AS BLOB-FILE.	01 BLOB-XML-FILE. 49 BLOB-XML-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC. 49 BLOB-XML-FILE-DATA-LENGTH PIC S9(9) COMP-5. 49 BLOB-XML-FILE-FILE-OPTION PIC S9(9) COMP-5. 49 BLOB-XML-FILE-NAME PIC X(255).
01 CLOB-XML-FILE USAGE IS SQL TYPE IS XML AS CLOB-FILE.	01 CLOB-XML-FILE. 49 CLOB-XML-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC. 49 CLOB-XML-FILE-DATA-LENGTH PIC S9(9) COMP-5. 49 CLOB-XML-FILE-FILE-OPTION PIC S9(9) COMP-5. 49 CLOB-XML-FILE-NAME PIC X(255).
01 DBCLOB-XML-FILE USAGE IS SQL TYPE IS XML AS DBCLOB-FILE.	01 DBCLOB-XML- FILE. 49 DBCLOB-XML-FILE-NAME-LENGTH PIC S9(9) COMP-5 SYNC. 49 DBCLOB-XML-FILE-DATA-LENGTH PIC S9(9) COMP-5. 49 DBCLOB-XML-FILE-FILE-OPTION PIC S9(9) COMP-5. 49 DBCLOB-XML-FILE-NAME PIC X(255).

Notas:

1. Para las variables de host XML AS BLOB o XML AS CLOB que tienen más de 32 767 bytes de longitud, Db2 crea varias declaraciones de lenguaje de host de 32 767 bytes o menos.
2. Para las variables de host XML AS DBCLOB que tienen más de 32 767 caracteres de doble byte de longitud, Db2 crea varias declaraciones de lenguaje de host de 32 767 caracteres de doble byte o menos.

Declaraciones de variables de host XML en PL/I

Las declaraciones que se generan para PL/I difieren, dependiendo de si se utiliza el precompilador Db2 o el coprocesador Db2 .

La siguiente tabla muestra las declaraciones PL/I que el precompilador Db2 genera para algunos tipos XML típicos.

Tabla 93. Ejemplos de declaraciones de variables PL/I

Usted declara esta variable	Db2 el precompilador genera esta variable
DCL BLOB XML SQL TYPE IS XML AS BLOB (1M);	DCL 1 BLOB_XML, 2 BLOB_XML_LENGTH BIN FIXED(31), 2 BLOB_XML_DATA, "1" en la página 577 3 BLOB_XML_DATA1 (32) CHAR(32767), 3 BLOB_XML_DATA2 CHAR(32);
DCL CLOB XML SQL TYPE IS XML AS CLOB (40000K);	DCL 1 CLOB_XML, 2 CLOB_XML_LENGTH BIN FIXED(31), 2 CLOB_XML_DATA, "1" en la página 577 3 CLOB_XML_DATA1 (1250) CHAR(32767), 3 CLOB_XML_DATA2 CHAR(1250);

Tabla 93. Ejemplos de declaraciones de variables PL/I (continuación)

Usted declara esta variable	Db2 el precompilador genera esta variable
DCL DBCLOB_XML SQL TYPE IS XML AS DBCLOB (4000K);	DCL 1 DBCLOB_XML, 2 DBCLOB_XML_LENGTH BIN FIXED(31), 2 DBCLOB_XML_DATA, "2" en la página <u>577</u> 3 DBCLOB_XML_DATA1 (250) GRAPHIC(16383), 3 DBCLOB_XML_DATA2 GRAPHIC(250);
DCL BLOB_XML_FILE SQL TYPE IS XML AS BLOB_FILE;	DCL 1 BLOB_XML_FILE, 2 BLOB_XML_FILE_NAME_LENGTH BIN FIXED(31) ALIGNED, 2 BLOB_XML_FILE_DATA_LENGTH BIN FIXED(31), 2 BLOB_XML_FILE_OPTIONS BIN FIXED(31), 2 BLOB_XML_FILE_NAME CHAR(255);
DCL CLOB_XML_FILE SQL TYPE IS XML AS CLOB_FILE;	DCL 1 CLOB_XML_FILE, 2 CLOB_XML_FILE_NAME_LENGTH BIN FIXED(31) ALIGNED, 2 CLOB_XML_FILE_DATA_LENGTH BIN FIXED(31), 2 CLOB_XML_FILE_OPTIONS BIN FIXED(31), 2 CLOB_XML_FILE_NAME CHAR(255);
DCL DBCLOB_XML_FILE SQL TYPE IS XML AS DBCLOB_FILE;	DCL 1 DBCLOB_XML_FILE, 2 DBCLOB_XML_FILE_NAME_LENGTH BIN FIXED(31) ALIGNED, 2 DBCLOB_XML_FILE_DATA_LENGTH BIN FIXED(31), 2 DBCLOB_XML_FILE_OPTIONS BIN FIXED(31), 2 DBCLOB_XML_FILE_NAME CHAR(255);

Tabla 93. Ejemplos de declaraciones de variables PL/I (continuación)

Usted declara esta variable	Db2 el precompilador genera esta variable
Notas:	
1. Para las variables de host XML AS BLOB o XML AS CLOB que tienen más de 32 767 bytes de longitud, Db2 crea declaraciones de lenguaje de host de la siguiente manera:	
<ul style="list-style-type: none">• Si la longitud del XML es superior a 32 767 bytes y es divisible por 32 767, Db2 crea una matriz de cadenas de 32 767 bytes. La dimensión de la matriz es <i>longitud/32767</i>.• Si la longitud del XML es superior a 32 767 bytes, pero no divisible por 32 767, Db2 crea dos declaraciones: la primera es una matriz de cadenas de 32 767 bytes, donde la dimensión de la matriz, <i>n</i>, es <i>longitud/32 767</i>. El segundo es una cadena de caracteres con <i>una longitud</i> de longitud - <i>n * 32767</i>.	
2. Para las variables de host XML AS DBCLOB que tienen más de 16 383 caracteres de doble byte de longitud, Db2 crea declaraciones de lenguaje de host de la siguiente manera:	
<ul style="list-style-type: none">• Si la longitud del XML es superior a 16 383 caracteres y es divisible por 16 383, Db2 crea una matriz de cadenas de 16 383 caracteres. La dimensión de la matriz es <i>longitud/16383</i>.• Si la longitud del XML es superior a 16 383 caracteres, pero no es divisible por 16 383, Db2 crea dos declaraciones: la primera es una matriz de cadenas de 16 383 bytes, donde la dimensión de la matriz, <i>m</i>, es <i>longitud/16 383</i>. El segundo es una cadena de caracteres con <i>una longitud</i> de longitud - <i>m * 16383</i>.	

Conceptos relacionados

[Inserción de filas con valores de columna XML \(Db2 Programming for XML\)](#)

[Recuperación de datos XML \(Db2 Programming for XML\)](#)

[Actualizaciones de columnas XML \(Db2 Programming for XML\)](#)

Actualizaciones de columnas XML en aplicaciones SQL incrustadas

Cuando actualice o inserte datos en columnas XML de una tabla de datos de Db2 , los datos de entrada deben estar en formato XML textual.

La codificación de los datos XML se puede obtener a partir de los propios datos, lo cual se conoce como datos *codificados internamente*, o a partir de fuentes externas, lo cual se conoce como datos *codificados externamente*. Los datos XML que se envían al servidor de bases de datos como datos binarios se tratan como datos codificados internamente. Los datos XML que se envían al servidor de la base de datos como datos de caracteres se tratan como datos codificados externamente.

Los datos codificados externamente pueden tener una codificación interna. Es decir, los datos pueden enviarse al servidor de la base de datos como datos de caracteres, pero contienen información de codificación. Db2 no exige la coherencia de la codificación interna y externa. Cuando la información de codificación interna y externa difiere, la codificación externa tiene prioridad. Sin embargo, si existe una diferencia entre la codificación externa e interna, es posible que se haya producido una conversión de caracteres intermedia en los datos y que se haya producido una pérdida de datos.

Los datos de caracteres en columnas XML se almacenan en una codificación UTF-8. El servidor de la base de datos se encarga de convertir los datos de su codificación interna o externa a UTF-8.

Los siguientes ejemplos muestran cómo actualizar columnas XML en aplicaciones ensambladoras, C, COBOL y PL/I. Los ejemplos utilizan una tabla llamada MYCUSTOMER, que es una copia de la tabla CUSTOMER de muestra.

Ejemplo

El siguiente ejemplo muestra un programa ensamblador que inserta datos de XML AS BLOB, XML AS CLOB y variables de host CLOB en una columna XML. Los datos XML AS BLOB se insertan como datos binarios, por lo que el servidor de la base de datos respeta la codificación interna. Los datos XML AS CLOB y CLOB se insertan como datos de caracteres, por lo que el servidor de la base de datos respeta la codificación externa.

```
*****
* UPDATE AN XML COLUMN WITH DATA IN AN XML AS CLOB HOST VARIABLE      *
*****
EXEC SQL
    UPDATE MYCUSTOMER
        SET INFO = :XMLBUF
        WHERE CID = 1000
*****
* UPDATE AN XML COLUMN WITH DATA IN AN XML AS BLOB HOST VARIABLE      *
*****
EXEC SQL
    UPDATE MYCUSTOMER
        SET INFO = :XMLBLOB
        WHERE CID = 1000
*****
* UPDATE AN XML COLUMN WITH DATA IN A CLOB HOST VARIABLE. USE          *
* THE XMLPARSE FUNCTION TO CONVERT THE DATA TO THE XML TYPE.           *
*****
EXEC SQL
    UPDATE MYCUSTOMER
        SET INFO = XMLPARSE(DOCUMENT :CLOBBUF)
        WHERE CID = 1000
...
LTORG
*****
* HOST VARIABLE DECLARATIONS *
*****
XMLBUF   SQL TYPE IS XML AS CLOB 10K
XMLBLOB  SQL TYPE IS XML AS BLOB 10K
CLOBBUF  SQL TYPE IS CLOB 10K
```

Ejemplo

El siguiente ejemplo muestra un programa en lenguaje C que inserta datos de XML AS BLOB, XML AS CLOB y variables de host CLOB en una columna XML. Los datos XML AS BLOB se insertan como datos binarios, por lo que el servidor de la base de datos respeta la codificación interna. Los datos XML AS CLOB y CLOB se insertan como datos de caracteres, por lo que el servidor de la base de datos respeta la codificación externa.

```
/*
* Host variable declarations */
EXEC SQL BEGIN DECLARE SECTION;
SQL TYPE IS XML AS CLOB( 10K ) xmlBuf;
SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
SQL TYPE IS CLOB( 10K ) clobBuf;
EXEC SQL END DECLARE SECTION;
/*
* Update an XML column with data in an XML AS CLOB host variable */
EXEC SQL UPDATE MYCUSTOMER SET INFO = :xmlBuf where CID = 1000;
/*
* Update an XML column with data in an XML AS BLOB host variable */
EXEC SQL UPDATE MYCUSTOMER SET INFO = :xmlblob where CID = 1000;
/*
* Update an XML column with data in a CLOB host variable. Use      */
/* the XMLPARSE function to convert the data to the XML type.      */
EXEC SQL UPDATE MYCUSTOMER SET INFO = XMLPARSE(DOCUMENT :clobBuf) where CID = 1000;
```

Ejemplo

El siguiente ejemplo muestra un programa COBOL que inserta datos de XML AS BLOB, XML AS CLOB y variables de host CLOB en una columna XML. Los datos XML AS BLOB se insertan como datos binarios, por lo que el servidor de la base de datos respeta la codificación interna. Los datos XML AS CLOB y CLOB se insertan como datos de caracteres, por lo que el servidor de la base de datos respeta la codificación externa.

```
*****
* Host variable declarations *
*****
01 XMLBUF USAGE IS SQL TYPE IS XML as CLOB(10K).
01 XMLBLOB  USAGE IS SQL TYPE IS XML AS BLOB(10K).
01 CLOBBUF  USAGE IS SQL TYPE IS CLOB(10K).
*****
* Update an XML column with data in an XML AS CLOB host variable *
```

```
*****
EXEC SQL UPDATE MYCUSTOMER SET INFO = :XMLBUF where CID = 1000.
*****
* Update an XML column with data in an XML AS BLOB host variable *
*****
EXEC SQL UPDATE MYCUSTOMER SET INFO = :XMLBLOB where CID = 1000.
*****
* Update an XML column with data in a CLOB host variable. Use      *
* the XMLPARSE function to convert the data to the XML type.      *
*****
EXEC SQL UPDATE MYCUSTOMER SET INFO = XMLPARSE(DOCUMENT :CLOBBUF) where CID = 1000.
```

Ejemplo

El siguiente ejemplo muestra un programa PL/I que inserta datos de XML AS BLOB, XML AS CLOB y variables de host CLOB en una columna XML. Los datos XML AS BLOB se insertan como datos binarios, por lo que el servidor de la base de datos respeta la codificación interna. Los datos XML AS CLOB y CLOB se insertan como datos de caracteres, por lo que el servidor de la base de datos respeta la codificación externa.

```
/*
/* Host variable declarations */
*/
DCL
  XMLBUF SQL TYPE IS XML AS CLOB(10K),
  XMLBLOB SQL TYPE IS XML AS BLOB(10K),
  CLOBBUF SQL TYPE IS CLOB(10K);
/*
* Update an XML column with data in an XML AS CLOB host variable */
/*
EXEC SQL UPDATE MYCUSTOMER SET INFO = :XMLBUF where CID = 1000;
/*
* Update an XML column with data in an XML AS BLOB host variable */
/*
EXEC SQL UPDATE MYCUSTOMER SET INFO = :XMLBLOB where CID = 1000;
/*
Update an XML column with data in a CLOB host variable. Use      */
/* the XMLPARSE function to convert the data to the XML type.      */
/*
EXEC SQL UPDATE MYCUSTOMER SET INFO = XMLPARSE(DOCUMENT :CLOBBUF) where CID = 1000;
```

Conceptos relacionados

[Inserción de filas con valores de columna XML \(Db2 Programming for XML\)](#)

[Actualizaciones de columnas XML \(Db2 Programming for XML\)](#)

Recuperación de datos XML en aplicaciones SQL incrustadas

En una aplicación SQL incrustada, si recupera los datos en una variable de host de caracteres, Db2 convierte los datos del esquema de codificación de UTF-8 al esquema de codificación de la aplicación. Si recupera los datos en una variable de host binaria, Db2 no convierte los datos a otro esquema de codificación.

Los datos de salida están en formato XML textual.

Db2 podría añadir una especificación de codificación XML a los datos recuperados, dependiendo de si llama a la función XMLSERIALIZE cuando recupera los datos. Si no llama a la función XMLSERIALIZE, Db2 añade la especificación de codificación XML correcta a los datos recuperados. Si llama a la función XMLSERIALIZE, Db2 añade una declaración de codificación XML interna para la codificación UTF-8, si especifica INCLUDING XMLDECLARATION en la llamada a la función. Cuando utilice INCLUDING XMLDECLARATION, debe asegurarse de que los datos recuperados no se conviertan de una codificación UTF-8 a otra.

Los siguientes ejemplos muestran cómo recuperar datos de columnas XML en aplicaciones ensambladoras, C, COBOL y PL/I. Los ejemplos utilizan una tabla llamada MYCUSTOMER, que es una copia de la tabla CUSTOMER de muestra.

Ejemplo : El siguiente ejemplo muestra un programa ensamblador que recupera datos de una columna XML en variables de host XML AS BLOB, XML AS CLOB y CLOB. Los datos que se recuperan en una variable de host XML AS BLOB se recuperan como datos binarios, por lo que el servidor de la base de datos genera una declaración XML con codificación UTF-8. Los datos que se recuperan en una variable

de host XML AS CLOB se recuperan como datos de caracteres, por lo que el servidor de la base de datos genera una declaración XML con una declaración de codificación interna que es coherente con la codificación externa. Los datos que se recuperan en una variable de host CLOB se recuperan como datos de caracteres, por lo que el servidor de la base de datos genera una declaración XML con una declaración de codificación interna. Esa declaración podría no ser coherente con la codificación externa.

```
*****
* RETRIEVE XML COLUMN DATA INTO AN XML AS CLOB HOST VARIABLE      *
*****
EXEC SQL
  SELECT INFO
  INTO :XMLBUF
  FROM MYCUSTOMER
  WHERE CID = 1000
*****
* RETRIEVE XML COLUMN DATA INTO AN XML AS BLOB HOST VARIABLE      *
*****
EXEC SQL
  SELECT INFO
  INTO :XMLBLOB
  FROM MYCUSTOMER
  WHERE CID = 1000
*****
* RETRIEVE DATA FROM AN XML COLUMN INTO A CLOB HOST VARIABLE.      *
* BEFORE SENDING THE DATA TO THE APPLICATION, INVOKE THE          *
* XMLSERIALIZE FUNCTION TO CONVERT THE DATA FROM THE XML          *
* TYPE TO THE CLOB TYPE.                                           *
*****
EXEC SQL
  SELECT XMLSERIALIZE(INFO AS CLOB(10K))
  INTO :CLOBBUF
  FROM MYCUSTOMER
  WHERE CID = 1000
...
LTORG
*****
* HOST VARIABLE DECLARATIONS *
*****
XMLBUF  SQL TYPE IS XML AS CLOB 10K
XMLBLOB SQL TYPE IS XML AS BLOB 10K
CLOBBUF SQL TYPE IS CLOB 10K
```

Ejemplo : El siguiente ejemplo muestra un programa en lenguaje C que recupera datos de una columna XML en variables de host XML AS BLOB, XML AS CLOB y CLOB. Los datos que se recuperan en una variable de host XML AS BLOB se recuperan como datos binarios, por lo que el servidor de la base de datos genera una declaración XML con codificación UTF-8. Los datos que se recuperan en una variable de host XML AS CLOB se recuperan como datos de caracteres, por lo que el servidor de la base de datos genera una declaración XML con una declaración de codificación interna que es coherente con la codificación externa. Los datos que se recuperan en una variable de host CLOB se recuperan como datos de caracteres, por lo que el servidor de la base de datos genera una declaración XML con una declaración de codificación interna. Esa declaración podría no ser coherente con la codificación externa.

```
*****
/* Host variable declarations */
*****
EXEC SQL BEGIN DECLARE SECTION;
SQL TYPE IS XML AS CLOB( 10K ) xmlBuf;
SQL TYPE IS XML AS BLOB( 10K ) xmlBlob;
SQL TYPE IS CLOB( 10K ) clobBuf;
EXEC SQL END DECLARE SECTION;
*****
/* Retrieve data from an XML column into an XML AS CLOB host variable */
*****
EXEC SQL SELECT INFO  INTO :xmlBuf from myTable where CID = 1000;
*****
/* Retrieve data from an XML column into an XML AS BLOB host variable */
*****
EXEC SQL SELECT INFO  INTO :xmlBlob from myTable where CID = 1000;
*****
/* RETRIEVE DATA FROM AN XML COLUMN INTO A CLOB HOST VARIABLE.      */
/* BEFORE SENDING THE DATA TO THE APPLICATION, INVOKE THE          */
/* XMLSERIALIZE FUNCTION TO CONVERT THE DATA FROM THE XML          */
/* TYPE TO THE CLOB TYPE.                                           */
*****
```

```
EXEC SQL SELECT XMLSERIALIZE(INFO AS CLOB(10K))
  INTO :clobBuf from myTable where CID = 1000;
```

Ejemplo : El siguiente ejemplo muestra un programa COBOL que recupera datos de una columna XML en variables de host XML AS BLOB, XML AS CLOB y CLOB. Los datos que se recuperan en una variable de host XML AS BLOB se recuperan como datos binarios, por lo que el servidor de la base de datos genera una declaración XML con codificación UTF-8. Los datos que se recuperan en una variable de host XML AS CLOB se recuperan como datos de caracteres, por lo que el servidor de la base de datos genera una declaración XML con una declaración de codificación interna que es coherente con la codificación externa. Los datos que se recuperan en una variable de host CLOB se recuperan como datos de caracteres, por lo que el servidor de la base de datos genera una declaración XML con una declaración de codificación interna. Esa declaración podría no ser coherente con la codificación externa.

```
*****
* Host variable declarations *
*****
01 XMLBUF  USAGE IS SQL TYPE IS XML AS CLOB(10K).
01 XMLBLOB USAGE IS SQL TYPE IS XML AS BLOB(10K).
01 CLOBBUF USAGE IS SQL TYPE IS CLOB(10K).
*****
* Retrieve data from an XML column into an XML AS CLOB host variable *
*****
EXEC SQL SELECT INFO
  INTO :XMLBUF
  FROM MYTABLE
  WHERE CID = 1000
END-EXEC.
*****
* Retrieve data from an XML column into an XML AS BLOB host variable *
*****
EXEC SQL SELECT INFO
  INTO :XMLBLOB
  FROM MYTABLE
  WHERE CID = 1000
END-EXEC.
*****
* RETRIEVE DATA FROM AN XML COLUMN INTO A CLOB HOST VARIABLE.      *
* BEFORE SENDING THE DATA TO THE APPLICATION, INVOKE THE          *
* XMLSERIALIZE FUNCTION TO CONVERT THE DATA FROM THE XML          *
* TYPE TO THE CLOB TYPE.                                         *
*****
EXEC SQL SELECT XMLSERIALIZE(INFO AS CLOB(10K))
  INTO :CLOBBUF
  FROM MYTABLE
  WHERE CID = 1000
END-EXEC.
```

Ejemplo : El siguiente ejemplo muestra un programa PL/I que recupera datos de una columna XML en variables de host XML AS BLOB, XML AS CLOB y CLOB. Los datos que se recuperan en una variable de host XML AS BLOB se recuperan como datos binarios, por lo que el servidor de la base de datos genera una declaración XML con codificación UTF-8. Los datos que se recuperan en una variable de host XML AS CLOB se recuperan como datos de caracteres, por lo que el servidor de la base de datos genera una declaración XML con una declaración de codificación interna que es coherente con la codificación externa. Los datos que se recuperan en una variable de host CLOB se recuperan como datos de caracteres, por lo que el servidor de la base de datos genera una declaración XML con una declaración de codificación interna. Esa declaración podría no ser coherente con la codificación externa.

```
/*
* Host variable declarations */
/
DCL
  XMLBUF SQL TYPE IS XML AS CLOB(10K),
  XMLBLOB SQL TYPE IS XML AS BLOB(10K),
  CLOBBUF SQL TYPE IS CLOB(10K);
/*
* Retrieve data from an XML column into an XML AS CLOB host variable */
/*
* Retrieve data from an XML column into an XML AS BLOB host variable */
EXEC SQL SELECT INFO  INTO :XMLBUF FROM MYTABLE WHERE CID = 1000;
/*
* Retrieve data from an XML column into an XML AS CLOB host variable */
EXEC SQL SELECT INFO  INTO :XMLBLOB FROM MYTABLE WHERE CID = 1000;
/*
* Retrieve data from an XML column into an XML AS BLOB host variable */
```

```

/*
 * RETRIEVE DATA FROM AN XML COLUMN INTO A CLOB HOST VARIABLE.
 */
/* BEFORE SENDING THE DATA TO THE APPLICATION, INVOKE THE
 */
/* XMLSERIALIZE FUNCTION TO CONVERT THE DATA FROM THE XML
 */
/* TYPE TO THE CLOB TYPE.
 */
***** EXEC SQL SELECT XMLSERIALIZE(INFO AS CLOB(10K))
      INTO :CLOBBUF FROM MYTABLE WHERE CID = 1000;

```

Recuperación de datos XML (Db2 Programming for XML)

Programas de ejemplo que llaman a procedimientos almacenados

Los ejemplos se pueden utilizar como modelos cuando escribe aplicaciones que llaman a procedimientos almacenados. Asimismo, *prefijo.SDSNSAMP* contiene trabajos de ejemplo de DSNTEJ6P y DSNTEJ6S y programas DSN8EP1 y DSN8EP2 de ejemplo, que puede ejecutar.

Conceptos relacionados

[Ejemplos de aplicaciones suministrados con Db2 for z/OS](#)

Db2 proporciona ejemplos de aplicaciones para ayudarle con las técnicas de programación e Db2 es y las prácticas de codificación dentro de cada uno de los cuatro entornos: batch, TSO, IMS, y CICS. Las aplicaciones de ejemplo contienen varias aplicaciones que pueden aplicarse a la gestión de una empresa.

Aplicaciones ensambladoras que emiten sentencias SQL

Puede codificar sentencias SQL en programas ensambladores siempre que pueda utilizar sentencias ejecutables.

Cada instrucción SQL en un programa ensamblador debe comenzar con EXEC SQL. Las palabras clave EXEC y SQL deben aparecer en una línea, pero el resto de la instrucción puede aparecer en líneas posteriores.

Podría codificar una instrucción UPDATE en un programa ensamblador de la siguiente manera:

<pre> EXEC SQL UPDATE DSN8C10.DEPT SET MGRNO = :MGRNUM WHERE DEPTNO = :INTDEPT </pre>	X X
---	--------

Comentarios

No puede incluir comentarios de ensamblador en las sentencias SQL. Sin embargo, puede incluir comentarios SQL en cualquier instrucción SQL incrustada. Para obtener más información, consulte [Comentarios SQL \(Db2 SQL\)](#).

Continuación para instrucciones SQL

Las reglas de continuación de línea para las sentencias SQL son las mismas que para las sentencias ensambladoras, excepto que debe especificar EXEC SQL dentro de una línea. Cualquier parte de la declaración que no quepa en una línea puede aparecer en las líneas siguientes, comenzando en el margen de continuación (columna 16, por defecto). Cada línea del estado de cuenta, excepto la última, debe tener un carácter de continuación (un carácter que no esté en blanco) inmediatamente después del margen derecho en la columna 72.

Delimitadores para instrucciones SQL

Delimita una instrucción SQL en su programa ensamblador con la palabra clave de inicio " EXEC SQL " y un final de línea o final de la última línea continuada.

Declaración de tablas y vistas

Su programa ensamblador debe incluir una instrucción DECLARE para describir cada tabla y ver los accesos del programa.

Incluir código

Para incluir sentencias SQL o sentencias de declaración de variables de host ensamblador de un miembro de un conjunto de datos particionado, coloque la siguiente sentencia SQL en el código fuente donde desee incluir las sentencias:

```

EXEC SQL INCLUDE member-name

```

No se pueden anidar sentencias SQL INCLUDE.

Márgenes

Utilice la opción del precompilador MARGINS para establecer un margen izquierdo, un margen derecho y un margen de continuación. Los valores predeterminados para estos márgenes son las columnas 1, 71 y 16, respectivamente. Si EXEC SQL comienza antes del margen izquierdo especificado, el precompilador de instrucciones SQL (Db2) no reconoce la instrucción SQL. Si utiliza los márgenes predeterminados, puede colocar una instrucción SQL en cualquier lugar entre las columnas 2 y 71.

Sentencia FETCH de varias filas

Solo puede utilizar el FETCH... USO DEL DESCRIPTOR forma de la sentencia FETCH de varias filas en un programa ensamblador. El precompilador de ensamblador (Db2) no reconoce las declaraciones de matrices de variables de host para un programa ensamblador.

Nombres

Puede utilizar cualquier nombre de ensamblador válido para una variable de host. Sin embargo, no utilice nombres de entrada externos o nombres de planes de acceso que empiecen por «DSN» ni nombres de variables de host que empiecen por «SQL». Estos nombres están reservados para Db2.

El primer carácter de una variable de host que se utiliza en SQL incrustado no puede ser un guión bajo. Sin embargo, puede utilizar un guión bajo como primer carácter de un símbolo que no se utilice en SQL incrustado.

Etiquetas de extracto

Puede anteponer una etiqueta a una instrucción SQL. La primera línea de una instrucción SQL puede utilizar una etiqueta que comience en el margen izquierdo (columna 1). Si no utiliza una etiqueta, deje la columna 1 en blanco.

WHENEVER, sentencia

El objetivo de la cláusula GOTO en una instrucción SQL WHENEVER debe ser una etiqueta en el código fuente del ensamblador y debe estar dentro del ámbito de las instrucciones SQL a las que WHENEVER afecta.

Consideraciones especiales para el montador

Las siguientes consideraciones se aplican a los programas escritos en ensamblador:

- Para permitir programas reentrantes, el precompilador coloca todas las variables y estructuras que genera dentro de un DSECT llamado SQLDSECT, y genera un símbolo ensamblador llamado SQLDLEN. SQLDLEN contiene la longitud del DSECT. Su programa debe asignar un área del tamaño indicado por SQLDLEN, inicializarla y proporcionarle direccionabilidad como DSECT SQLDSECT. El precompilador no genera código para asignar el almacenamiento para SQLDSECT; el programa de aplicación debe asignar el almacenamiento.

CICS : A continuación se muestra un ejemplo de código para admitir programas reentrantes, que se ejecutan bajo CICS, sigue:

```
DFHEISTG DSECT
    DFHEISTG
        EXEC SQL INCLUDE SQLCA
*
    DS      OF
SQDWSREG EQU   R7
SQDWSTOR DS    (SQLDLEN)C  RESERVE STORAGE TO BE USED FOR SQLDSECT
:
XXPROGRM DFHEIENT CODEREG=R12,EIBREG=R11,DATAREG=R13
*
*
*  SQL WORKING STORAGE
    LA      SQDWSREG,SQDWSTOR      GET ADDRESS OF SQLDSECT
    USING  SQLDSECT,SQDWSREG      AND TELL ASSEMBLER ABOUT IT
*
```

En este ejemplo, la asignación de almacenamiento real la realiza la macro DFHEIENT.

TSO : El programa de muestra en *prefijo.SDSNSAMP(DSNTIAD)* contiene un ejemplo de cómo adquirir almacenamiento para SQLDSECT en un programa que se ejecuta en un entorno TSO. El siguiente código de ejemplo contiene fragmentos de *prefijo.SDSNSAMP(DSNTIAD)* con explicaciones en los comentarios.

```

DSNTIAD CSECT           CONTROL SECTION NAME
        SAVE (14,12)      ANY SAVE SEQUENCE
        LR   R12,R15      CODE ADDRESSABILITY
        USING DSNTIAD,R12 TELL THE ASSEMBLER
        LR   R7,R1       SAVE THE PARM POINTER
*
* Allocate storage of size PRGSIZ1+SQLDSIZ, where:
* - PRGSIZ1 is the size of the DSNTIAD program area
* - SQLDSIZ is the size of the SQLDSECT, and declared
*   when the DB2 precompiler includes the SQLDSECT
*
        L     R6,PRGSIZ1    GET SPACE FOR USER PROGRAM
        A     R6,SQLDSIZ    GET SPACE FOR SQLDSECT
        GETMAIN R,LV=(6)   GET STORAGE FOR PROGRAM VARIABLES
        LR   R10,R1       POINT TO IT
*
* Initialize the storage
*
        LR   R2,R10        POINT TO THE FIELD
        LR   R3,R6        GET ITS LENGTH
        SR   R4,R4        CLEAR THE INPUT ADDRESS
        SR   R5,R5        CLEAR THE INPUT LENGTH
        MVCL R2,R4       CLEAR OUT THE FIELD
*
* Map the storage for DSNTIAD program area
*
        ST   R13,FOUR(R10)  CHAIN THE SAVEAREA PTRS
        ST   R10,EIGHT(R13) CHAIN SAVEAREA FORWARD
        LR   R13,R10       POINT TO THE SAVEAREA
        USING PRGAREA1,R13 SET ADDRESSABILITY
*
* Map the storage for the SQLDSECT
*
        LR   R9,R13        POINT TO THE PROGAREA
        A     R9,PRGSIZ1    THEN PAST TO THE SQLDSECT
        USING SQLDSECT,R9  SET ADDRESSABILITY
...
        LTORG
*****
*      DECLARE VARIABLES, WORK AREAS
*
*****WORKING STORAGE FOR THE PROGRAM
PRGAREA1 DSECT
...
        DS   0D
PRGSIZE1 EQU  *-PRGAREA1      DYNAMIC WORKAREA SIZE
...
DSNTIAD CSECT           RETURN TO CSECT FOR CONSTANT
PRGSIZ1 DC   A(PRGSIZE1)  SIZE OF PROGRAM WORKING STORAGE
CA   DSECT
      EXEC SQL INCLUDE SQLCA
...

```

- Db2 no procesa símbolos establecidos en sentencias SQL.
- El código generado puede incluir más de dos continuaciones por comentario.
- El código generado utiliza constantes literales (por ejemplo, =F'-84'), por lo que podría ser necesaria una instrucción LTORG.
- El código generado utiliza los registros 0, 1, 14 y 15. Registrar 13 puntos en un área de guardado que utiliza el programa llamado. El registro 15 no contiene un código de retorno después de una llamada generada por una instrucción SQL.

CICS : Un CICS programa de aplicación utiliza la macro DFHEIENT para generar el código del punto de entrada. Al utilizar esta macro, tenga en cuenta lo siguiente:

- Si utiliza el DATAREG predeterminado en la macro DFHEIENT, registre 13 puntos en el área de guardado.

- Si utiliza cualquier otro DATAREG en la macro DFHEIENT, debe proporcionar direccionabilidad a un área segura.

Por ejemplo, para usar SAVED, puede codificar instrucciones para guardar, cargar y restaurar el registro 13 alrededor de cada instrucción SQL como en el siguiente ejemplo.

```
ST    13,SAVER13      SAVE REGISTER 13
LA    13,SAVED        POINT TO SAVE AREA
EXEC  SQL . . .
L     13,SAVER13      RESTORE REGISTER 13
```

- Si tiene un error de direccionamiento en el código generado por el precompilador debido a variables de host de entrada o salida en una instrucción SQL, compruebe que tiene suficientes registros base.
- No ponga CICS opciones de traducción en el código fuente del ensamblado. En su lugar, pase las opciones al traductor utilizando el campo PARM.

Manejo de códigos de error de SQL

Las aplicaciones ensambladoras pueden solicitar más información sobre errores SQL en Db2.

Para obtener más información, consulte “[Gestión de códigos de error de SQL en aplicaciones de ensamblador](#)” en la página 602.

Tareas relacionadas

[Descripción general de las aplicaciones de programación que acceden a datos de Db2 for z/OS](#)

Las aplicaciones que interactúan con Db2, en primer lugar se deben conectar a Db2. Entonces pueden leer, añadir o modificar datos o manipular objetos de Db2.

[Inclusión de SQL dinámico en el programa](#)

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

[Establecimiento de límites para el uso de recursos de sistema utilizando el recurso de límite de recursos \(Db2 Performance\)](#)

Ejemplos de programación de ensamblador

Puede escribir programas de Db2 en ensamblador. Estos programas pueden acceder a un subsistema Db2 local o remoto y pueden ejecutar sentencias de SQL estático o dinámico. Esta información contiene varios ejemplos de programación de ese tipo.

Para preparar y ejecutar estas aplicaciones, utilice el JCL en *el prefijo.SDSNSAMP* como modelo para su JCL.

Referencia relacionada

[Entornos y lenguajes de aplicación para las aplicaciones de ejemplo](#)

Las aplicaciones de muestra demuestran cómo ejecutar aplicaciones de Db2 en el TSO, IMS, o CICS.

Definición del área de comunicación SQL, SQLSTATE y SQLCODE en el ensamblador

Los programas de ensamblador que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Acerca de esta tarea

Si especifica la opción de procesamiento SQL STDSQL(YES), no defina un SQLCA. Si lo hace, Db2 ignora su SQLCA y la definición de su SQLCA provoca errores en tiempo de compilación. Si especifica la opción de procesamiento SQL STDSQL(NO), incluya un SQLCA explícitamente.

Si su aplicación contiene instrucciones SQL y no incluye un área de comunicaciones SQL (SQLCA), debe declarar variables de host SQLCODE y SQLSTATE individuales. Su programa puede utilizar estas variables para comprobar si una instrucción SQL se ha ejecutado correctamente.

Procedimiento

Elija una de estas acciones:

Opción	Descripción
Para definir el área de comunicaciones SQL:	<p>a. Codifique el SQLCA directamente en el programa o utilice la siguiente instrucción SQL INCLUDE para solicitar una declaración SQLCA estándar:</p> <pre>EXEC SQL INCLUDE SQLCA</pre> <p>Si su programa es reentrant, debe incluir el SQLCA dentro de un área de datos única que se adquiere para su tarea (un DSECT). Por ejemplo, al comienzo de su programa, especifique el siguiente código:</p> <pre>PROGAREA DSECT EXEC SQL INCLUDE SQLCA</pre> <p>Como alternativa, puede crear un área de almacenamiento independiente para el SQLCA y proporcionar direccionabilidad a esa área.</p> <p>Db2 establece los valores SQLCODE y SQLSTATE en el SQLCA después de que se ejecute cada instrucción SQL. Su aplicación debe comprobar estos valores para determinar si la última instrucción SQL se ha realizado correctamente.</p>
Para declarar variables de host SQLCODE y SQLSTATE:	<p>a. Declare la variable SQLCODE dentro de una declaración BEGIN DECLARE SECTION y una declaración END DECLARE SECTION en sus declaraciones de programa como un entero de palabra completa.</p> <p>b. Declare la variable SQLSTATE dentro de una instrucción BEGIN DECLARE SECTION y una instrucción END DECLARE SECTION en las declaraciones de su programa como una cadena de caracteres de longitud 5 (CL5).</p> <p>Restricción: No declare una variable SQLSTATE como un elemento de una estructura.</p> <p>Requisito: Despues de declarar las variables SQLCODE y SQLSTATE, asegúrese de que todas las sentencias SQL del programa estén dentro del ámbito de la declaración de estas variables.</p>

Tareas relacionadas

Comprobación de la ejecución de sentencias SQL

Después de ejecutar una sentencia SQL, el programa debe comprobar si hay errores antes de confirmar los datos y manejar los errores que representan.

Comprobación de la ejecución de sentencias SQL utilizando SQLCA

Un modo de comprobar si una sentencia SQL se ha ejecutado correctamente es utilizar el área de comunicación SQL (SQLCA). Esta área se distingue de la comunicación con Db2.

Comprobación de la ejecución de sentencias SQL utilizando SQLCODE y SQLSTATE

Siempre que se ejecuta una sentencia SQL, los campos SQLCODE y SQLSTATE de SQLCA reciben un código de retorno.

Definición de elementos que su programa puede utilizar para comprobar si una sentencia SQL se ha ejecutado correctamente

Si su programa contiene sentencias SQL, el programa debe definir determinada infraestructura para poder comprobar si las sentencias se han ejecutado correctamente. También puede incluir un área de comunicación SQL (SQLCA), que contenga variables SQLCODE y SQLSTATE, o declarar variables host SQLCODE y SQLSTATE.

Definición de áreas de descriptor de SQL (SQLDA) en ensamblador

Si su programa incluye determinadas sentencias SQL, debe definir al menos un área de descriptor SQL (SQLDA). Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las

sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Procedimiento

Codifique el SQLDA directamente en el programa, o utilice la siguiente instrucción SQL INCLUDE para solicitar una declaración SQLDA estándar:

```
EXEC SQL INCLUDE SQLDA
```

Restricción: Debe colocar las declaraciones SQLDA antes de la primera instrucción SQL que haga referencia al descriptor de datos, a menos que utilice la opción de procesamiento TWOPASS SQL.

Ejemplo

Puede utilizar matrices de variables de host para ciertas operaciones de varias filas en otros lenguajes de host, como C, C++, COBOL y PL/I, pero el precompilador de ensamblador (Db2) no reconoce las declaraciones de matrices de variables de host. Sin embargo, puede utilizar declaraciones SQLDA para lograr resultados similares en programas ensambladores, como se muestra en los siguientes ejemplos:

- La compatibilidad del ensamblador con FETCH de varias filas se limita a la sentencia FETCH con la cláusula INTO DESCRIPTOR. Por ejemplo:

```
EXEC SQL FETCH NEXT ROWSET FROM C1 FOR 10 ROWS          X  
      INTO DESCRIPTOR :SQLDA
```

- La compatibilidad del ensamblador para INSERT de varias filas se limita a los siguientes casos:

- Sentencia INSERT estática de varias filas con valores escalares (variables de host escalares o expresiones escalares) en la cláusula VALUES. Por ejemplo:

```
EXEC SQL INSERT INTO T1 VALUES (1, CURRENT DATE, 'TEST')          X  
      FOR 10 ROWS
```

- INSERT dinámico de varias filas ejecutado con la cláusula USING DESCRIPTOR en la sentencia EXECUTE. Por ejemplo:

```
ATR      DS      CL20          ATTRIBUTES FOR PREPARE  
S1       DS      H,CL30          VARCHAR STATEMENT STRING  
MVC      ATR(20),=C'FOR MULTIPLE ROWS'  
MVC      S1(2),=H'25'  
MVC      S1+2(30),=C'INSERT INTO T1 VALUES (?) '  
EXEC    SQL PREPARE STMT ATTRIBUTES :ATR FROM :S1  
EXEC    SQL EXECUTE STMT USING DESCRIPTOR :SQLDA FOR 10 ROWS  
where the descriptor is set up correctly in advance according to the  
specifications for dynamic execution of a multiple-row INSERT statement  
with a descriptor
```

- El ensamblador no admite la combinación de varias filas. No se pueden especificar instrucciones MERGE que hagan referencia a matrices de variables de host.

Tareas relacionadas

[Definición de áreas de descriptor de SQL \(SQLDA\)](#)

Si su programa incluye ciertas sentencias SQL, debe definir al menos *un área de descriptor SQL (SQLDA)*. Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Referencia relacionada

[Área de descriptor de SQL \(SQLDA\) \(Db2 SQL\)](#)

Declaración de variables de host y variables de indicador en ensamblador

Puede utilizar variables de host, matrices de variables de host y estructuras de host en instrucciones SQL en su programa para pasar datos entre Db2 y su aplicación.

Procedimiento

Para declarar variables de host, matrices de variables de host y estructuras de host:

1. Declare las variables de acuerdo con las siguientes reglas y directrices:

- Puede declarar variables de host en estilo ensamblador normal (DC o DS), dependiendo del tipo de datos y de las limitaciones de ese tipo de datos. Puede especificar un valor en las declaraciones DC o DS (por ejemplo, DC H '5'). El precompilador de C++ (Db2) examina solo las declaraciones decimales empaquetadas.
- Si especifica la opción de procesamiento ONEPASS SQL, debe declarar explícitamente cada variable de host y cada matriz de variables de host antes de utilizarlas en una instrucción SQL. Si especifica la opción del precompilador TWOPASS, debe declarar cada variable de host antes de usarla en la instrucción DECLARE CURSOR.
- Si especifica la opción de procesamiento SQL STDSQL(YES), debe preceder las sentencias del lenguaje del host que definen las variables del host y las matrices de variables del host con la sentencia BEGIN DECLARE SECTION y seguir las sentencias del lenguaje del host con la sentencia END DECLARE SECTION. De lo contrario, estas declaraciones son opcionales.
- Asegúrese de que cualquier instrucción SQL que utilice una variable de host o una matriz de variables de host esté dentro del ámbito de la instrucción que declara esa variable o matriz.
- Si utiliza el lenguaje de programación C (Db2 precompiler), asegúrese de que los nombres de las variables de host y de las matrices de variables de host sean únicos dentro del programa, incluso si las variables y las matrices de variables se encuentran en bloques, clases, procedimientos, funciones o subrutinas diferentes. Puede calificar los nombres con un nombre de estructura para hacerlos únicos.

2. Opcional: Definir cualquier variable, matriz y estructura de indicadores asociados.

Tareas relacionadas

[Declaración de variables host y variables de indicación](#)

Puede utilizar variables host y variables de indicación en sentencias SQL del programa para pasar datos entre Db2 y la aplicación.

Variables de host en Assembler

En programas ensamblador, puede especificar variables host numéricas, de caracteres, gráficas, binarias, LOB, XML y ROWID. También puede especificar conjuntos de resultados, tabla y localizadores LOB y variables de referencia de archivos LOB y XML.

Restricciones:

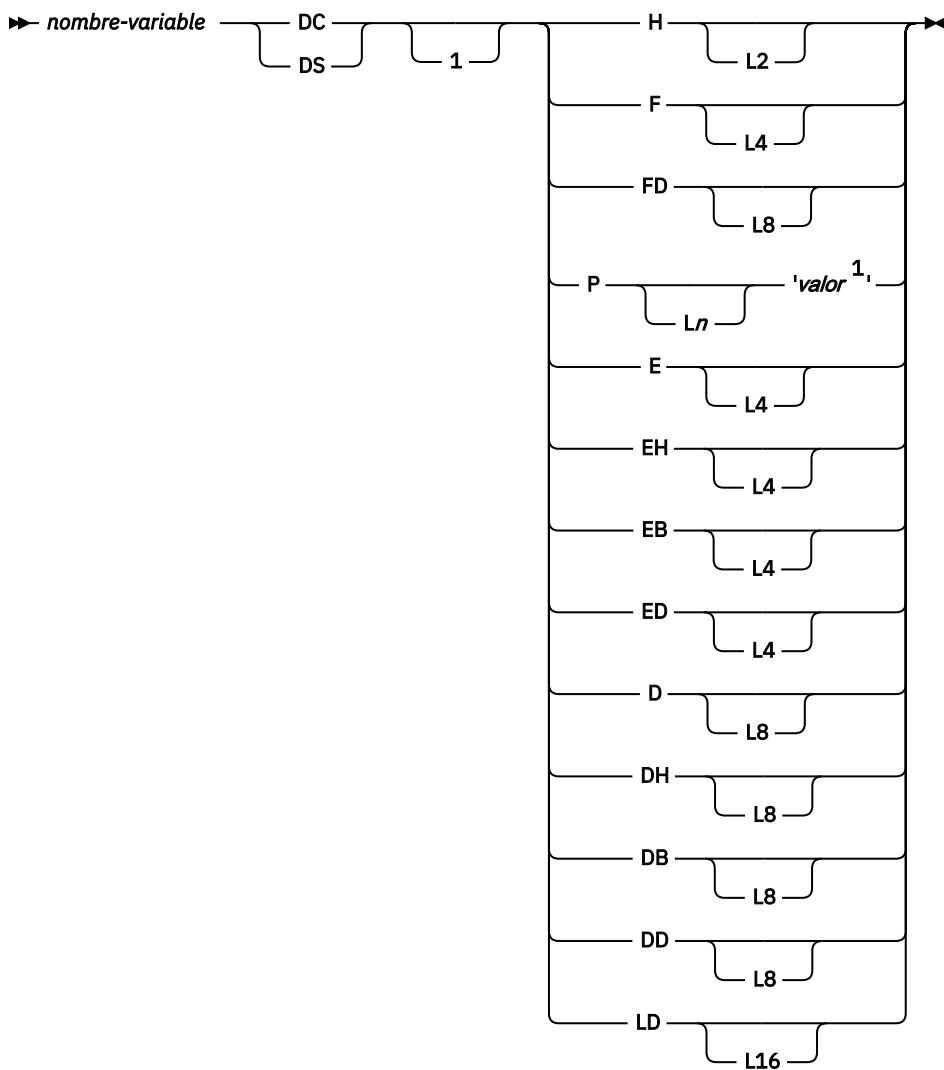
- Solo algunas de las declaraciones de ensamblador válidas son declaraciones de variables de host válidas. Si la declaración de una variable de host no es válida, cualquier instrucción SQL que haga referencia a la variable podría dar lugar al mensaje UNDECLARED HOST VARIABLE.
- Los tipos de datos del localizador son tipos de datos de lenguaje ensamblador y tipos de datos SQL. No puede utilizar localizadores como tipos de columna.

Recomendaciones:

- Tenga cuidado con el desbordamiento. Por ejemplo, supongamos que recupera un valor de columna INTEGER en una variable de host DS H, y el valor de columna es mayor que 32767. Recibirá una advertencia de desbordamiento o un error, dependiendo de si proporciona una variable indicadora.
- Tenga cuidado con el truncamiento. Por ejemplo, si recupera un valor de columna CHAR de 80 caracteres en una variable de host que se declara como DS CL70, los diez caracteres situados más a la derecha de la cadena recuperada se truncan. Si recupera un valor de columna de punto flotante o decimal en una variable de host declarada como DS F, se elimina cualquier parte fraccionaria del valor.

Variables numéricas del host

El siguiente diagrama muestra la sintaxis para declarar variables de host numéricas.



Notas:

¹ es un valor numérico que especifica la escala de la variable decimal empaquetada. Si el *valor* no incluye un punto decimal, la escala es 0.

Para los tipos de datos de punto flotante (E, EH, EB, D, DH y DB), utilice la opción de procesamiento FLOAT SQL para especificar si la variable host está en formato de punto flotante binario IEEE o de punto flotante hexadecimal de arquitectura z. Si especifica FLOAT(S390), debe definir sus variables de host de punto flotante como E, EH, D o DH. Si especifica FLOAT(IEEE), debe definir sus variables de host de punto flotante como EB o DB. Db2 no comprueba si las declaraciones de variables de host o el formato del contenido de las variables de host coinciden con el formato que especificó con la opción de procesamiento FLOAT SQL. Por lo tanto, debe asegurarse de que los tipos y contenidos de las variables de host de punto flotante coincidan con el formato que especificó con la opción de procesamiento FLOAT SQL. Db2 convierte todos los datos de entrada de punto flotante al formato de punto flotante hexadecimal de z/Architecture antes de almacenarlos.

Restricción: Las opciones de procesamiento FLOAT SQL no se aplican a los tipos de variables de host de punto flotante decimal ED, DD o LD.

Para los tipos de variables de host de coma decimal ED, DD y LD, puede especificar los siguientes valores especiales: MIN, MAX, NAN, SNAN e INFINITY.

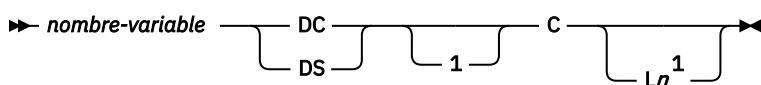
Variables de host de caracteres

Puede especificar las siguientes formas de variables de host de caracteres:

- Series de longitud fija
- Series de longitud variable
- CLOB

Los siguientes diagramas muestran la sintaxis para formularios distintos de CLOB.

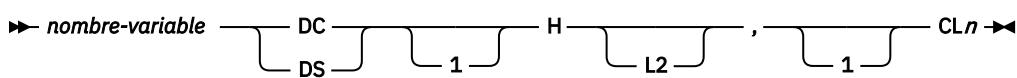
El siguiente diagrama muestra la sintaxis para declarar cadenas de caracteres de longitud fija.



Notas:

¹ Si declara una variable de host de cadena de caracteres sin longitud (por ejemplo, DC C 'ABCD'), Db2 interpreta la longitud como 1. Para obtener la longitud correcta, especifique un atributo de longitud (por ejemplo, DC CL 4 'ABCD').

El siguiente diagrama muestra la sintaxis para declarar cadenas de caracteres de longitud variable.



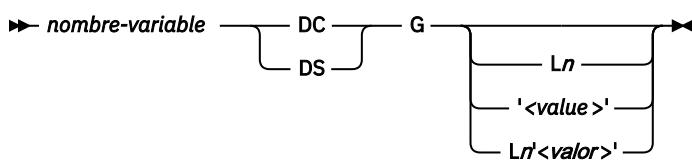
Variables gráficas de host

Puede especificar las siguientes formas de variables de host gráficas:

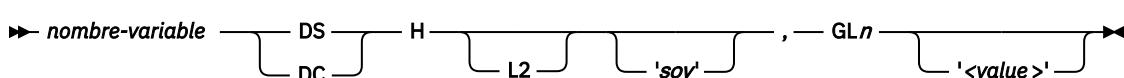
- Series de longitud fija
- Series de longitud variable
- DBCLOB

Los siguientes diagramas muestran la sintaxis para formularios distintos de DBCLOB. En los diagramas de sintaxis, el *valor* denota uno o más caracteres DBCS, y los símbolos < y > representan los caracteres de desplazamiento hacia fuera y hacia dentro.

El siguiente diagrama muestra la sintaxis para declarar cadenas gráficas de longitud fija.



El siguiente diagrama muestra la sintaxis para declarar cadenas gráficas de longitud variable.



Variables binarias de host

El siguiente diagrama muestra la sintaxis para declarar variables de host binarias.

►— *variable-name* — DS — X — Ln ¹ ►

Notas:

¹ $1 \leq n \leq 255$

Variables de host varbinarias

El siguiente diagrama muestra la sintaxis para declarar variables de host varbinary.

►— *variable-name* — DS — H — L2 — , — X — Ln ¹ ►

Notas:

¹ $1 \leq n \leq 32704$

Localizadores de conjuntos de resultados

El siguiente diagrama muestra la sintaxis para declarar localizadores de conjuntos de resultados.

►— *nombre-variable* — SQL TYPE IS RESULT_SET_LOCATOR VARYING ¹ ►

Notas:

¹ Para ser compatibles con versiones anteriores, las variables de host del localizador de conjuntos de resultados pueden declararse como enteros de palabra completa (FL4), pero el método mostrado es la sintaxis preferida.

Localizadores de mesas

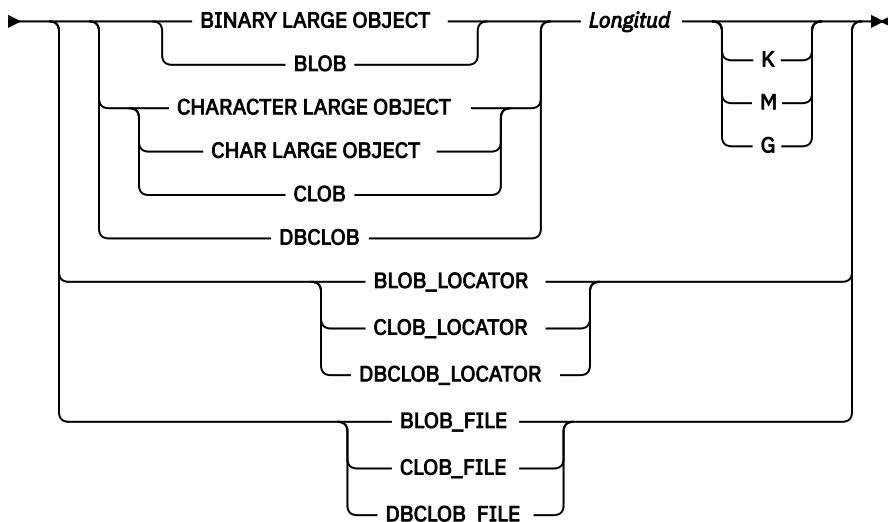
El siguiente diagrama muestra la sintaxis para declarar localizadores de tablas.

►— *nombre-variable* — SQL TYPE IS — TABLE LIKE — *nombre-de-tabla* — AS LOCATOR ►

Variables LOB, localizadores y variables de referencia de archivos

El siguiente diagrama muestra la sintaxis para declarar variables de host BLOB, CLOB y DBCLOB, localizadores y variables de referencia de archivo.

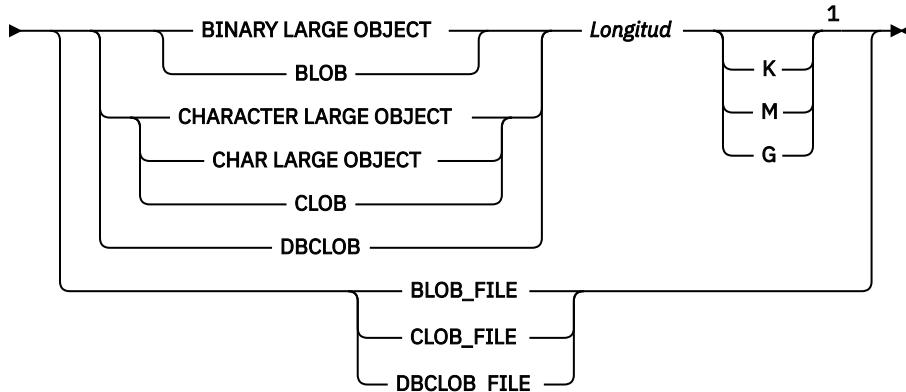
►► *variable-name* — SQL TYPE IS →



Variables de referencia de archivos y host de datos XML

El siguiente diagrama muestra la sintaxis para declarar variables de host BLOB, CLOB y DBCLOB y variables de referencia de archivo para tipos de datos XML.

►► *variable-name* — SQL TYPE IS XML AS →



Notas:

¹ Si especifica la longitud del LOB en términos de KB, MB o GB, no deje espacios entre la longitud y K, M o G.

ROWID

El siguiente diagrama muestra la sintaxis para declarar variables de host ROWID.

►► *variable-name* — SQL TYPE IS — ROWID ►►

Conceptos relacionados

Variables de host

Utilice variables host para pasar un único elemento de datos entre Db2 y la aplicación.

Uso de variables host en sentencias SQL

Utilice variables de host escalares en sentencias de SQL incorporado para representar un único valor. Las variables host son útiles para almacenar datos recuperados o para pasar valores que van a asignar o utilizar las comparaciones.

Tareas relacionadas

Determinación de si un valor recuperado en una variable host es nulo o está truncado

Antes de que su aplicación manipule los datos recuperados de Db2 en una variable host, determine si el valor es nulo. Determine también si se han truncado al asignarse a la variable. Puede utilizar las variables indicadores para obtener esta información.

Inserción de una sola fila utilizando una variable host

Utilice variables host en la sentencia INSERT cuando no conoce al menos algunos de los valores a insertar hasta que se ejecuta el programa.

Inserción de valores nulos en columnas utilizando las matrices o variables indicadoras

Si necesita insertar valores nulos en una columna, utilizar una matriz o variable indicadora es una forma fácil de hacerlo. Una variable o matriz de indicador está asociada a una variable de host o matriz de variables de lenguaje de host determinada.

Almacenamiento de datos LOB en tablas de Db2

Db2 maneja los datos LOB de manera diferente a otros tipos de datos. Como resultado, a veces es necesario realizar acciones adicionales al definir columnas LOB e insertar los datos LOB.

Recuperación de una única fila de datos en variables host

Si sabe que la consulta devuelve una sola fila, puede especificar una o más variables host que contienen los valores de columna de la fila recuperada.

Actualización de datos utilizando variables de host

Cuando quiera actualizar un valor en una tabla Db2, pero no conoce el valor exacto hasta que el programa se ejecute, utilice variables host. Db2 puede cambiar un valor de tabla para que coincida con el valor de la variable de host.

Referencia relacionada

Descripciones de opciones de proceso de SQL

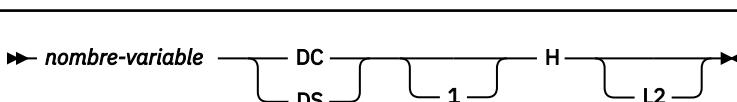
Puede especificar cualquier opción de proceso de SQL tanto si utiliza el precompilador de Db2 como si utiliza el coprocesador de Db2. Sin embargo, es probable que el coprocesador de Db2 omita ciertas opciones ya que existen opciones del compilador del lenguaje principal que proporcionan la misma información.

High Level Assembler (HLASM) y Toolkit Feature Library

Variables de indicador en Assembler

Una variable indicadora es un entero de 2 bytes (DS HL2). Declare las variables indicadoras de la misma forma que las variables host. Puede mezclar las declaraciones de dos tipos de variables.

El siguiente diagrama muestra la sintaxis para declarar una variable indicadora en ensamblador.



Ejemplo

El siguiente ejemplo muestra una sentencia FETCH con las declaraciones de las variables de host que se necesitan para la sentencia FETCH y sus variables indicadoras asociadas.

```
EXEC SQL FETCH CLS_CURSOR INTO :CLSCD,          X
                           :DAY :DAYIND,        X
                           :BGN :BGNIND,        X
                           :END :ENDIND
```

Puede declarar estas variables de la siguiente manera:

```
CLSCD    DS CL7
DAY      DS HL2
BGN      DS CL8
END      DS CL8
DAYIND   DS HL2      INDICATOR VARIABLE FOR DAY
BGNIND   DS HL2      INDICATOR VARIABLE FOR BGN
ENDIND   DS HL2      INDICATOR VARIABLE FOR END
```

Conceptos relacionados

Variables de indicación, matrices y estructuras

Una variable de indicación se asocia con una variable host concreta. Cada variable de indicación contiene un valor entero pequeño que indica alguna información sobre la variable host asociada. Las estructuras y matrices de indicador tienen el mismo propósito para las estructuras y matrices de variables de host.

Tareas relacionadas

Inserción de valores nulos en columnas utilizando las matrices o variables indicadoras

Si necesita insertar valores nulos en una columna, utilizar una matriz o variable indicadora es una forma fácil de hacerlo. Una variable o matriz de indicador está asociada a una variable de host o matriz de variables de lenguaje de host determinada.

Tipos de datos SQL y ensamblador equivalentes

Cuando declara variables host en los programas ensamblador, el precompilador utiliza tipos de datos SQL equivalentes. Cuando recupera datos de un tipo de datos SQL concreto en una variable host, asegúrese de que la variable host es de un tipo de datos equivalente.

La siguiente tabla describe el tipo de datos SQL y los valores base SQLTYPE y SQLLEN que el precompilador utiliza para las variables de host en las sentencias SQL.

Tabla 94. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas ensambladores

Tipo de datos de variable de host ensamblador	SQLTYPE de la variable de host ¹	SQLLEN de la variable de host	Tipos de datos SQL
DS HL2	500	2	SMALLINT
DS FL4	496	4	INTEGER
DS P'<value>' DS PLn'<value>' or DS PLn 1<=n<=16	484	p en el byte 1, s en el byte 2	DECIMAL(p,s)
decimal corto FLOAT: SDFP DC ED SDFP DC EDL4 SDFP DC EDL4'11.11'	996	4	DECFLOAT
fFLOAT decimal largo: LDFP DC DD LDFP DC DDL8 LDFP DC DDL8'22.22'	996	8	DECFLOAT

Tabla 94. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas ensambladores (continuación)

Tipo de datos de variable de host ensamblador	SQLTYPE de la variable de host ¹	SQLLEN de la variable de host	Tipos de datos SQL
fFLOAT decimal extendido: EDFP DC LD EDFP DC LDL16 EDFP DC LDL16'33.33'	996	16	DECFLOAT
DS EL4 DS EHL4 DS EBL4	480	4	REAL or FLOAT (n) 1<=n<=21
DS DL8 DS DHL8 DS DBL8	480	8	DOUBLE PRECISION, or FLOAT (n) 22<=n<=53
DS FDL8 DS FD	492	8	BIGINT
SQL TYPE IS BINARY(n) 1<=n<=255	912	n	BINARY(n)
SQL TYPE IS VARBINARY(n) or SQL TYPE IS BINARY(n) VARYING 1<=n<=32704	908	n	VARBINARY(n)
DS CLn 1<=n<=255	452	n	CHAR(n)
DS HL2,CLn 1<=n<=255	448	n	VARCHAR(n)
DS HL2,CLn n>255	456	n	VARCHAR(n)
DS GLm 2<=m<=254	468	n	GRAPHIC(n)
2			3
DS HL2,GLm 2<=m<=254	464	n	VARGRAPHIC(n)
2			3
DS HL2,GLm m>254	472	n	VARGRAPHIC(n)
2			3
SQL TYPE IS RESULT_SET_LOCATOR	972	4	Result set locator ^{4,5}

Tabla 94. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas ensambladores (continuación)

Tipo de datos de variable de host ensamblador	SQLTYPE de la variable de host ¹	SQLLEN de la variable de host	Tipos de datos SQL
SQL TYPE IS TABLE LIKE <i>table-name</i> AS LOCATOR	976	4	Table locator ⁴
SQL TYPE IS BLOB_LOCATOR	960	4	BLOB locator ⁴
SQL TYPE IS CLOB_LOCATOR	964	4	CLOB locator ⁴
SQL TYPE IS DBCLOB_LOCATOR	968	4	DBCLOB locator ⁴
SQL TYPE IS BLOB(<i>n</i>) $1 \leq n \leq 2147483647$	404	<i>n</i>	BLOB(<i>n</i>)
SQL TYPE IS CLOB(<i>n</i>) $1 \leq n \leq 2147483647$	408	<i>n</i>	CLOB(<i>n</i>)
SQL TYPE IS DBCLOB(<i>n</i>) $1 \leq n \leq 1073741823$	412	<i>n</i> 3	DBCLOB(<i>n</i>)
SQL TYPE IS XML AS BLOB(<i>n</i>)	404	0	XML
SQL TYPE IS XML AS CLOB(<i>n</i>)	408	0	XML
SQL TYPE IS XML AS DBCLOB(<i>n</i>)	412	0	XML
SQL TYPE IS BLOB_FILE	916/917	267	BLOB file reference ⁴
SQL TYPE IS CLOB_FILE	920/921	267	CLOB file reference ⁴
SQL TYPE IS DBCLOB_FILE	924/925	267	DBCLOB file reference ⁴
SQL TYPE IS XML AS BLOB_FILE	916/917	267	XML BLOB file reference ⁴
SQL TYPE IS XML AS CLOB_FILE	920/921	267	XML CLOB file reference ⁴
SQL TYPE IS XML AS DBCLOB_FILE	924/925	267	XML DBCLOB file reference ⁴

Tabla 94. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas ensambladores (continuación)

Tipo de datos de variable de host ensamblador	SQLTYPE de la variable de host ¹	SQLLEN de la variable de host	Tipos de datos SQL
SQL TYPE IS ROWID	904	40	ROWID ^{note 5}

Notas:

1. Si una variable host incluye una variable indicadora, el valor SQLTYPE es el valor SQLTYPE base más 1.
2. *m* es el número de bytes.
3. *n* es el número de caracteres de doble byte.
4. Este tipo de datos no se puede utilizar como tipo de columna.
5. Para ser compatibles con versiones anteriores, las variables de host del localizador de conjuntos de resultados pueden declararse como enteros de palabra completa (FL4), pero el método mostrado es la sintaxis preferida.

La siguiente tabla muestra las variables de host del ensamblador equivalentes para cada tipo de datos SQL. Utilice esta tabla para determinar el tipo de datos del ensamblador para las variables de host que defina para recibir la salida de la base de datos. Por ejemplo, si recupera datos TIMESTAMP, puede definir la variable DS_CLn.

Esta tabla muestra las conversiones directas entre los tipos de datos SQL y los tipos de datos ensamblador. Sin embargo, varios tipos de datos SQL son compatibles. Cuando realiza asignaciones o comparaciones de datos que tienen tipos de datos compatibles, Db2 convierte esos tipos de datos compatibles.

Tabla 95. Variables equivalentes de host ensamblador que puede utilizar al recuperar datos de un tipo de datos SQL concreto

Tipos de datos SQL	Variable de host ensamblador equivalente	Notas
SMALLINT	DS HL2	
ENTERO	DS F	
BIGINT	DS FD O DS FDL8	FDL8 High Level Assembler, requiere Microsoft®.NET Framework 4.0 (HLASM), versión 4 o posterior.

Tabla 95. Variables equivalentes de host ensamblador que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host ensamblador equivalente	Notas
DECIMAL(<i>p,s</i>) o NUMERIC(<i>p,s</i>)	DS P 'valor' DS PL <i>n</i> 'valor' DS PL <i>n</i>	<i>p</i> es precisión; <i>s</i> es escala. $1 \leq p \leq 31$ y $0 \leq s \leq p$. $1 \leq n \leq 16$. <i>valor</i> es un valor literal que incluye un punto decimal. Debe utilizar <i>Ln</i> , <i>value</i> o ambos. Se recomienda utilizar solo <i>valor</i> . Precisión : si utiliza <i>Ln</i> , es $2n-1$; de lo contrario, es el número de dígitos del <i>valor</i> . Escala : si utiliza <i>el valor</i> , es el número de dígitos a la derecha del punto decimal; de lo contrario, es 0. Para un uso eficiente de los índices : Utilizar <i>valor</i> . Si <i>p</i> es par, no utilice <i>Ln</i> y asegúrese de que la precisión del <i>valor</i> es <i>p</i> y la escala <i>del valor</i> es <i>s</i> . Si <i>p</i> es impar, puedes usar <i>Ln</i> (aunque no se aconseja), pero debes elegir <i>n</i> de manera que $2n-1 = p$, y <i>valor</i> de manera que la escala sea <i>s</i> . Incluya un punto decimal en <i>el valor</i> , incluso cuando la escala de <i>valor</i> sea 0.
REAL o FLOAT(<i>n</i>)	DS EL4 DS EHL4 DS EBL4 ¹	$1 \leq n \leq 21$
DOBLE PRECISIÓN, DOBLE, o FLOAT(<i>n</i>)	DS DL8 DS DHL8 DS DBL8 ¹	$2 \leq n \leq 53$
DECFLOAT	EDL4 DC DDL8 DC LDL16	
CHAR (<i>n</i>)	DS CL <i>n</i>	$1 \leq n \leq 255$
VAR CHAR(<i>n</i>)	DS HL2, CL <i>n</i>	
GRÁFICO(<i>s</i>)	DS GL <i>m</i>	<i>m</i> se expresa en bytes. <i>n</i> es el número de caracteres de doble byte. $1 \leq n \leq 127$
VARGRÁFICO(<i>n</i>)	DS HL2, GL <i>x</i> DS HL2 ' <i>m</i> ', GL ' <i>x'<value></i> '	<i>x</i> y <i>m</i> se expresan en bytes. <i>n</i> es el número de caracteres de doble byte. < <i>y</i> > representan caracteres de desplazamiento hacia fuera y hacia dentro.
BINARIO(<i>n</i>)	Formato 1: variable-name-- DS--X--Ln Formato 2: TIPO SQL ES BINARIO(<i>n</i>)	$1 \leq n \leq 255$
VARBINARY(<i>n</i>)	Formato 1: variable-name-- DS--H--L2--, --X-- Ln Formato 2: SQL TYPE IS VARBINARY(<i>n</i>) o SQL TYPE IS BINARY(<i>n</i>) VARYING	$1 \leq n \leq 32704$

Tabla 95. Variables equivalentes de host ensamblador que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host ensamblador equivalente	Notas
FECHA	<i>DS CLn</i>	Si está utilizando una rutina de salida de fecha, <i>n</i> viene determinada por dicha rutina; de lo contrario, <i>n</i> debe ser al menos 10.
HORA	<i>DS CLn</i>	Si está utilizando una rutina de salida de tiempo, <i>n</i> está determinado por esa rutina. De lo contrario, <i>n</i> debe ser al menos 6; para incluir segundos, <i>n</i> debe ser al menos 8.
TIMESTAMP	<i>DS CLn</i>	<i>n</i> debe tener al menos 19 años. Para incluir microsegundos, <i>n</i> debe ser 26; si <i>n</i> es menor que 26, se produce un truncamiento en la parte de los microsegundos.
TIMESTAMP(0)	<i>DS CLn</i>	<i>n</i> debe tener al menos 19 años.
TIMESTAMP(<i>p</i>) <i>p</i> > 0	<i>DS CLn</i>	<i>n</i> debe tener al menos 19 años. Para incluir fracciones de segundo, <i>n</i> debe ser 20+ <i>x</i> , donde <i>x</i> es el número de fracciones de segundo que se van a incluir; si <i>x</i> es menor que <i>p</i> , se produce un truncamiento en la parte de las fracciones de segundo.
TIMESTAMP (0) WITH TIME ZONE	<i>DS HL2, CLn</i>	<i>n</i> debe tener al menos 25 años.
FECHA Y HORA (<i>p</i>) CON ZONA HORARIA <i>p</i> > 0	<i>DS HL2, CLn</i>	<i>n</i> debe tener al menos 26+ <i>p</i> .
Localizador de conjunto de resultados	<i>DS F</i>	Utilice este tipo de datos solo para recibir conjuntos de resultados. No utilice este tipo de datos como tipo de columna.
Localizador de tablas	SQL TYPE IS TABLE LIKE <i>nombre-tabla</i> AS LOCATOR	Utilice este tipo de datos solo en una función definida por el usuario o en un procedimiento almacenado para recibir filas de una tabla de transición. No utilice este tipo de datos como tipo de columna.
localizador de BLOB	TIPO SQL ES BLOB_LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas BLOB. No utilice este tipo de datos como tipo de columna.
localizador de CLOB	TIPO SQL ES CLOB_LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas CLOB. No utilice este tipo de datos como tipo de columna.
localizador de DBCLOB	TIPO SQL ES DBCLOB_LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas DBCLOB. No utilice este tipo de datos como tipo de columna.
BL OB (<i>n</i>)	TIPO SQL ES BLOB(<i>n</i>)	1≤ <i>n</i> ≤2147483647
CLOB(<i>n</i>)	TIPO SQL ES CLOB(<i>n</i>)	1≤ <i>n</i> ≤2147483647
DBCL OB (<i>n</i>)	TIPO SQL ES DBCLOB(<i>n</i>)	<i>n</i> es el número de caracteres de doble byte. 1≤ <i>n</i> ≤1073741823

Tabla 95. Variables equivalentes de host ensamblador que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host ensamblador equivalente	Notas
XML	SQL TYPE IS XML AS <i>BLOB(n)</i>	1≤n≤2147483647
XML	SQL TYPE IS XML AS <i>CLOB(n)</i>	1≤n≤2147483647
XML	SQL TYPE IS XML AS <i>DBCLOB(n)</i>	<i>n</i> es el número de caracteres de doble byte. 1≤n≤1073741823
Referencia de archivo BLOB	TIPO SQL ES BLOB_FILE	Utilice este tipo de datos solo para manipular datos en columnas BLOB. No utilice este tipo de datos como tipo de columna.
Referencia de archivo CLOB	TIPO SQL ES CLOB_FILE	Utilice este tipo de datos solo para manipular datos en columnas CLOB. No utilice este tipo de datos como tipo de columna.
Referencia del archivo DBCLOB	TIPO SQL ES DBCLOB_FILE	Utilice este tipo de datos solo para manipular datos en columnas DBCLOB. No utilice este tipo de datos como tipo de columna.
Referencia de archivo XML BLOB	SQL TYPE IS XML AS BLOB_FILE	Utilice este tipo de datos solo para manipular datos XML como archivos BLOB. No utilice este tipo de datos como tipo de columna.
Referencia de archivo XML CLOB	SQL TYPE IS XML AS CLOB_FILE	Utilice este tipo de datos solo para manipular datos XML como archivos CLOB. No utilice este tipo de datos como tipo de columna.
Referencia de archivo XML DBCLOB	SQL TYPE IS XML AS DBCLOB_FILE	Utilice este tipo de datos solo para manipular datos XML como archivos DBCLOB. No utilice este tipo de datos como tipo de columna.
ROWID	EL TIPO SQL ES ROWID	

Notas:

1. Aunque los procedimientos almacenados y las funciones definidas por el usuario pueden utilizar variables de host de punto flotante IEEE, no se puede declarar una función definida por el usuario o un parámetro de procedimiento almacenado como IEEE.

La siguiente tabla muestra las definiciones de lenguaje ensamblador que se deben utilizar en los procedimientos almacenados en ensamblador y en las funciones definidas por el usuario, cuando los tipos de datos de los parámetros en las definiciones de rutina son LOB, ROWID o localizadores. Para otros tipos de datos de parámetros, las definiciones del lenguaje ensamblador son las mismas que las de la sección anterior ([Tabla 95 en la página 597](#)).

Tabla 96. Declaraciones de lenguaje ensamblador equivalentes para LOB, ROWID y localizadores en definiciones de rutinas definidas por el usuario

Tipo de datos SQL en la definición	Declaración del ensamblador
localizador de tablas	DS FL4
localizador de BLOB	
localizador de CLOB	
localizador de DBCLOB	

Tabla 96. Declaraciones de lenguaje ensamblador equivalentes para LOB, ROWID y localizadores en definiciones de rutinas definidas por el usuario (continuación)

Tipo de datos SQL en la definición	Declaración del ensamblador
BL OB (n)	<p>Si n <= 65535:</p> <pre>var DS 0FL4 var_length DS FL4 var_data DS CLn</pre> <p>Si n > 65535:</p> <pre>var DS 0FL4 var_length DS FL4 var_data DS CL65535 ORG var_data+(n-65535)</pre>
CLOB(n)	<p>Si n <= 65535:</p> <pre>var DS 0FL4 var_length DS FL4 var_data DS CLn</pre> <p>Si n > 65535:</p> <pre>var DS 0FL4 var_length DS FL4 var_data DS CL65535 ORG var_data+(n-65535)</pre>
DBCL OB (n)	<p>Si n (=2*n) <= 65534:</p> <pre>var DS 0FL4 var_length DS FL4 var_data DS CLm</pre> <p>Si n > 65534:</p> <pre>var DS 0FL4 var_length DS FL4 var_data DS CL65534 ORG var_data+(m-65534)</pre>
ROWID	DS HL2,CL40

Conceptos relacionados

[Compatibilidad de SQL y tipos de datos de lenguaje](#)

Los tipos de datos de variable host que se utilizan en sentencias SQL deben ser compatibles con los tipos de datos de las columnas con las que tiene intención de utilizarlos.

[Declaraciones de variable de referencia de archivo LOB, variable host LOB y localizador LOB](#)

Cuando escribe aplicaciones para manipular datos LOB, necesita declarar variables hosto para contener los datos LOB o el localizador LOB. De lo contrario, debe declarar variables de referencia de archivo LOB para apuntar a datos LOB.

[Tipos de datos de variable host para datos XML en aplicaciones de SQL incorporado \(Db2 Programming for XML\)](#)

Macros para aplicaciones ensambladoras

Conjunto de datos DSN1210.SDSNMACS contiene todas las macros de Db2 , que están disponibles para su uso.

Gestión de códigos de error de SQL en aplicaciones de ensamblador

Las aplicaciones Assembler pueden solicitar más información sobre los códigos de error de SQL utilizando la subrutina DSNTIAR o emitiendo una sentencia GET DIAGNOSTICS.

Procedimiento

Para solicitar información sobre errores SQL en programas ensambladores, utilice los siguientes métodos:

- Puede utilizar la subrutina DSNTIAR para convertir un código de retorno SQL en un mensaje de texto. DSNTIAR toma datos de SQLCA, los formatea en un mensaje y coloca el resultado en un área de salida de mensajes que usted proporciona en su programa de aplicación. Para conceptos y más información sobre el comportamiento de DSNTIAR, consulte “[Visualización de campos SQLCA invocando DSNTIAR](#)” en la página 558.

Sintaxis DSNTIAR

DSNTIAR tiene la siguiente sintaxis:

```
CALL DSNTIAR,(sqlca, message, lrecl),MF=(E,PARM)
```

Parámetros DSNTIAR

Los parámetros DSNTIAR tienen los siguientes significados:

sqlca

Un área de comunicación SQL.

mensaje

Un área de salida, definida como una cadena de longitud variable, en la que DSNTIAR coloca el texto del mensaje. La primera semicifra contiene la longitud del área restante; su valor mínimo es 240.

Las líneas de salida de texto, cada una con la longitud especificada en *lrecl*, se colocan en esta área. Por ejemplo, podría especificar el formato del área de salida como:

```
LINES      EQU    10
LRECL      EQU    132
...
MSGLRECL  DC     AL4(LRECL)
MESSAGE    DS     H,CL(LINES*LRECL)
ORG       MESSAGE
MESSAGE1   DC     AL2(LINES*LRECL)
MESSAGE1   DS     CL(LRECL)      text line 1
MESSAGE2   DS     CL(LRECL)      text line 2
...
MESSAGEEn  DS     CL(LRECL)      text line n
...
CALL DSNTIAR,(SQLCA,MESSAGE,MSGLRECL),MF=(E,PARM)
```

donde MESSAGE es el nombre del área de salida del mensaje, LINES es el número de líneas en el área de salida del mensaje y LRECL es la longitud de cada línea.

lrecl

Una palabra completa que contiene la longitud de registro lógica de los mensajes de salida, en el rango de 72 a 240.

La expresión MF=(E,PARM) es un z/OS parámetro macro que indica ejecución dinámica. PARM es el nombre de un área de datos que contiene una lista de punteros a los parámetros de llamada de DSNTIAR.

Visite “[Aplicaciones de ejemplo suministradas con Db2 for z/OS](#)” en la página 1091 para obtener instrucciones sobre cómo acceder e imprimir el código fuente del programa de muestra.

- Si su CICS solicitud requiere CICS almacenamiento, debe utilizar la subrutina DSNTIAC en lugar de DSNTIAR.

Sintaxis DSNTIAC

DSNTIAC tiene la siguiente sintaxis:

```
CALL DSNTIAC,(eib,commarea,sqlca,msg,lrecl),MF=(E,PARM)
```

Parámetros DSNTIAC

DSNTIAC tiene parámetros adicionales, que debe utilizar para llamadas a rutinas que utilizan CICS comandos.

eib

Bloque de interfaz EXEC

área de clientes

área de comunicación

Para obtener más información sobre estos parámetros, consulte la guía de programación de aplicaciones correspondiente para CICS. Las descripciones de los parámetros restantes son las mismas que las de DSNTIAR. Tanto DSNTIAC como DSNTIAR formatean el SQLCA de la misma manera.

Debe definir DSNTIA1 en el CSD. Si carga DSNTIAR o DSNTIAC, también debe definirlos en el CSD. Para ver un ejemplo de declaraciones de generación de entradas CSD para su uso con DSNTIAC, consulte el miembro DSN8FRDO en *el prefijo* del conjunto de datos.SDSENSAMP.

El código fuente ensamblador para DSNTIAC y el trabajo DSNTEJ5A, que ensambla y edita enlaces DSNTIAC, también están en *el prefijo* del conjunto de datos.SDSENSAMP.

- También puede utilizar el campo de elemento de condición MESSAGE_TEXT de la instrucción GET DIAGNOSTICS para convertir un código de retorno SQL en un mensaje de texto.

Los programas que requieren un soporte de mensajes token largos deben codificar la instrucción GET DIAGNOSTICS en lugar de DSNTIAR. Para obtener más información sobre GET DIAGNOSTICS, consulte ["Comprobación de la ejecución de sentencias SQL utilizando la instrucción GET DIAGNOSTICS"](#) en la página 563.

Tareas relacionadas

[Manejo de códigos de error de SQL](#)

Los programas de aplicación pueden solicitar más información sobre los códigos de error SQL en Db2.

Referencia relacionada

[GET DIAGNOSTICS declaración \(Db2 SQL\)](#)

Aplicaciones C y C++ que emiten sentencias SQL

Puede codificar sentencias SQL en un programa C o C++ siempre que pueda utilizar sentencias ejecutables.

Cada instrucción SQL en un programa C o C++ debe comenzar con EXEC SQL y terminar con un punto y coma (;). Las palabras clave EXEC y SQL deben aparecer en una línea, pero el resto de la instrucción puede aparecer en líneas posteriores.

En general, como C distingue entre mayúsculas y minúsculas, utilice letras mayúsculas para introducir todas las palabras clave SQL. Sin embargo, si utiliza la subopción del precompilador FOLD, Db2 convierte en mayúsculas las letras minúsculas de los identificadores ordinarios de SBCS SQL. Para obtener información sobre las opciones del precompilador del idioma de destino, consulte [Tabla 140 en la página 914.](#)

Debe mantener la coherencia de los nombres de las variables de host en todo el programa. Por ejemplo, si el nombre de una variable de host está en minúsculas en su declaración, debe estar en minúsculas en todas las sentencias SQL. Podría codificar una instrucción UPDATE en un programa C de la siguiente manera:

```
EXEC SQL  
  UPDATE DSN8C10.DEPT  
  SET MGRNO = :mgr_num  
 WHERE DEPTNO = :int_dept;
```

Comentarios

Puede incluir comentarios C /*... */ dentro de las sentencias SQL siempre que pueda utilizar un espacio en blanco, excepto entre las palabras clave EXEC y SQL. Puede utilizar comentarios de una sola línea (que comienzan con //) en las sentencias del lenguaje C, pero no en SQL incrustado. Puede utilizar comentarios SQL dentro de instrucciones SQL incrustadas. Para obtener más información, consulte [Comentarios SQL \(Db2 SQL\)](#).

Puede anidar comentarios.

Para incluir caracteres EBCDIC DBCS en comentarios, debe delimitar los caracteres con un carácter de control de desplazamiento de salida y de entrada; el primer carácter de desplazamiento de entrada en la cadena DBCS señala el final de la cadena DBCS.

Continuación para instrucciones SQL

Puede utilizar una barra invertida para continuar una cadena de caracteres constante o un identificador delimitado en la línea siguiente. Sin embargo, las constantes de cadena EBCDIC DBCS no pueden continuar en una segunda línea.

Delimitadores

Delimite una instrucción SQL en su programa C con la palabra clave inicial " EXEC SQL " y un punto y coma (;).

Declaración de tablas y vistas

Su programa C debe utilizar la instrucción DECLARE TABLE para describir cada tabla y ver los accesos del programa. Puede utilizar el generador de declaraciones de Db2 (DCLGEN) para generar las declaraciones DECLARE TABLE. Para obtener más información, consulte ["DCLGEN \(generador de declaraciones\)"](#) en la página 490.

Incluir instrucciones SQL y declaraciones de variables en el código fuente que va a ser procesado por el precompilador de PHP (Db2)

Para incluir sentencias SQL o declaraciones de variables de host C de un miembro de un conjunto de datos particionado, añada la siguiente sentencia SQL al código fuente donde desee incluir las sentencias:

```
EXEC SQL INCLUDE member-name;
```

No se pueden anidar sentencias SQL INCLUDE. No utilice sentencias C #include para incluir sentencias SQL o declaraciones de variables de host C.

Márgenes

Codifique las sentencias SQL en las columnas 1-72, a menos que especifique otros márgenes al precompilador Db2 . Si EXEC SQL no está dentro de los márgenes especificados, el precompilador de SQL (Db2) no reconoce la instrucción SQL. Las reglas de margen no se aplican al coprocesador de la tecnología de pago de tarjetas de crédito (Db2). El coprocesador de longitud variable (Db2) permite la entrada de fuentes de longitud variable.

Nombres

Puede utilizar cualquier nombre C válido para una variable de host, sujeto a las siguientes restricciones:

- No utilice caracteres DBCS.
- No utilice nombres de entrada externos o nombres de planes de acceso que empiecen por «DSN», ni nombres de variables de host o nombres de macros que empiecen por «SQL» (en cualquier combinación de letras mayúsculas o minúsculas). Estos nombres están reservados para Db2.

Un identificador SQL que comienza con el carácter almohadilla ('#') puede interpretarse como una instrucción de macro C.

Nulos y NUL

C y SQL difieren en la forma en que utilizan la palabra *nula*. El lenguaje C tiene un carácter nulo (NUL), un puntero nulo (NULL) y una instrucción nula (solo un punto y coma). El C NUL es un único carácter que se compara igual a 0. El C NULL es un valor de puntero reservado especial que no apunta a ningún objeto de datos válido. El valor nulo SQL es un valor especial que se distingue de todos los valores no

nulos y denota la ausencia de un valor (no nulo). NUL (o NUL-terminator) es el carácter nulo en C y C++, y NULL es el valor nulo de SQL.

Números de secuencia

El precompilador de Db2 genera instrucciones sin números de secuencia. (El coprocesador Db2 no realiza esta acción, porque el compilador lee y modifica el código fuente)

Etiquetas de extracto

Puede preceder las sentencias SQL con una etiqueta.

Caracteres trigraph

Algunos caracteres del conjunto de caracteres C no están disponibles en todos los teclados. Puede introducir estos caracteres en un programa fuente C utilizando una secuencia de tres caracteres llamada *trigraph*. Los caracteres trigráficos que admite Db2 son los mismos que admite el compilador C.

WHENEVER, sentencia

El objetivo de la cláusula GOTO en una instrucción SQL WHENEVER debe estar dentro del ámbito de cualquier instrucción SQL a la que afecte la instrucción WHENEVER.

Consideraciones especiales de C/C++

- El uso de la C/370 función multitarea, en la que múltiples tareas ejecutan sentencias SQL, provoca resultados impredecibles.
- Excepto para el coprocesador de C++ (Db2), debe ejecutar el precompilador de C++ (Db2) antes de ejecutar el preprocesador de C.
- A excepción del coprocesador Db2 , el precompilador Db2 no admite directivas del preprocesador C.
- Si utiliza directivas de compilador condicionales que contienen código C, colóquelas después del primer token C en su programa de aplicación o inclúyelas en el programa C utilizando la directiva de preprocesador "#include".

Consulte la documentación C adecuada para obtener más información sobre las directivas del preprocesador C.

Para utilizar el tipo de datos de host de coma flotante decimal, debe hacer lo siguiente:

- Utilice z/OS 1.10 o superior (z/OS V1R10 XL C/C++).
- Compilar con la opción de compilador C/C++, DFP.
- Especifique la opción del compilador SQL para habilitar el coprocesador de optimización de consultas (Db2).
- Especifique la opción del compilador C/C++, ARCH(7). Es requerido por la opción del compilador DFP si el tipo DFP se utiliza en el código fuente.
- Especifique la opción del compilador 'DEFINE(__STDC_WANT_DEC_FP__).'

Manejo de códigos de error de SQL

Las aplicaciones C y C++ pueden solicitar más información sobre errores SQL en Db2. Para obtener más información, consulte "[Gestión de códigos de error de SQL en aplicaciones C y C++](#)" en la página [652](#).

Conceptos relacionados

[Utilización de matrices de variables de host en sentencias SQL](#)

Utilice matrices de variables de lenguaje de host en sentencias de SQL incorporado para representar valores que el programa no conoce hasta que se ejecuta la consulta. Las matrices de variables de host son útiles para almacenar un conjunto de valores recuperados o para pasar un conjunto de valores que se van a insertar en una tabla.

Tareas relacionadas

[Descripción general de las aplicaciones de programación que acceden a datos de Db2 for z/OS](#)

Las aplicaciones que interactúan con Db2, en primer lugar se deben conectar a Db2. Entonces pueden leer, añadir o modificar datos o manipular objetos de Db2.

[Inclusión de SQL dinámico en el programa](#)

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

Manejo de códigos de error de SQL

Los programas de aplicación pueden solicitar más información sobre los códigos de error SQL en Db2.

Establecimiento de límites para el uso de recursos de sistema utilizando el recurso de límite de recursos (Db2 Performance)

Ejemplos de programación en C y C++

Puede escribir programas en Db2 en C y C++. Estos programas pueden acceder a un subsistema de e Db2 es local o remoto y pueden ejecutar sentencias SQL estáticas o dinámicas. Esta información contiene varios ejemplos de programación de ese tipo.

Para preparar y ejecutar estas aplicaciones, comience con el JCL del miembro para su idioma en *prefijo.SDSENSAMP* como modelo para su JCL:

- Para C, utilice el trabajo [DSNTEJ2D](#).
- Para C++, utilice el trabajo [DSNTEJ2E](#).

Referencia relacionada

[Ejemplos de programación de Assembler, C, C++, COBOL, PL/I y REXX \(Db2 Programming samples\)](#)

SQL estático y dinámico de ejemplo en un programa C

Los programas que acceden a Db2 pueden contener SQL estático, SQL dinámico o ambos.

Este ejemplo muestra un programa en C que contiene SQL estático y dinámico.

La siguiente figura ilustra SQL dinámico y SQL estático incrustado en un programa C. Cada sección del programa se identifica con un comentario. La sección 1 del programa muestra SQL estático; las secciones 2, 3 y 4 muestran SQL dinámico. La función de cada sección se explica detalladamente en el prólogo del programa.

```
/********************************************/  
/* Descriptive name = Dynamic SQL sample using C language */  
/* Function = To show examples of the use of dynamic and static */  
/*             SQL. */  
/* */  
/* Notes = This example assumes that the EMP and DEPT tables are */  
/*         defined. They need not be the same as the DB2 Sample */  
/*         tables. */  
/* */  
/* Module type      = C program */  
/* Processor       = DB2 precompiler, C compiler */  
/* Module size     = see link edit */  
/* Attributes      = not reentrant or reusable */  
/* */  
/* Input          = */  
/* */  
/*             symbolic label/name = DEPT */  
/*             description = arbitrary table */  
/*             symbolic label/name = EMP */  
/*             description = arbitrary table */  
/* */  
/* Output         = */  
/* */  
/*             symbolic label/name = SYSPRINT */  
/*             description = print results via printf */  
/* */  
/* Exit-normal    = return code 0 normal completion */  
/* */  
/* Exit-error     = */  
/* */  
/*     Return code   = SQLCA */  
/* */  
/*     Abend codes   = none */  
/* */  
/* External references = none */  
/* */  
/*     Control-blocks = */  
/*             SQLCA - sql communication area */  
/* */
```

```

/*
/* Logic specification:                                */
/* There are four SQL sections.                      */
/*
/* 1) STATIC SQL 1: using static cursor with a SELECT statement.   */
/* Two output host variables.                         */
/* 2) Dynamic SQL 2: Fixed-list SELECT, using same SELECT statement */
/* used in SQL 1 to show the difference. The prepared string      */
/* :iptstr can be assigned with other dynamic-able SQL statements.*/
/* 3) Dynamic SQL 3: Insert with parameter markers.               */
/* Using four parameter markers which represent four input host   */
/* variables within a host structure.                      */
/* 4) Dynamic SQL 4: EXECUTE IMMEDIATE                  */
/* A GRANT statement is executed immediately by passing it to DB2 */
/* via a varying string host variable. The example shows how to  */
/* set up the host variable before passing it.            */
/*
*****                                         */

#include "stdio.h"
#include "stdefs.h"
EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;
EXEC SQL BEGIN DECLARE SECTION;
short edlevel;
struct { short len;
          char x1??(56??);
      } stmtbf1, stmtbf2, inpstr;
struct { short len;
          char x1??(15??);
      } lname;
short hv1;
struct { char deptno??(4??);
          struct { short len;
                    char x??(36??);
                } deptname;
          char mgrno??(7??);
          char admrdept??(4??);
          char location??(17??);
      } hv2;
short ind??(4??);
EXEC SQL END    DECLARE SECTION;
EXEC SQL DECLARE EMP TABLE
  (EMPNO           CHAR(6)
  FIRSTNAME        VARCHAR(12)
  MIDINIT          CHAR(1)
  LASTNAME         VARCHAR(15)
  WORKDEPT        CHAR(3)
  PHONENO          CHAR(4)
  HIREDATE        DECIMAL(6)
  JOBCODE          DECIMAL(3)
  EDLEVEL          SMALLINT
  SEX              CHAR(1)
  BIRTHDATE        DECIMAL(6)
  SALARY           DECIMAL(8,2)
  FORFNAME         VARGRAPHIC(12)
  FORMNAME         GRAPHIC(1)
  FORLNAME         VARGRAPHIC(15)
  FORADDR          VARGRAPHIC(256) );
EXEC SQL DECLARE DEPT TABLE
  (
  DEPTNO           CHAR(3)
  DEPTNAME         VARCHAR(36)
  MGRNO            CHAR(6)
  ADMRDEPT         CHAR(3)
  LOCATION          CHAR(16));
main ()
{
printf("??/n***      begin of program                               ***");
EXEC SQL WHENEVER SQLERROR GO TO HANDLERR;
EXEC SQL WHENEVER SQLWARNING GO TO HANDWARN;
EXEC SQL WHENEVER NOT FOUND GO TO NOTFOUND;
/*
*****                                         */
/* Assign values to host variables which will be input to DB2      */
/******                                         */
strcpy(hv2.deptno,"M92");
strcpy(hv2.deptname.x,"DDL");
hv2.deptname.len = strlen(hv2.deptname.x);
strcpy(hv2.mgrno,"000010");
strcpy(hv2.admrdept,"A00");
/*
*****                                         */

```

```

/* Static SQL 1: DECLARE CURSOR, OPEN, FETCH, CLOSE */  

/* Select into :edlevel, :lname */  

/*********************************************************************  

printf("??/n***      begin declare                      ***");  

EXEC SQL DECLARE C1 CURSOR FOR SELECT EDLEVEL, LASTNAME FROM EMP  

      WHERE EMPNO = '000010';  

printf("??/n***      begin open                         ***");  

EXEC SQL OPEN C1;  

printf("??/n***      begin fetch                        ***");  

EXEC SQL FETCH C1 INTO :edlevel, :lname;  

printf("??/n*** returned values                      ***");  

printf("??/n??/nedlevel = %d",edlevel);  

printf("??/nlname = %s\n",lname.x1);  

printf("??/n***      begin close                        ***");  

EXEC SQL CLOSE C1;  

/*********************************************************************  

/* Dynamic SQL 2: PREPARE, DECLARE CURSOR, OPEN, FETCH, CLOSE */  

/* Select into :edlevel, :lname */  

/*********************************************************************  

sprintf (inpstr.x1,  

        "SELECT EDLEVEL, LASTNAME FROM EMP WHERE EMPNO = '000010'");  

inpstr.len = strlen(inpstr.x1);  

printf("??/n***      begin prepare                     ***");  

EXEC SQL PREPARE STAT1 FROM :inpstr;  

printf("??/n***      begin declare                     ***");  

EXEC SQL DECLARE C2 CURSOR FOR STAT1;  

printf("??/n***      begin open                       ***");  

EXEC SQL OPEN C2;  

printf("??/n***      begin fetch                        ***");  

EXEC SQL FETCH C2 INTO :edlevel;  

printf("??/n*** returned values                      ***");  

printf("??/n??/nedlevel = %d",edlevel);  

printf("??/nlname = %s??/n",lname.x1);  

printf("??/n***      begin close                        ***");  

EXEC SQL CLOSE C2;  

/*********************************************************************  

/* Dynamic SQL 3: PREPARE with parameter markers */  

/* Insert into with five values. */  

/*********************************************************************  

sprintf (stmtbf1.x1,  

        "INSERT INTO DEPT VALUES (?,?,?,?,?)");  

stmtbf1.len = strlen(stmtbf1.x1);  

printf("??/n***      begin prepare                     ***");  

EXEC SQL PREPARE s1 FROM :stmtbf1;  

printf("??/n***      begin execute                    ***");  

EXEC SQL EXECUTE s1 USING :hv2:ind;  

printf("??/n***      following are expected insert results ***");  

printf("??/n hv2.deptno = %s",hv2.deptno);  

printf("??/n hv2.deptname.len = %d",hv2.deptname.len);  

printf("??/n hv2.deptname.x = %s",hv2.deptname.x);  

printf("??/n hv2.mgrno = %s",hv2.mgrno);  

printf("??/n hv2.admrdept = %s",hv2.admrdept);  

printf("??/n hv2.location = %s",hv2.location);  

EXEC SQL COMMIT;  

/*********************************************************************  

/* Dynamic SQL 4: EXECUTE IMMEDIATE */  

/* Grant select */  

/*********************************************************************  

sprintf (stmtbf2.x1,  

        "GRANT SELECT ON EMP TO USERX");  

stmtbf2.len = strlen(stmtbf2.x1);  

printf("??/n***      begin execute immediate       ***");  

EXEC SQL EXECUTE IMMEDIATE :stmtbf2;  

printf("??/n***      end of program                  ***");  

goto progend;  

HANDWARN: HANDLERR: NOTFOUND: ;  

printf("??/n SQLCODE = %d",SQLCODE);  

printf("??/n SQLWARN0 = %c",SQLWARN0);  

printf("??/n SQLWARN1 = %c",SQLWARN1);  

printf("??/n SQLWARN2 = %c",SQLWARN2);  

printf("??/n SQLWARN3 = %c",SQLWARN3);  

printf("??/n SQLWARN4 = %c",SQLWARN4);  

printf("??/n SQLWARN5 = %c",SQLWARN5);  

printf("??/n SQLWARN6 = %c",SQLWARN6);  

printf("??/n SQLWARN7 = %c",SQLWARN7);  

printf("??/n SQLERRMC = %s",sqlca.sqlerrmc);

```

```
    progend: ;
}
```

Ejemplo de programa C que llama a un procedimiento almacenado

Puede llamar a la versión en lenguaje C del procedimiento almacenado GETPRML que utiliza la convención de vinculación GENERAL WITH NULLS.

Dado que el procedimiento almacenado devuelve conjuntos de resultados, este programa comprueba si hay conjuntos de resultados y recupera el contenido de los mismos. La siguiente figura contiene el programa de ejemplo en C que llama al procedimiento almacenado GETPRML.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main()
{
    /*****
    /* Include the SQLCA and SQLDA
    *****/
    EXEC SQL INCLUDE SQLCA;
    EXEC SQL INCLUDE SQLDA;
    /*****
    /* Declare variables that are not SQL-related.
    *****/
    short int i;           /* Loop counter */
    /*****
    /* Declare the following:
    /* - Parameters used to call stored procedure GETPRML
    /* - An SQLDA for DESCRIBE PROCEDURE
    /* - An SQLDA for DESCRIBE CURSOR
    /* - Result set variable locators for up to three result
    /* sets
    *****/
    EXEC SQL BEGIN DECLARE SECTION;
    char procnm[19];        /* INPUT parm -- PROCEDURE name */
    char schema[9];          /* INPUT parm -- User's schema */
    long int out_code;        /* OUTPUT -- SQLCODE from the
                                /* SELECT operation.
    char parmlst[255];        /* OUTPUT -- RUNOPTS values
                                /* for the matching row in
                                /* catalog table SYSROUTINES */
    struct indicators {
        short int procnm_ind;
        short int schema_ind;
        short int out_code_ind;
        short int parmlst_ind;
    } parmind;
    /* Indicator variable structure */

    struct sqlda *proc_da;      /* SQLDA for DESCRIBE PROCEDURE */
    struct sqlda *res_da;        /* SQLDA for DESCRIBE CURSOR */
    static volatile
        SQL TYPE IS RESULT_SET_LOCATOR *loc1, *loc2, *loc3;
        /* Locator variables */
    EXEC SQL END DECLARE SECTION;

    /*****
    /* Allocate the SQLDAs to be used for DESCRIBE
    /* PROCEDURE and DESCRIBE CURSOR. Assume that at most
    /* three cursors are returned and that each result set
    /* has no more than five columns.
    *****/
    proc_da = (struct sqlda *)malloc(SQLDASIZE(3));
    res_da = (struct sqlda *)malloc(SQLDASIZE(5));

    /*****
    /* Call the GETPRML stored procedure to retrieve the
    /* RUNOPTS values for the stored procedure. In this
    /* example, we request the PARMLIST definition for the
    /* stored procedure named DSN8EP2.
    /* The call should complete with SQLCODE +466 because
    /* GETPRML returns result sets.
    */
```

```

/***********************/
strcpy(procnm,"dsn8ep2");
/* Input parameter -- PROCEDURE to be found */
strcpy(schema,"");
/* Input parameter -- Schema name for proc */
parmind.procnm_ind=0;
parmind.schema_ind=0;
parmind.out_code_ind=0;
/* Indicate that none of the input parameters */
/* have null values */
parmind.parmlst_ind=-1;
/* The parmlst parameter is an output parm. */
/* Mark PARMLST parameter as null, so the DB2 */
/* requester does not have to send the entire */
/* PARMLST variable to the server. This */
/* helps reduce network I/O time, because */
/* PARMLST is fairly large. */

EXEC SQL
CALL GETPRML(:procnm INDICATOR :parmind.procnm_ind,
: schema INDICATOR :parmind.schema_ind,
:out_code INDICATOR :parmind.out_code_ind,
:parm1st INDICATOR :parmind.parm1st_ind);
if(SQLCODE!=+466) /* If SQL CALL failed,
{
    /* print the SQLCODE and any */
    /* message tokens */
printf("SQL CALL failed due to SQLCODE = %d\n",
      sqlca.sqlcode);
printf("sqlca.sqlerrmc = ");
for(i=0;i<sqlca.sqlerrm1;i++)
    printf("%c",sqlca.sqlerrmc[i]);
printf("\n");
}

else /* If the CALL worked,
if(out_code!=0) /* Did GETPRML hit an error?
printf("GETPRML failed due to RC = %d\n", out_code);
/*****************/
/* If everything worked, do the following: */
/* - Print out the parameters returned. */
/* - Retrieve the result sets returned. */
/*****************/
else
{
    printf("RUNOPTS = %s\n", parmlst);
    /* Print out the runopts list */

/*****************/
/* Use the statement DESCRIBE PROCEDURE to */
/* return information about the result sets in the */
/* SQLDA pointed to by proc_da: */
/* - SQLD contains the number of result sets that were */
/* returned by the stored procedure. */
/* - Each SQLVAR entry has the following information */
/*   about a result set: */
/*     - SQLNAME contains the name of the cursor that */
/*       the stored procedure uses to return the result */
/*       set. */
/*     - SQLIND contains an estimate of the number of */
/*       rows in the result set. */
/*     - SQLDATA contains the result locator value for */
/*       the result set. */
/*****************/
EXEC SQL DESCRIBE PROCEDURE INTO :*proc_da;
/*****************/
/* Assume that you have examined SQLD and determined */
/* that there is one result set. Use the statement */
/* ASSOCIATE LOCATORS to establish a result set locator */
/* for the result set. */
/*****************/
EXEC SQL ASSOCIATE LOCATORS (:loc1) WITH PROCEDURE GETPRML;

/*****************/
/* Use the statement ALLOCATE CURSOR to associate a */
/* cursor for the result set. */
/*****************/
EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :loc1;
/*****************/
/* Use the statement DESCRIBE CURSOR to determine the */
/* columns in the result set. */
*/

```

```

/*****************/
EXEC SQL DESCRIBE CURSOR C1 INTO :*res_da;

/*****************/
/* Call a routine (not shown here) to do the following: */
/* - Allocate a buffer for data and indicator values */
/*   fetched from the result table. */
/* - Update the SQLDATA and SQLIND fields in each */
/*   SQLVAR of *res_da with the addresses at which to */
/*   put the fetched data and values of indicator */
/*   variables. */
/*****************/
alloc_outbuff(res_da);

/*****************/
/* Fetch the data from the result table. */
/*****************/
while(SQLCODE==0)
    EXEC SQL FETCH C1 USING DESCRIPTOR :*res_da;
}
return;
}

```

Ejemplo C de procedimiento almacenado con una convención de vinculación GENERAL

Puede llamar a un procedimiento almacenado que utilice la convención de vinculación GENERAL desde un programa C.

Este ejemplo de procedimiento almacenado hace lo siguiente:

- Busca en la tabla SYSROUTINES del catálogo Db2 una fila que coincida con los parámetros de entrada del programa cliente. Los dos parámetros de entrada contienen valores para NOMBRE y ESQUEMA.
- Busca en la tabla SYSTABLES del catálogo Db2 todas las tablas en las que el valor de CREATOR coincide con el valor del parámetro de entrada SCHEMA. El procedimiento almacenado utiliza un cursor para devolver los nombres de las tablas.

La convención de vinculación utilizada para este procedimiento almacenado es GENERAL.

Los parámetros de salida de este procedimiento almacenado contienen el SQLCODE de la instrucción SELECT y el valor de la columna RUNOPTS de SYSROUTINES.

La declaración CREATE PROCEDURE para este procedimiento almacenado podría tener este aspecto:

```

CREATE PROCEDURE GETPRML(Procnm CHAR(18) IN, Schema CHAR(8) IN,
    Outcode INTEGER OUT, Parmlst VARCHAR(254) OUT)
LANGUAGE C
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME "GETPRML"
COLLID GETPRML
ASUTIME NO LIMIT
PARAMETER STYLE GENERAL
STAY RESIDENT NO
RUN OPTIONS "MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)"
WLM ENVIRONMENT SAMPPROG
PROGRAM TYPE MAIN
SECURITY DB2
RESULT SETS 2
COMMIT ON RETURN NO;

```

El siguiente ejemplo es un procedimiento almacenado C con convención de vinculación GENERAL

```

#pragma runopts(plist(os))
#include <stdlib.h>

EXEC SQL INCLUDE SQLCA;

/*****************/
/* Declare C variables for SQL operations on the parameters. */
/* These are local variables to the C program, which you must */
/* copy to and from the parameter list provided to the stored */

```

```

/* procedure. */ 
EXEC SQL BEGIN DECLARE SECTION;
char PROCNM[19];
char SCHEMA[9];
char PARMLST[255];
EXEC SQL END DECLARE SECTION;

/* Declare cursors for returning result sets to the caller. */
EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
  SELECT NAME
    FROM SYSIBM.SYSTABLES
   WHERE CREATOR=:SCHEMA;

main(argc,argv)
  int argc;
  char *argv[];
{
  /* Copy the input parameters into the area reserved in
   * the program for SQL processing. */
  strcpy(PROCNM, argv[1]);
  strcpy(SCHEMA, argv[2]);

  /* Issue the SQL SELECT against the SYSROUTINES
   * DB2 catalog table. */
  strcpy(PARMLST, ""); /* Clear PARMLST */
EXEC SQL
  SELECT RUNOPTS INTO :PARMLST
    FROM SYSIBM.ROUTINES
   WHERE NAME=:PROCNM AND
        SCHEMA=:SCHEMA;

  /* Copy SQLCODE to the output parameter list. */
  *(int *) argv[3] = SQLCODE;
  strcpy(argv[4], PARMLST);

  /* Open cursor C1 to cause DB2 to return a result set
   * to the caller. */
EXEC SQL OPEN C1;
}

```

Ejemplo de procedimiento almacenado C con una convención de vinculación GENERAL WITH NULLS

Puede llamar a un procedimiento almacenado que utilice la convención de vinculación GENERAL WITH NULLS desde un programa C.

Este ejemplo de procedimiento almacenado hace lo siguiente:

- Busca en la tabla SYSROUTINES del catálogo Db2 una fila que coincida con los parámetros de entrada del programa cliente. Los dos parámetros de entrada contienen valores para NAME y SCHEMA.
- Busca en la tabla SYSTABLES del catálogo Db2 todas las tablas en las que el valor de CREATOR coincide con el valor del parámetro de entrada SCHEMA. El procedimiento almacenado utiliza un cursor para devolver los nombres de las tablas.

La convención de vinculación para este procedimiento almacenado es GENERAL WITH NULLS.

Los parámetros de salida de este procedimiento almacenado contienen el SQLCODE de la operación SELECT y el valor de la columna RUNOPTS recuperado de la tabla SYSROUTINES.

La declaración CREATE PROCEDURE para este procedimiento almacenado podría tener este aspecto:

```
CREATE PROCEDURE GETPRML(Procnm CHAR(18) IN, Schema CHAR(8) IN,
    Outcode INTEGER OUT, Parmlst VARCHAR(254) OUT)
LANGUAGE C
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME "GETPRML"
COLLID GETPRML
ASUTIME NO LIMIT
PARAMETER STYLE GENERAL WITH NULLS
STAY RESIDENT NO
RUN OPTIONS "MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)"
WLM ENVIRONMENT SAMPPROG
PROGRAM TYPE MAIN
SECURITY DB2
RESULT SETS 2
COMMIT ON RETURN NO;
```

El siguiente ejemplo es un procedimiento almacenado C con la convención de vinculación GENERAL WITH NULLS.

```
#pragma runopts(plist(os))
#include <stdlib.h>

EXEC SQL INCLUDE SQLCA;

/*********************************************************************
/* Declare C variables used for SQL operations on the          */
/* parameters. These are local variables to the C program,      */
/* which you must copy to and from the parameter list provided */
/* to the stored procedure.                                       */
/*********************************************************************
EXEC SQL BEGIN DECLARE SECTION;
char Procnm[19];
char Schema[9];
char Parmlst[255];
struct Indicators {
    short int Procnm_Ind;
    short int Schema_Ind;
    short int Out_Code_Ind;
    short int Parmlst_Ind;
} PARM_Ind;
EXEC SQL END DECLARE SECTION;

/*********************************************************************
/* Declare cursors for returning result sets to the caller.    */
/*********************************************************************
EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
    SELECT Name
    FROM sysibm.systables
    WHERE Creator=:Schema;

main(argc,argv)
    int argc;
    char *argv[];
{
    /*********************************************************************
    /* Copy the input parameters into the area reserved in       */
    /* the local program for SQL processing.                      */
    /*********************************************************************
    strcpy(Procnm, argv[1]);
    strcpy(Schema, argv[2]);

    /*********************************************************************
    /* Copy null indicator values for the parameter list.     */
    /*********************************************************************
    memcpy(&Parm_Ind,(struct Indicators *) argv[5],
        sizeof(Parm_Ind));

    /*********************************************************************
    /* If any input parameter is NULL, return an error           */
    /* return code and assign a NULL value to Parmlst.          */
    /*********************************************************************
if (Parm_Ind.Procnm_Ind<0 ||
    Parm_Ind.Schema_Ind<0 || {
    *(int *) argv[3] = 9999; /* set output return code */
```

```

PARM_IND.OUT_CODE_IND = 0;      /* value is not NULL      */
PARM_IND.PARMLST_IND = -1;     /* PARMLST is NULL        */
}

else {
    /*****
    /* If the input parameters are not NULL, issue the SQL   */
    /* SELECT against the SYSIBM.SYSROUTINES catalog       */
    /* table.                                              */
    *****/
    strcpy(PARMLST, "");           /* Clear PARMLST          */
EXEC SQL
    SELECT RUNOPTS INTO :PARMLST
        FROM SYSIBM.SYSROUTINES
        WHERE NAME=:PROCNAME AND
            SCHEMA=:SCHEMA;
    /*****
    /* Copy SQLCODE to the output parameter list.          */
    *****/
    *(int *) argv[3] = SQLCODE;
    PARM_IND.OUT_CODE_IND = 0;      /* OUT_CODE is not NULL */
}

    /*****
    /* Copy the RUNOPTS value back to the output parameter */
    /* area.                                              */
    *****/
strcpy(argv[4], PARMLST);

    /*****
    /* Copy the null indicators back to the output parameter*/
    /* area.                                              */
    *****/
memcpy((struct INDICATORS *) argv[5],&PARM_IND,
sizeof(PARM_IND));

    /*****
    /* Open cursor C1 to cause DB2 to return a result set   */
    /* to the caller.                                         */
    *****/
EXEC SQL OPEN C1;
}

```

Definición del área de comunicaciones SQL, SQLSTATE y SQLCODE en C y C++

Los programas en C y C++ que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Acerca de esta tarea

Si especifica la opción de procesamiento SQL STDSQL(YES), no defina un SQLCA. Si lo hace, Db2 ignora su SQLCA y la definición de su SQLCA provoca errores en tiempo de compilación. Si especifica la opción de procesamiento SQL STDSQL(NO), incluya un SQLCA explícitamente.

Si su aplicación contiene instrucciones SQL y no incluye un área de comunicaciones SQL (SQLCA), debe declarar variables de host SQLCODE y SQLSTATE individuales. Su programa puede utilizar estas variables para comprobar si una instrucción SQL se ha ejecutado correctamente.

Procedimiento

Elija una de estas acciones:

Opción	Descripción
Para definir el área de comunicaciones SQL:	a. Codifique el SQLCA directamente en el programa o utilice la siguiente instrucción SQL INCLUDE para solicitar una declaración SQLCA estándar:

Opción	Descripción
	<pre>EXEC SQL INCLUDE SQLCA</pre> <p>La declaración estándar incluye tanto una definición de estructura como un área de datos estáticos denominada «sqlca».</p> <p>Db2 establece los valores SQLCODE y SQLSTATE en el SQLCA después de que se ejecute cada instrucción SQL. Su aplicación debe comprobar estos valores para determinar si la última instrucción SQL se ha realizado correctamente.</p>
Para declarar las variables de host SQLCODE y SQLSTATE:	<p>a. Declare la variable SQLCODE dentro de una declaración BEGIN DECLARE SECTION y una declaración END DECLARE SECTION en sus declaraciones de programa como un entero largo:</p> <pre>long SQLCODE;</pre> <p>b. Declare la variable SQLSTATE dentro de una instrucción BEGIN DECLARE SECTION y una instrucción END DECLARE SECTION en las declaraciones de su programa como una matriz de caracteres de longitud 6:</p> <pre>char SQLSTATE[6];</pre> <p>Restricción: No declare una variable SQLSTATE como un elemento de una estructura.</p> <p>Requisito: Después de declarar las variables SQLCODE y SQLSTATE, asegúrese de que todas las sentencias SQL del programa estén dentro del ámbito de la declaración de estas variables.</p>

Tareas relacionadas

[Comprobación de la ejecución de sentencias SQL](#)

Después de ejecutar una sentencia SQL, el programa debe comprobar si hay errores antes de confirmar los datos y manejar los errores que representan.

[Comprobación de la ejecución de sentencias SQL utilizando SQLCA](#)

Un modo de comprobar si una sentencia SQL se ha ejecutado correctamente es utilizar el área de comunicación SQL (SQLCA). Esta área se distingue de la comunicación con Db2.

[Comprobación de la ejecución de sentencias SQL utilizando SQLCODE y SQLSTATE](#)

Siempre que se ejecuta una sentencia SQL, los campos SQLCODE y SQLSTATE de SQLCA reciben un código de retorno.

[Definición de elementos que su programa puede utilizar para comprobar si una sentencia SQL se ha ejecutado correctamente](#)

Si su programa contiene sentencias SQL, el programa debe definir determinada infraestructura para poder comprobar si las sentencias se han ejecutado correctamente. También puede incluir un área de comunicación SQL (SQLCA), que contenga variables SQLCODE y SQLSTATE, o declarar variables host SQLCODE y SQLSTATE.

Definición de áreas de descriptor de SQL (SQLDA) en C y C++

Si su programa incluye determinadas sentencias SQL, debe definir al menos un área de descriptor SQL (SQLDA). Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Procedimiento

Codifique el SQLDA directamente en el programa, o utilice la siguiente instrucción SQL INCLUDE para solicitar una declaración SQLDA estándar:

```
EXEC SQL INCLUDE SQLDA
```

Puede colocar una declaración SQLDA donde C permita una definición de estructura. Se aplican las reglas normales de alcance de C. La declaración estándar incluye solo una definición de estructura con el nombre sqlda.

Restricción: Debe colocar las declaraciones SQLDA antes de la primera instrucción SQL que haga referencia al descriptor de datos, a menos que utilice la opción de procesamiento TWOPASS SQL.

Tareas relacionadas

[Definición de áreas de descriptor de SQL \(SQLDA\)](#)

Si su programa incluye ciertas sentencias SQL, debe definir al menos *un área de descriptor SQL (SQLDA)*. Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Referencia relacionada

[Área de descriptor de SQL \(SQLDA\) \(Db2 SQL\)](#)

Declaración de variables de host y variables de indicador en C y C++

Puede utilizar variables de host, matrices de variables de host y estructuras de host en instrucciones SQL en su programa para pasar datos entre Db2 y su aplicación.

Procedimiento

Para declarar variables de host, matrices de variables de host y estructuras de host:

1. Declare las variables de acuerdo con las siguientes reglas y directrices:

- Puede tener más de una sección de declaración de variables de host en su programa.
- Puede utilizar miembros de clase como variables de host. Los miembros de clase que se utilizan como variables de host son accesibles para cualquier instrucción SQL dentro de la clase. Sin embargo, no puede utilizar objetos de clase como variables de host.
- Si especifica la opción de procesamiento ONEPASS SQL, debe declarar explícitamente cada variable de host y cada matriz de variables de host antes de utilizarlas en una instrucción SQL. Si especifica la opción del precompilador TWOPASS, debe declarar cada variable de host antes de usarla en la instrucción DECLARE CURSOR.

Restricción: Db2 coprocessorDb2 C/C++ solo admite la opción ONEPASS.

- Si especifica la opción de procesamiento SQL STDSQL(YES), debe preceder las sentencias del lenguaje del host que definen las variables del host y las matrices de variables del host con la sentencia BEGIN DECLARE SECTION y seguir las sentencias del lenguaje del host con la sentencia END DECLARE SECTION. De lo contrario, estas declaraciones son opcionales.
- Asegúrese de que cualquier instrucción SQL que utilice una variable de host o una matriz de variables de host esté dentro del ámbito de la instrucción que declara esa variable o matriz.
- Si utiliza el lenguaje de programación C (Db2 precompilador), asegúrese de que los nombres de las variables de host y de las matrices de variables de host sean únicos dentro del programa, incluso si las variables y las matrices de variables se encuentran en bloques, clases, procedimientos, funciones o subrutinas diferentes. Puede calificar los nombres con un nombre de estructura para hacerlos únicos.

2. Opcional: Definir cualquier variable, matriz y estructura de indicadores asociados.

Tareas relacionadas

[Declaración de variables host y variables de indicación](#)

Puede utilizar variables host y variables de indicación en sentencias SQL del programa para pasar datos entre Db2 y la aplicación.

Variables de host en C y C++

En programas C y C++, puede especificar variables host numéricas, de caracteres, gráficas, binarias, LOB, XML y ROWID. También puede especificar conjuntos de resultados, tabla y localizadores LOB y variables de referencia de archivos LOB y XML.

Restricciones:

- Solo algunas de las declaraciones C válidas son declaraciones de variables de host válidas. Si la declaración de una variable no es válida, cualquier instrucción SQL que haga referencia a la variable podría dar lugar al mensaje UNDECLARED HOST VARIABLE.
- C admite algunos tipos de datos y clases de almacenamiento sin equivalentes SQL, como la clase de almacenamiento de registros, `typedef` y `long long`.
- Los siguientes tipos de datos de localización son tipos de datos SQL especiales que no tienen equivalentes en C:
 - Localizador de conjunto de resultados
 - Localizador de tablas
 - localizadores de LOB

No puede utilizarlos para definir tipos de columna.

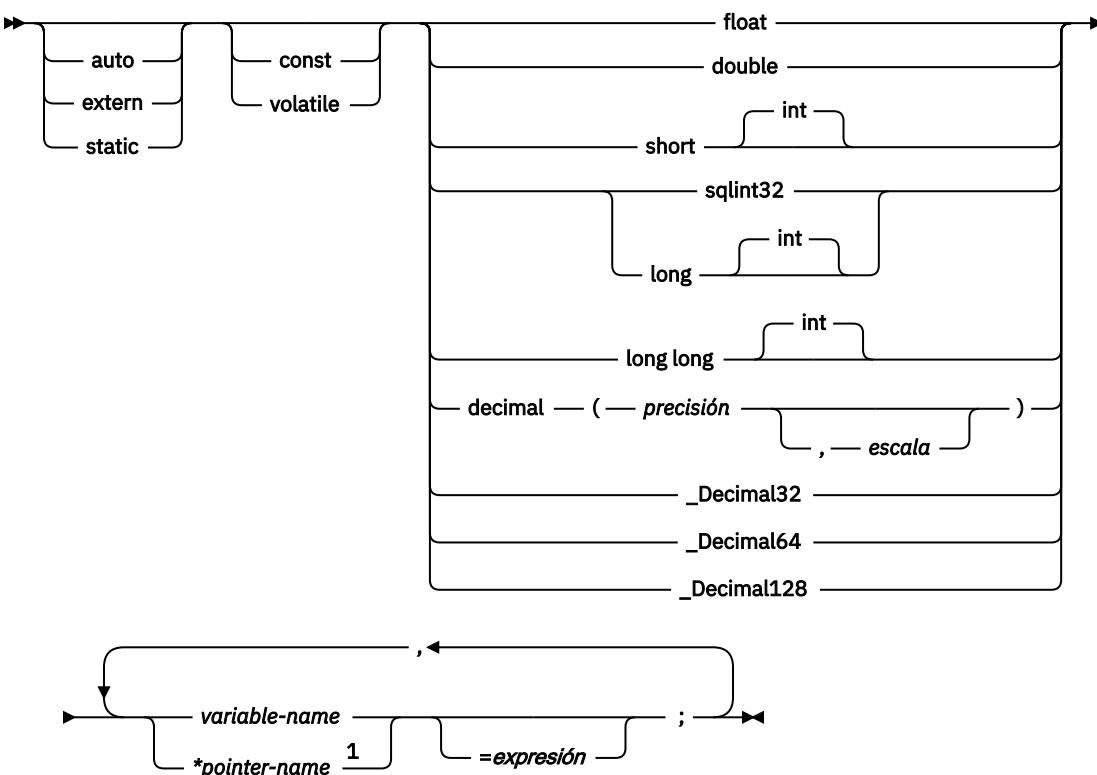
- Aunque Db2 le permite utilizar L-literales formados correctamente en programas de aplicación C, Db2 no comprueba todas las restricciones que el compilador C impone al L-literal. \
- No utilice L-literales en las sentencias SQL. Utilice constantes de cadena gráfica de tipo "Db2" en las sentencias SQL para trabajar con el literal L.

Recomendaciones:

- Tenga cuidado con el desbordamiento. Por ejemplo, supongamos que recupera un valor de columna INTEGER en una variable de host entera corta, y el valor de columna es mayor que 32767. Recibirá una advertencia de desbordamiento o un error, dependiendo de si proporciona una variable indicadora.
- Tenga cuidado con el truncamiento. Asegúrese de que la variable de host que declara puede contener los datos y un terminador NUL, si es necesario. Al recuperar un valor de columna de punto flotante o decimal en una variable de host entera larga, se elimina cualquier parte fraccionaria del valor.

Variables numéricas del host

El siguiente diagrama muestra la sintaxis para declarar variables de host numéricas.



Notas:

- ¹ Si utiliza la notación de puntero de la variable host, debe utilizar el coprocesador de la biblioteca de funciones de C (Db2).

Restricciones:

- Si su compilador C no tiene un tipo de datos decimal, no existe un equivalente exacto para el tipo de datos SQL DECIMAL. En este caso, puede utilizar una de las siguientes variables o técnicas para manejar valores decimales:
 - Una variable entera o de punto flotante, que convierte el valor. Si utiliza una variable entera, perderá la parte fraccionaria del número. Si el número decimal puede exceder el valor máximo de un número entero o si desea conservar un valor fraccionario, utilice variables de punto flotante. Los números de coma flotante son aproximaciones de números reales. Por lo tanto, cuando asignas un número decimal a una variable de punto flotante, el resultado puede ser diferente del número original.
 - Una variable de host de cadena de caracteres. Utilice la función CHAR para obtener una representación de cadena de un número decimal.
 - La función DECIMAL para convertir explícitamente un valor a un tipo de datos decimal, como se muestra en el siguiente ejemplo:

```

long duration=10100; /* 1 year and 1 month */
char result_dt[11];

EXEC SQL SELECT START_DATE + DECIMAL(:duration,8,0)
  INTO :result_dt FROM TABLE1;
  
```

- z/OS 1.10 o superior (z/OS V1R10 XL C/C++) para utilizar el tipo de datos de host de punto flotante decimal.
- El tipo de datos de host especial C only 'complex floating-point' no es un tipo admitido para la variable de host.
- La opción del precompilador FLOAT no se aplica a los tipos de variables de punto flotante decimal del host.

- Para utilizar variables de host de punto flotante decimal, debe utilizar el coprocesador de punto flotante de coma flotante (Db2).

Para los tipos de datos de punto flotante, utilice la opción de procesamiento FLOAT SQL para especificar si la variable host está en formato de punto flotante binario IEEE o de punto flotante hexadecimal de arquitectura z. Db2 no comprueba si el formato del contenido de la variable host coincide con el formato que especificó con la opción de procesamiento FLOAT SQL. Por lo tanto, debe asegurarse de que el contenido de la variable de host de punto flotante coincida con el formato que especificó con la opción de procesamiento FLOAT SQL. Db2 convierte todos los datos de entrada de punto flotante al formato de punto flotante hexadecimal de z/Architecture antes de almacenarlos.

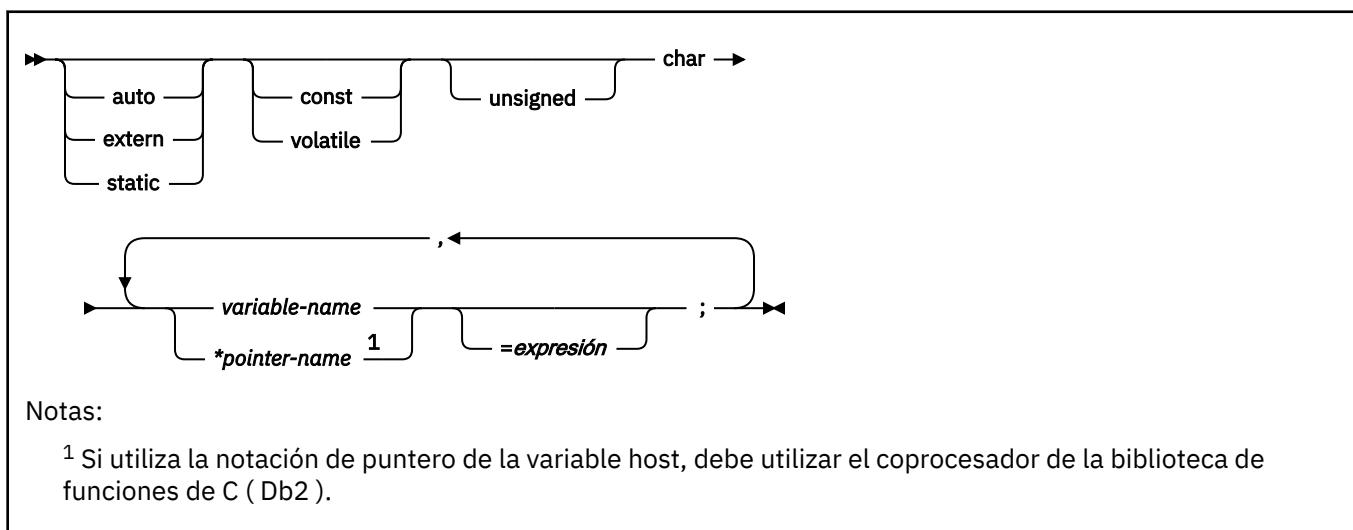
Variabes de host de caracteres

Puede especificar las siguientes formas de variables de host de caracteres:

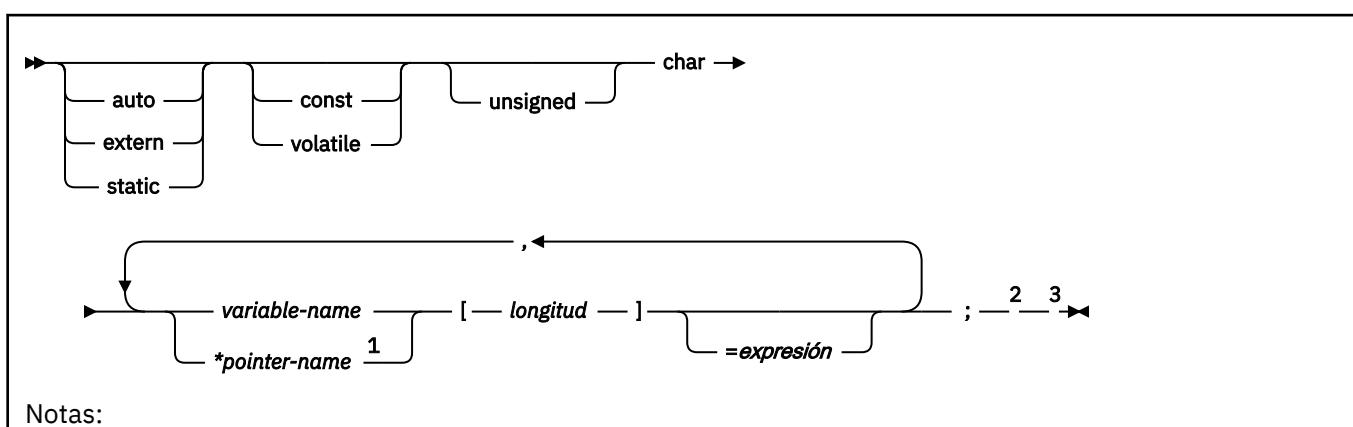
- Formulario de un solo carácter
- Formulario de caracteres terminados en NUL
- Formulario estructurado VARCHAR
- CLOB

Los siguientes diagramas muestran la sintaxis para formularios distintos de CLOB.

El siguiente diagrama muestra la sintaxis para declarar variables de host de un solo carácter.

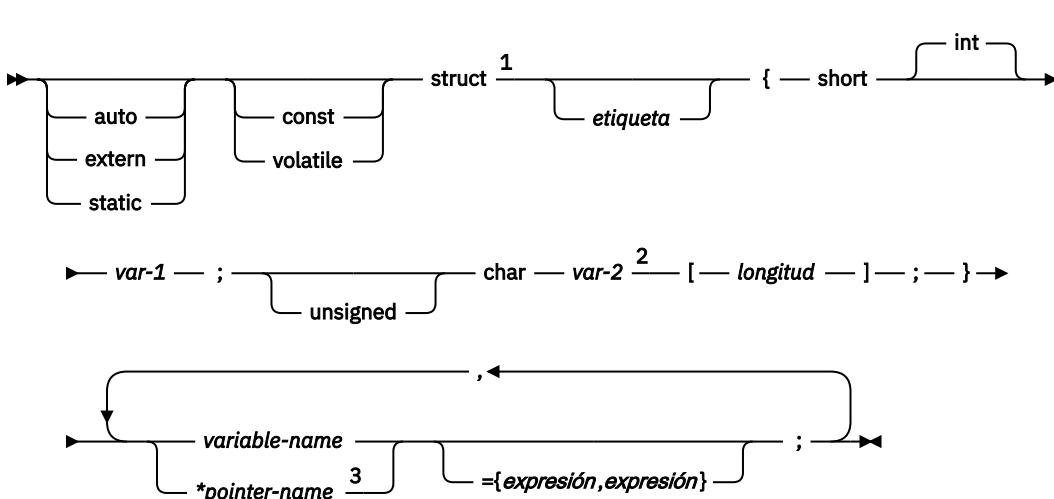


El siguiente diagrama muestra la sintaxis para declarar variables de host de caracteres terminadas en NUL.



- ¹ Si utiliza la notación de puntero de la variable host, debe utilizar el coprocesador de la biblioteca de funciones de C (Db2).
- ² Cualquier cadena que se asigne a esta variable debe terminar en NUL. Cualquier cadena que se recupere de esta variable terminará en NUL.
- ³ Una variable de host de caracteres terminada en NUL se asigna a una cadena de caracteres de longitud variable (excepto para el NUL).

El siguiente diagrama muestra la sintaxis para declarar variables de host de caracteres de longitud variable que utilizan la forma estructurada VARCHAR.



Notas:

- ¹ Puede utilizar la etiqueta struct para definir otras variables, pero no puede utilizarlas como variables de host en SQL.
- ² No puede utilizar var-1 y var-2 como variables de host en una instrucción SQL.
- ³ Si utiliza la notación de puntero de la variable host, debe utilizar el coprocesador de la biblioteca de funciones de C (Db2).

Ejemplo

El siguiente código de ejemplo muestra declaraciones válidas e inválidas de la forma estructurada VARCHAR:

```
EXEC SQL BEGIN DECLARE SECTION;
/* valid declaration of host variable VARCHAR vstring */
struct VARCHAR {
    short len;
    char s[10];
} vstring;
/* invalid declaration of host variable VARCHAR wstring */
struct VARCHAR wstring;
```

Para las variables de host de cadena terminadas en NUL, utilice las opciones de procesamiento SQL PADNTSTR y NOPADNTSTR para especificar si la variable debe rellenarse con espacios en blanco. La opción que especifique determina dónde se coloca el terminador NUL.

Si asigna una cadena de longitud *n* a una variable de host de cadena terminada en NUL, la variable tendrá uno de los valores que se muestran en la siguiente tabla.

Tabla 97. Valor de una variable de host de cadena terminada en NUL a la que se le asigna una cadena de longitud n

Longitud de la variable de host de cadena terminada en NUL	Valor de la variable
Menor o igual que n	<p>La cadena de origen hasta una longitud de $n-1$ y un NUL al final de la cadena.¹</p> <p>Db2 establece SQLWARN[1] en W y cualquier variable indicadora que proporcione a la longitud original de la cadena de origen.</p>
Igual a $n+1$	<p>La cadena de origen y un NUL al final de la cadena. ¹</p>
Mayor que $n+1$ y el origen es una cadena de longitud fija	<p>Si PADNTSTR está en vigor La cadena de origen, espacios en blanco para llenar el valor y un NUL al final de la cadena.</p> <p>Si NOPADNTSTR está en vigor La cadena de origen y un NUL al final de la cadena.</p>
Mayor que $n+1$ y el origen es una cadena de longitud variable	<p>La cadena de origen y un NUL al final de la cadena. ¹</p>

Nota:

- 1. En estos casos, es irrelevante si NOPADNTSTR o PADNTSTR está en vigor.

Restricción: Si utiliza el precompilador Db2 , no puede utilizar una variable de host que tenga la forma terminada en NUL en una instrucción PREPARE o DESCRIBE. Sin embargo, si utiliza el coprocesador Db2 , puede utilizar variables de host de la forma terminada en NUL en las sentencias PREPARE, DESCRIBE y EXECUTE IMMEDIATE.

Variables gráficas de host

Puede especificar las siguientes formas de variables de host gráficas:

- Formulario de gráfico único
- Forma gráfica terminada en NUL
- Formulario estructurado VARGRAPHIC.
- DBCLOB

Recomendación: En lugar de utilizar el tipo de datos C wchar_t para definir variables host gráficas y vargráficas, utilice una de las siguientes técnicas:

- Defina el tipo de datos sqldbchar utilizando la siguiente declaración typedef:

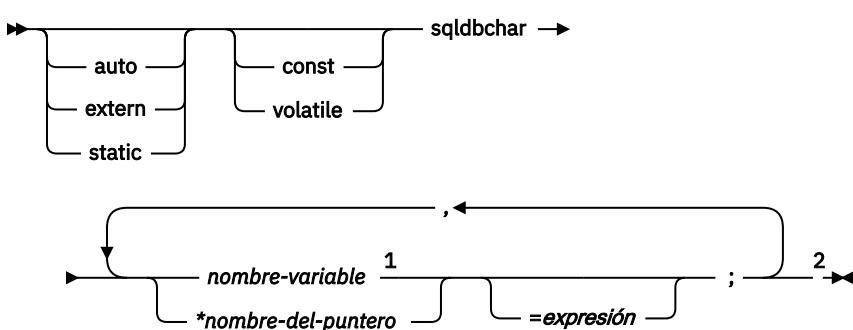
```
typedef unsigned short sqldbchar;
```

- Utilice el tipo de datos sqldbchar que se define en la instrucción typedef en uno de los siguientes archivos o bibliotecas:
 - Biblioteca SQL, sql.h
 - Db2 Biblioteca CLI, sqlcli.h
 - Archivo SQLUDF en el conjunto de datos DSN1210. SDSNC.H
- Utiliza el tipo de datos C sin signo corto.

El uso de sqldbchar o unsigned short le permite manipular datos DBCS y Unicode UTF-16 en el mismo formato en el que se almacenan en Db2. El uso de sqldbchar también facilita la migración de aplicaciones a otras plataformas.

Los siguientes diagramas muestran la sintaxis para formularios distintos de DBCLOB.

El siguiente diagrama muestra la sintaxis para declarar variables de host de un solo gráfico.

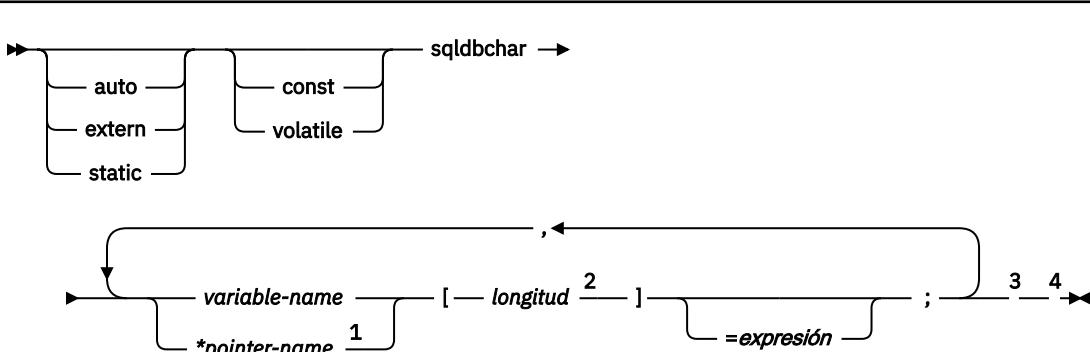


Notas:

¹ No puede utilizar notación de matriz en *variable-nombre*.

² El formulario de gráfico único declara una cadena gráfica de longitud fija de longitud 1.

El siguiente diagrama muestra la sintaxis para declarar variables de host gráficas terminadas en NUL.



Notas:

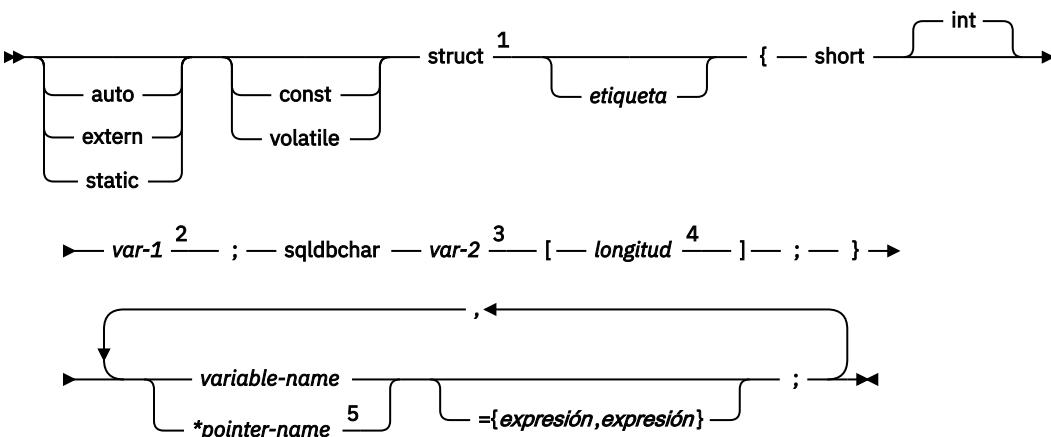
¹ Si utiliza la notación de puntero de la variable host, debe utilizar el coprocesador de la biblioteca de funciones de C (Db2).

² *longitud* debe ser una constante entera decimal mayor que 1 y no mayor que 16352.

³ Cualquier cadena que se asigne a esta variable debe terminar en NUL. Cualquier cadena que se recupere de esta variable terminará en NUL.

⁴ La forma gráfica terminada en NUL no acepta caracteres de un solo byte para la variable.

El siguiente diagrama muestra la sintaxis para declarar variables de host gráficas que utilizan la forma estructurada VARGRAPHIC.



Notas:

¹ Puede utilizar la etiqueta `struct` para definir otras variables, pero no puede utilizarlas como variables de host en SQL.

² `var-1` debe ser menor o igual que `length`.

³ No puede utilizar `var-1` o como variables de host en una instrucción SQL `var-2` como variables de host en una instrucción SQL.

⁴ `longitud` debe ser una constante entera decimal mayor que 1 y no mayor que 16352.

⁵ Si utiliza la notación de puntero de la variable host, debe utilizar el coprocesador de memoria compartida (Db2).

Ejemplo

El siguiente ejemplo muestra declaraciones válidas e inválidas de variables de host gráficas que utilizan la forma estructurada VARGRAPHIC:

```
EXEC SQL BEGIN DECLARE SECTION;
/* valid declaration of host variable structured vgraph */
struct VARGRAPH {
    short len;
    sqldbchar d[10];
} vgraph;
/* invalid declaration of host variable structured wgraph */
struct VARGRAPH wgraph;
```

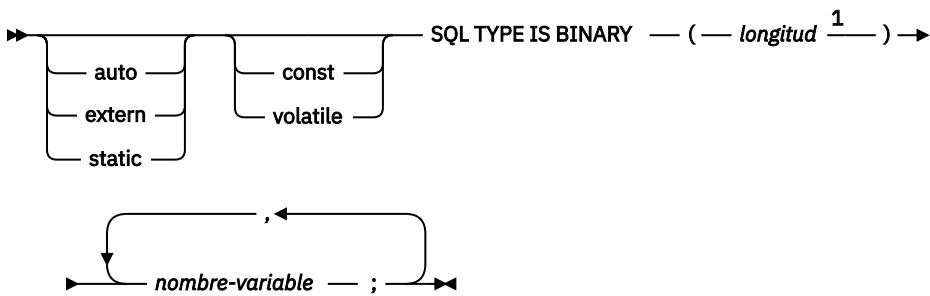
Variables binarias de host

Puede especificar las siguientes formas de variables de host binarias:

- Series de longitud fija
- Series de longitud variable
- BLOB

Los siguientes diagramas muestran la sintaxis para formularios que no sean BLOB.

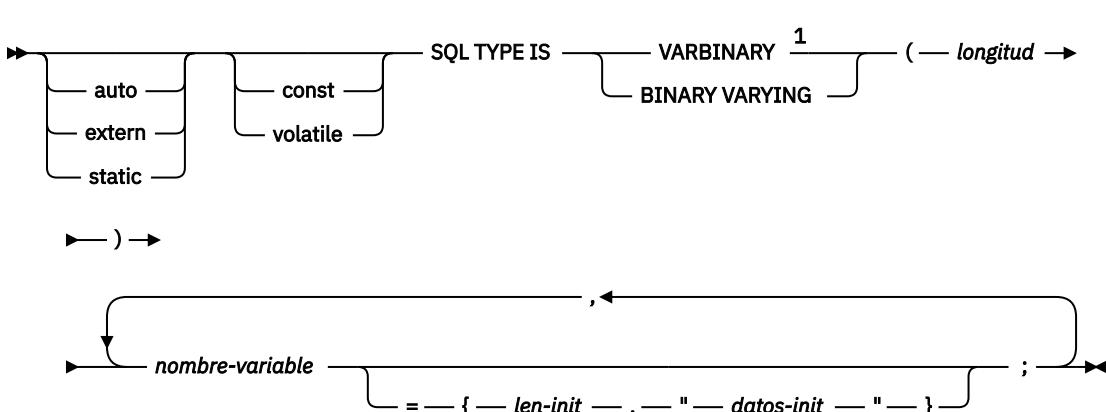
El siguiente diagrama muestra la sintaxis para declarar variables de host binarias.



Notas:

- ¹ La longitud debe ser un valor en el rango 1-255.

El siguiente diagrama muestra la sintaxis para declarar variables de host VARBINARY.



Notas:

- ¹ Para las variables de host VARBINARY, la longitud debe estar en el rango de 1 a 32704.

El lenguaje C no tiene variables que se correspondan con los tipos de datos binarios SQL BINARY y VARBINARY. Para crear variables de host que puedan utilizarse con estos tipos de datos, utilice la cláusula SQL TYPE IS. El precompilador SQL reemplaza esta declaración con la estructura del lenguaje C en el miembro de origen de salida.

Cuando haces referencia a una variable de host BINARIA o VARBINARIA en una instrucción SQL, debes utilizar la variable que especifiques en la declaración SQL TYPE. Cuando haga referencia a la variable de host en una instrucción de lenguaje de host, debe utilizar la variable que genera Db2 .

Ejemplos de declaraciones de variables binarias

La siguiente tabla muestra ejemplos de variables que genera Db2 cuando se declaran variables de host binarias.

Tabla 98. Ejemplos de declaraciones de variables BINARIAS y VARBINARIAS para C

Declaración variable que se incluye en el programa C	Variable correspondiente que genera Db2 en el miembro de origen de salida
SQL TYPE IS BINARY(10) bin_var;	char bin_var[10]

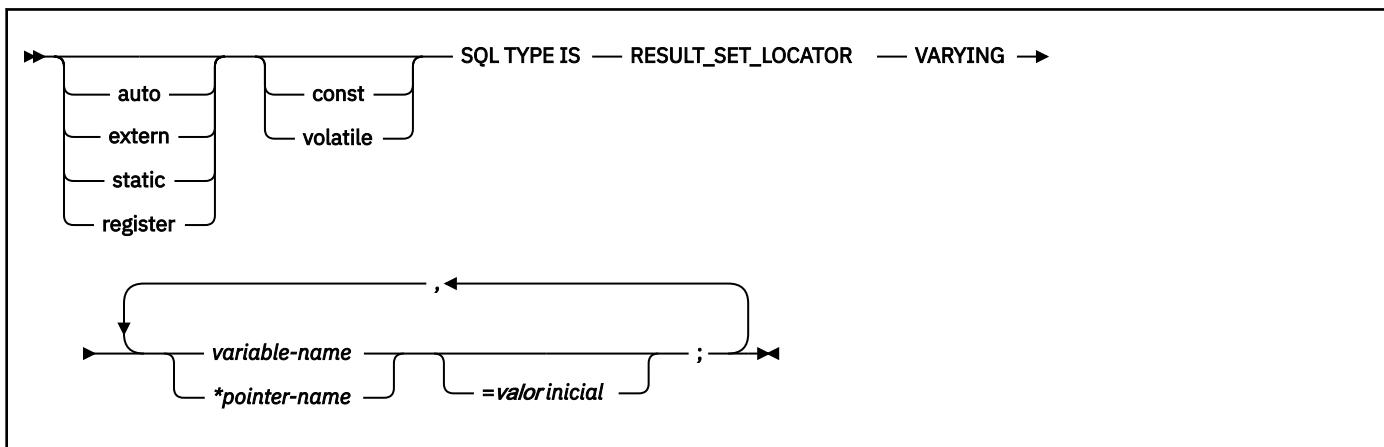
Tabla 98. Ejemplos de declaraciones de variables BINARIAS y VARBINARIAS para C (continuación)

Declaración variable que se incluye en el programa C	Variable correspondiente que genera Db2 en el miembro de origen de salida
SQL TYPE IS VARBINARY(10) vbin_var;	struct { short length; char data[10]; } vbin_var;

Recomendación: Tenga cuidado cuando utilice variables de host binarias con C y C++. La declaración SQL TYPE para BINARY y VARBINARY no tiene en cuenta el terminador NUL que espera C, porque las cadenas binarias no son cadenas terminadas en NUL. Además, la variable binaria host puede contener ceros en cualquier punto de la cadena.

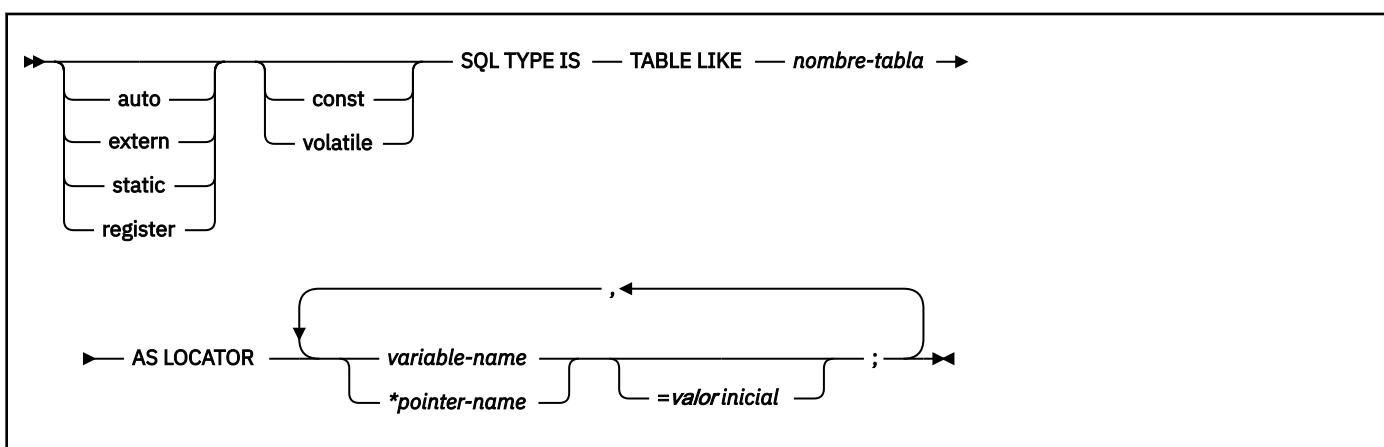
Localizadores de conjuntos de resultados

El siguiente diagrama muestra la sintaxis para declarar localizadores de conjuntos de resultados.



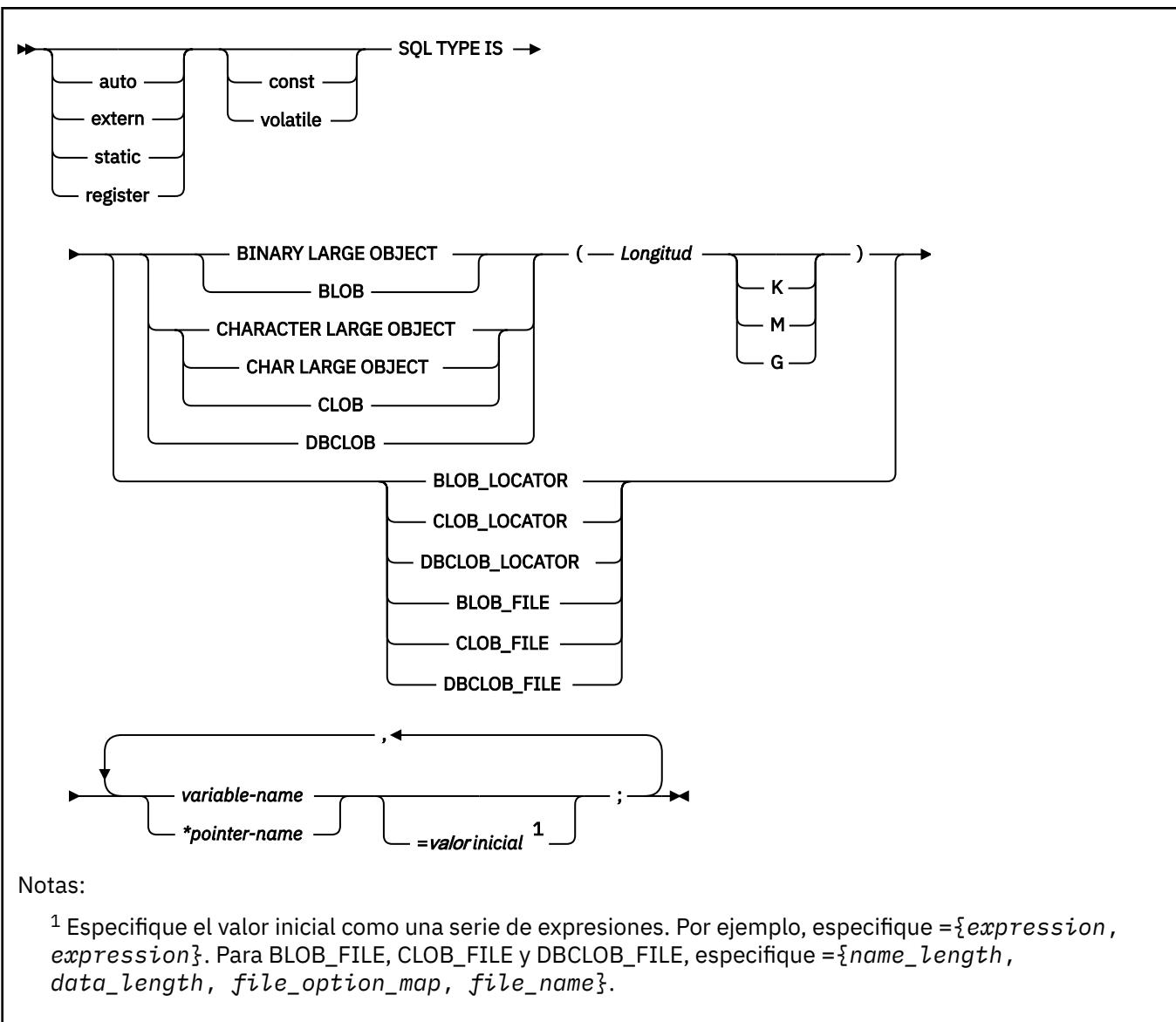
Localizadores de mesas

El siguiente diagrama muestra la sintaxis para declarar localizadores de tablas.



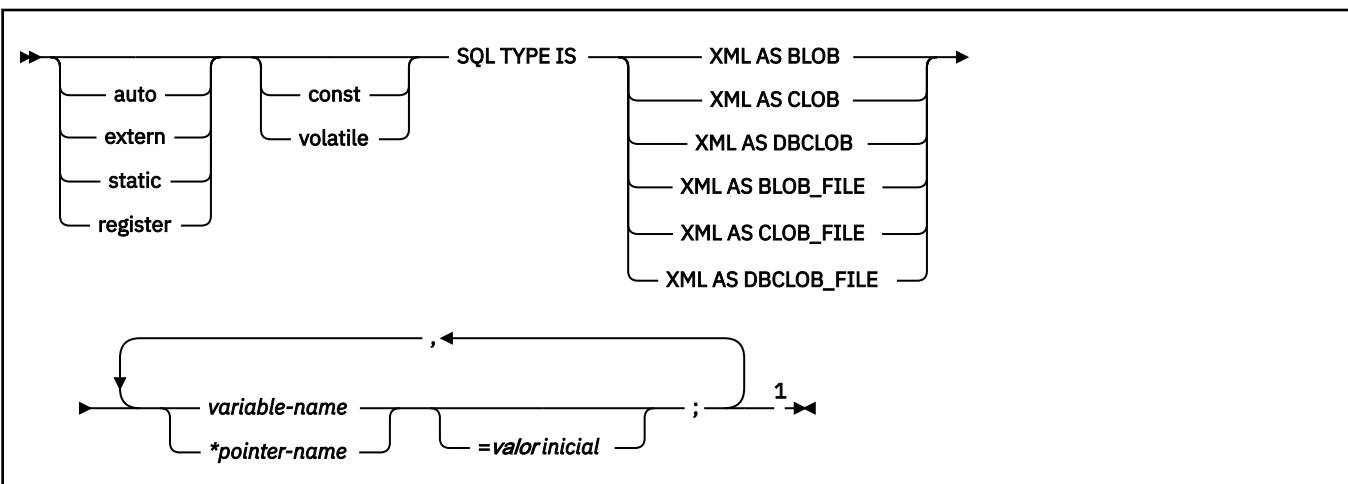
Variables LOB, localizadores y variables de referencia de archivos

El siguiente diagrama muestra la sintaxis para declarar variables de host BLOB, CLOB y DBCLOB, localizadores y variables de referencia de archivos.



Variables de referencia de archivos y host de datos XML

El siguiente diagrama muestra la sintaxis para declarar variables de host BLOB, CLOB y DBCLOB y variables de referencia de archivo para tipos de datos XML.

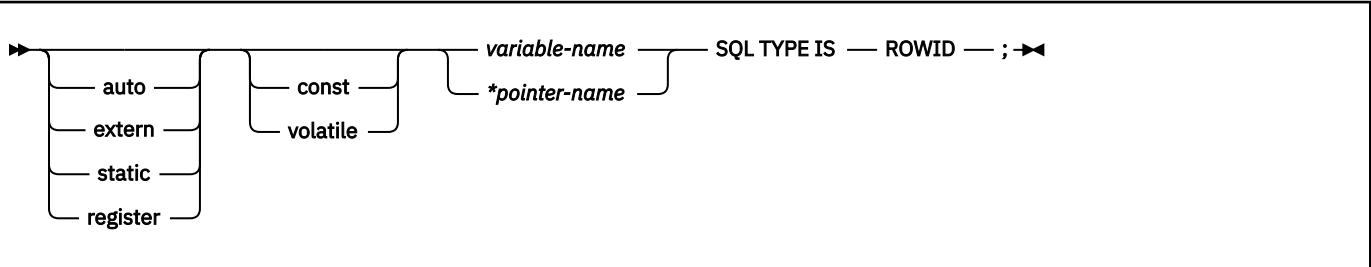


Notas:

¹ Especifique el valor inicial como una serie de expresiones. Por ejemplo, especifique ={expression, expression}. Para BLOB_FILE, CLOB_FILE y DBCLOB_FILE, especifique ={name_length, data_length, file_option_map, file_name}.

Variables de host ROWID

El siguiente diagrama muestra la sintaxis para declarar variables de host ROWID.



Constantes

La sintaxis de las constantes en los programas C y C++ difiere de la sintaxis de las constantes en las sentencias SQL de las siguientes maneras:

- C/C++ utiliza varias formas para literales numéricos (los posibles sufijos son: ll, LL, u, U, f, F, l, L, df, DF, dd, DD, dl, DL, d, D). Por ejemplo, en C/C++:
 - 4850976 es un literal decimal
 - 0x4bD es un literal entero hexadecimal
 - 03245 es un literal entero octal
 - 3.2E+4 es un literal de doble punto flotante
 - 3.2E+4f es un flotante literal de punto flotante
 - 3.2E+4l es un literal largo de doble punto flotante
 - 0x4bDP+4 es un literal de punto flotante hexadecimal doble
 - 22.2df es un literal de punto flotante decimal e _Decimal32
 - 0.00D es un literal decimal de punto fijo (z/OS solo cuando se especifica LANGLVL(EXTENDED))
- Utilice la forma literal C/C++ solo fuera de las sentencias SQL. En las sentencias SQL, utilice constantes numéricas.
- En C, las constantes de caracteres y las constantes de cadenas pueden utilizar secuencias de escape. No puede utilizar las secuencias de escape en las sentencias SQL.
- Los apóstrofos y las comillas tienen significados diferentes en C y SQL. En C, puede utilizar comillas dobles para delimitar constantes de cadena y apóstrofos para delimitar constantes de caracteres.

Ejemplo: Uso de comillas en C

```
printf( "%d lines read. \n", num_lines);
```

Ejemplo: Uso de apóstrofos en C

```
#define NUL '\0'
```

En SQL, puede utilizar comillas dobles para delimitar identificadores y apóstrofos para delimitar constantes de cadena.

Ejemplo: comillas en SQL

```
SELECT "COL#1" FROM TBL1;
```

Ejemplo: apóstrofes en SQL

```
SELECT COL1 FROM TBL1 WHERE COL2 = 'BELL';
```

- Los datos de caracteres en SQL son distintos de los datos enteros. Los datos de caracteres en C son un subtipo de datos enteros.

Conceptos relacionados

Variables de host

Utilice variables host para pasar un único elemento de datos entre Db2 y la aplicación.

Uso de variables host en sentencias SQL

Utilice variables de host escalares en sentencias de SQL incorporado para representar un único valor.

Las variables host son útiles para almacenar datos recuperados o para pasar valores que van a asignar o utilizar las comparaciones.

Tareas relacionadas

Determinación de si un valor recuperado en una variable host es nulo o está truncado

Antes de que su aplicación manipule los datos recuperados de Db2 en una variable host, determine si el valor es nulo. Determine también si se han truncado al asignarse a la variable. Puede utilizar las variables indicadores para obtener esta información.

Inserción de una sola fila utilizando una variable host

Utilice variables host en la sentencia INSERT cuando no conoce al menos algunos de los valores a insertar hasta que se ejecuta el programa.

Inserción de valores nulos en columnas utilizando las matrices o variables indicadoras

Si necesita insertar valores nulos en una columna, utilizar una matriz o variable indicadora es una forma fácil de hacerlo. Una variable o matriz de indicador está asociada a una variable de host o matriz de variables de lenguaje de host determinada.

Almacenamiento de datos LOB en tablas de Db2

Db2 maneja los datos LOB de manera diferente a otros tipos de datos. Como resultado, a veces es necesario realizar acciones adicionales al definir columnas LOB e insertar los datos LOB.

Recuperación de una única fila de datos en variables host

Si sabe que la consulta devuelve una sola fila, puede especificar una o más variables host que contienen los valores de columna de la fila recuperada.

Recuperación de una única fila de datos en una estructura de host

Si sabe que la consulta devuelve varios valores de columna para una única fila, puede especificar una estructura de host para contener los valores de columna.

Actualización de datos utilizando variables de host

Cuando quiera actualizar un valor en una tabla Db2, pero no conoce el valor exacto hasta que el programa se ejecute, utilice variables host. Db2 puede cambiar un valor de tabla para que coincida con el valor de la variable de host.

Referencia relacionada

Descripciones de opciones de proceso de SQL

Puede especificar cualquier opción de proceso de SQL tanto si utiliza el precompilador de Db2 como si utiliza el coprocesador de Db2. Sin embargo, es probable que el coprocesador de Db2 omita ciertas opciones ya que existen opciones del compilador del lenguaje principal que proporcionan la misma información.

Matrices de variables de host en C y C++

En los programas C y C++, puede especificar matrices de variables de host de tipo numérico, carácter, gráfico, binario, LOB, XML y ROWID. También puede especificar localizadores LOB y variables de referencia de archivos LOB y XML.

Solo se puede hacer referencia a las matrices de variables de lenguaje principal como una referencia simple en los contextos siguientes. En los diagramas de sintaxis, *host-variable-array* designa una referencia a una matriz de variables de host.

- En una sentencia `FETCH` para una captación de varias filas. Consulte [FETCH declaración \(Db2 SQL\)](#).
- En la forma `FOR n ROWS` de la sentencia `INSERT` con una matriz de variables de host para los datos de origen. Consulte [INSERT declaración \(Db2 SQL\)](#).
- En una sentencia `MERGE` con varias filas de datos de origen. Consulte [MERGE declaración \(Db2 SQL\)](#).
- En una sentencia `EXECUTE` para proporcionar un valor para un marcador de parámetro en una forma dinámica `FOR n ROWS` de la sentencia `INSERT` o una sentencia `MERGE`. Consulte [EXECUTE declaración \(Db2 SQL\)](#).

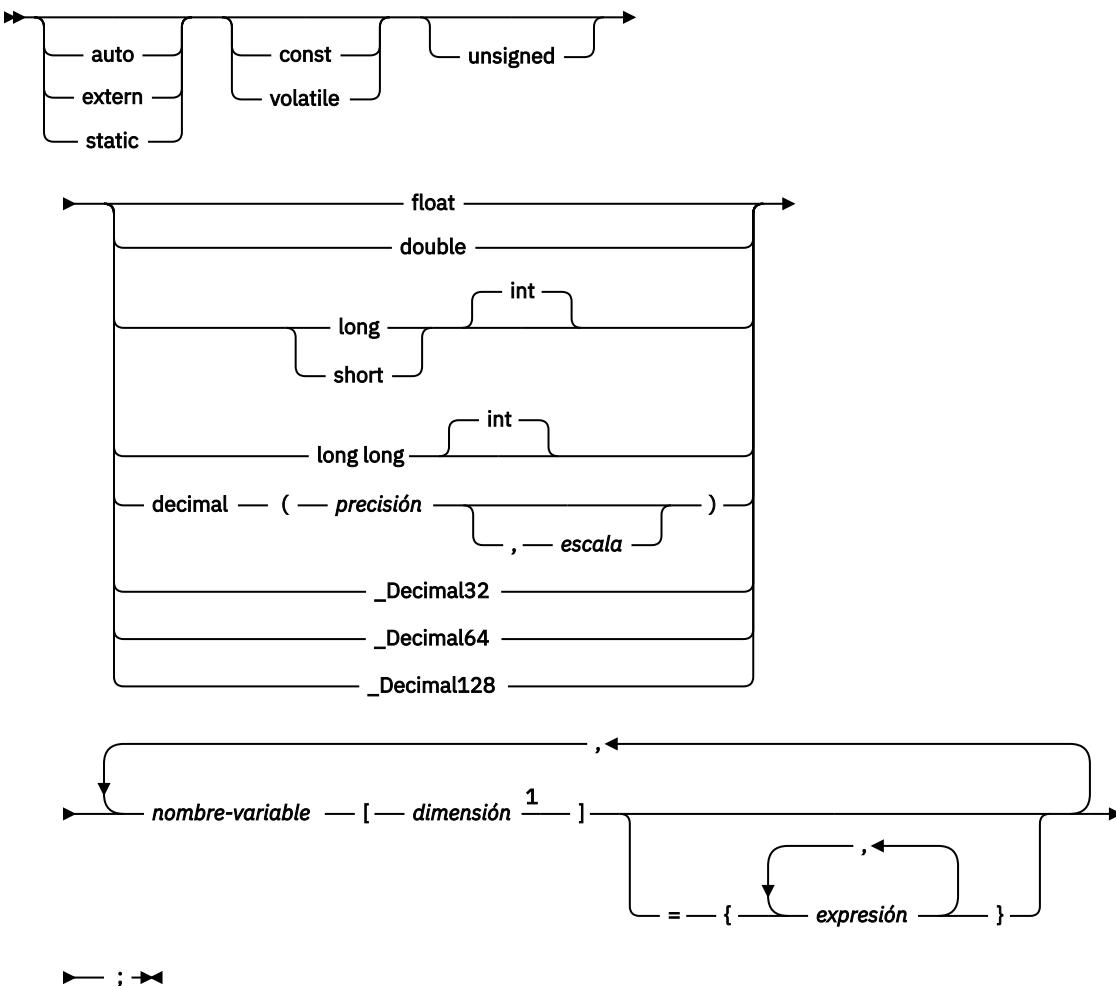
Restricciones:

- Solo algunas de las declaraciones C válidas son declaraciones de matriz de variables de host válidas. Si la declaración de una matriz variable no es válida, cualquier instrucción SQL que haga referencia a la matriz variable podría dar como resultado el mensaje UNDECLARED HOST VARIABLE ARRAY.
- Tanto para C como para C++, no se puede especificar el atributo `_packed` en las declaraciones de estructura para las siguientes matrices que se utilizan en sentencias `INSERT`, `FETCH` y `MERGE` de varias filas:
 - matrices de caracteres de longitud variable
 - matrices gráficas de longitud variable
 - Matrices LOB

Además, la directiva `#pragma pack(1)` no puede estar en vigor si planea utilizar estas matrices en sentencias de varias filas.

Matrices de variables de host numéricas

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host numéricas.



Notas:

¹ La dimensión debe ser una constante entera en el rango de 1 a 32767.

Ejemplo

El siguiente ejemplo muestra una declaración de una matriz numérica de variables de host:

```

EXEC SQL BEGIN DECLARE SECTION;
/* declaration of numeric host-variable array */
long serial_num[10];
EXEC SQL END DECLARE SECTION;

```

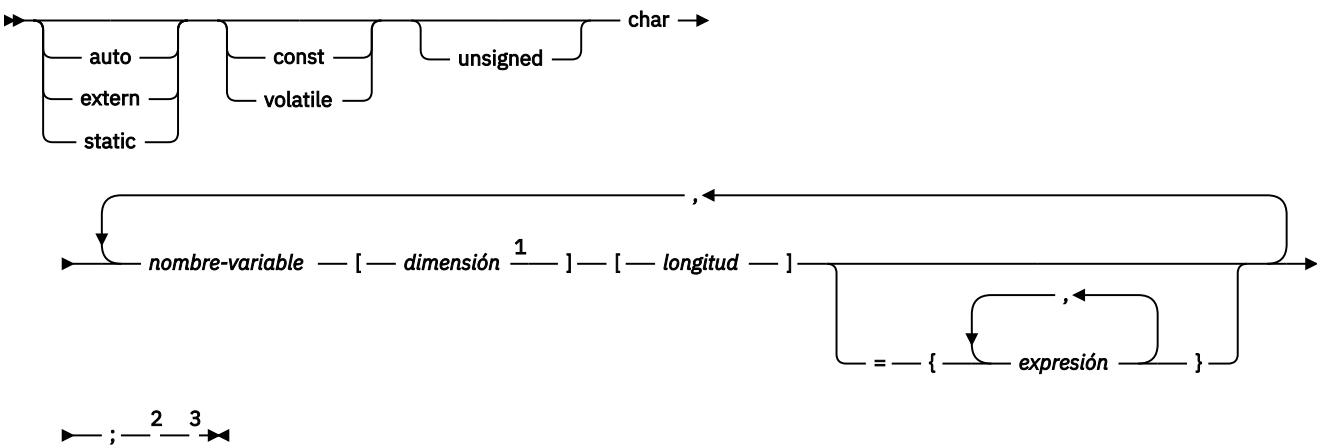
Matrices de variables de host de caracteres

Puede especificar las siguientes formas de matrices de variables de host de caracteres:

- Formulario de caracteres terminados en NUL
- Formulario estructurado VARCHAR
- CLOB

Los siguientes diagramas muestran la sintaxis para formularios distintos de CLOB.

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host de caracteres terminadas en NUL.



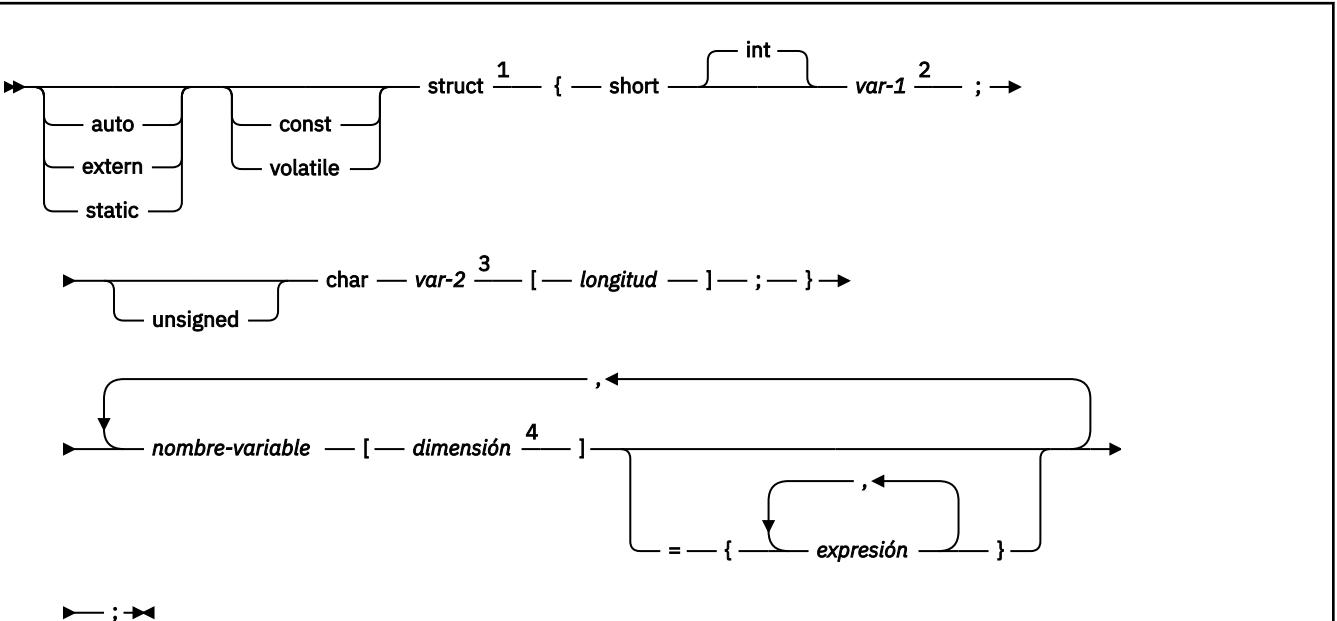
Notas:

¹ la dimensión debe ser una constante entera en el rango de 1 a 32767.

² Cualquier cadena que se asigne a esta variable debe terminar en NUL. Cualquier cadena que se recupere de esta variable terminará en NUL.

³ Las cadenas de un array de variables de host de caracteres terminados en NUL se asignan a cadenas de caracteres de longitud variable (excepto para el NUL).

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host de caracteres de longitud variable que utilizan la forma estructurada VARCHAR.



Notas:

¹ Puede utilizar la etiqueta struct para definir otras variables, pero no puede utilizarlas como matrices de variables de host en SQL.

² var-1 debe ser una variable numérica escalar.

³ var-2 debe ser una variable de matriz escalar CHAR.

⁴ la dimensión debe ser una constante entera en el rango de 1 a 32767.

Ejemplo

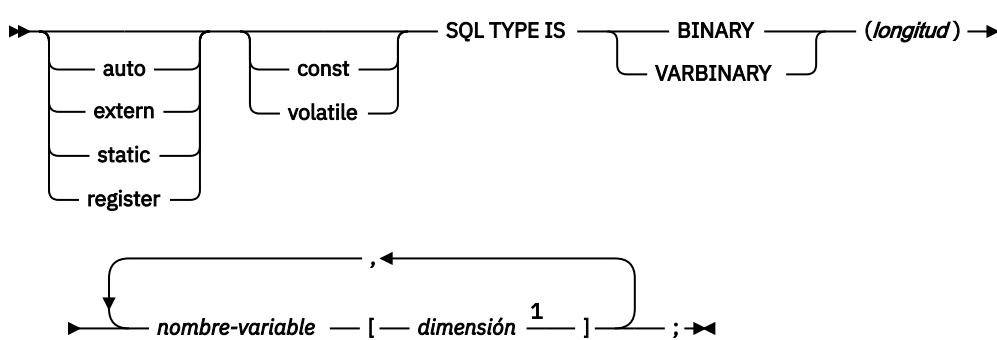
El siguiente ejemplo muestra declaraciones válidas e inválidas de matrices de variables de host VARCHAR.

```
EXEC SQL BEGIN DECLARE SECTION;
/* valid declaration of VARCHAR host-variable array */
struct VARCHAR {
    short len;
    char s[18];
} name[10];

/* invalid declaration of VARCHAR host-variable array */
struct VARCHAR name[10];
```

Matrices binarias de variables de host

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host binarias.



Notas:

¹ la dimensión debe ser una constante entera en el rango de 1 a 32767.

Matrices gráficas de variables de host

Puede especificar las siguientes formas de matrices de variables de host gráficas:

- Forma gráfica terminada en NUL
- Formulario estructurado VARGRAPHIC.

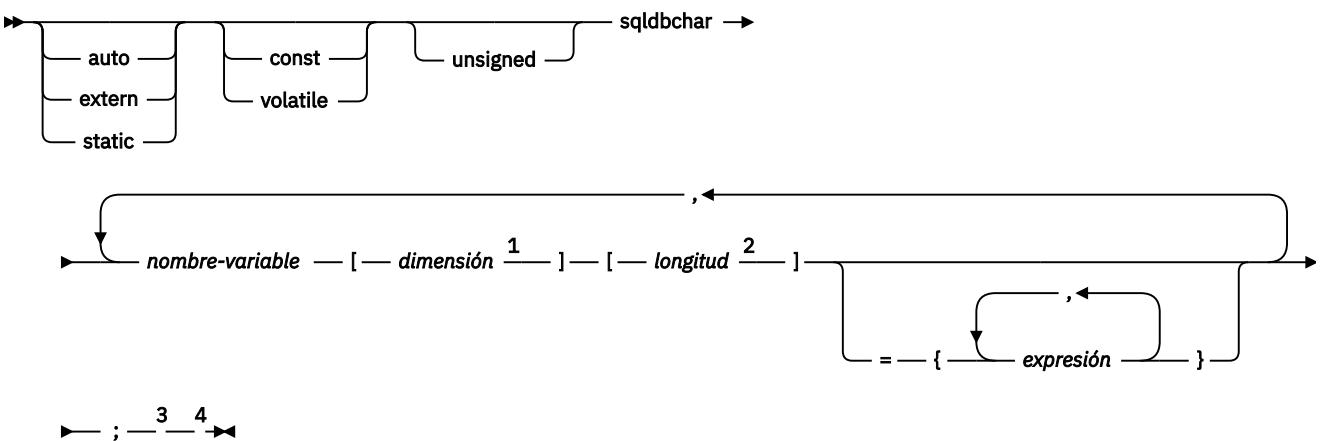
Recomendación: En lugar de utilizar el tipo de datos C wchar_t para definir matrices de variables de host gráficas y vargráficas, utilice una de las siguientes técnicas:

- Defina el tipo de datos sqldbchar utilizando la siguiente declaración typedef:

```
typedef unsigned short sqldbchar;
```

- Utilice el tipo de datos sqldbchar que se define en la instrucción typedef en los archivos de cabecera que proporciona Db2.
- Utiliza el tipo de datos C sin signo corto.

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host gráficas terminadas en NUL.



Notas:

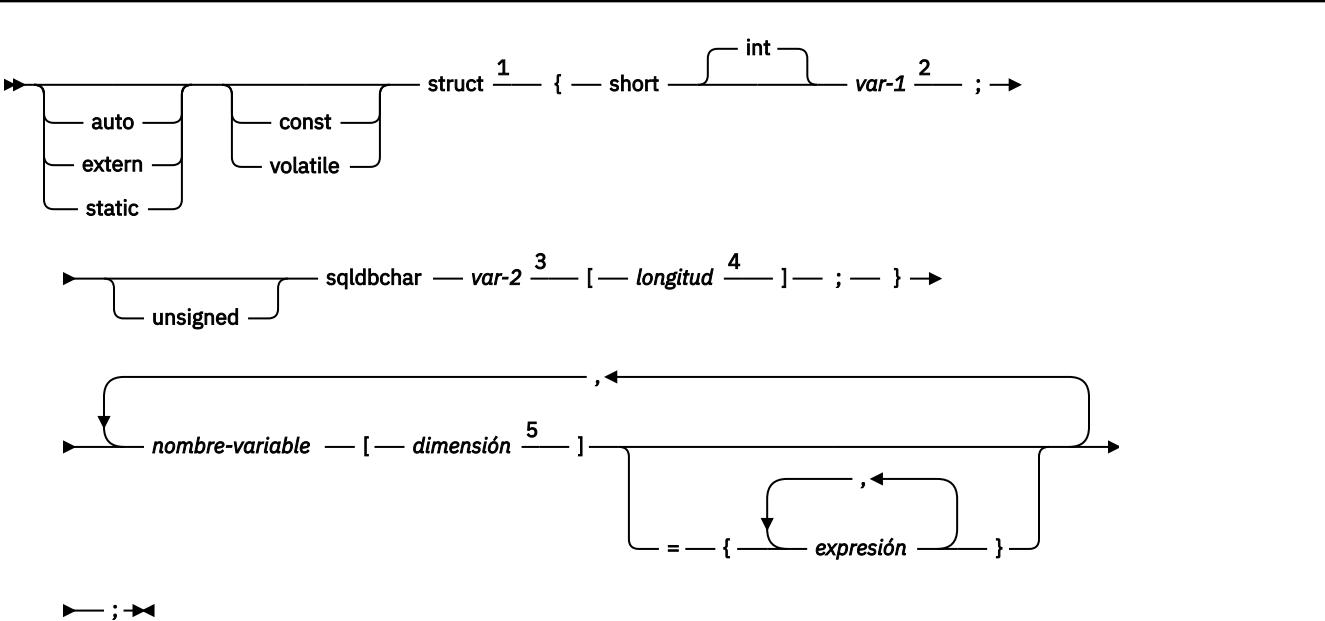
¹ la *dimensión* debe ser una constante entera en el rango de 1 a 32767.

² *longitud* debe ser una constante entera decimal mayor que 1 y no mayor que 16352.

³ Cualquier cadena que se asigne a esta variable debe terminar en NUL. Cualquier cadena que se recupere de esta variable terminará en NUL.

⁴ No asigne caracteres de un solo byte a una matriz de variables de host gráficas terminada en NUL

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host gráficas que utilizan la forma estructurada VARGRAPHIC.



Notas:

¹ Puede utilizar la etiqueta `struct` para definir otras variables, pero no puede utilizarlas como matrices de variables de host en SQL.

² *var-1* debe ser una variable numérica escalar.

³ *var-2* debe ser una variable de matriz de caracteres escalar.

⁴ *longitud* debe ser una constante entera decimal mayor que 1 y no mayor que 16352.

⁵ la *dimensión* debe ser una constante entera en el rango de 1 a 32767.

Ejemplo

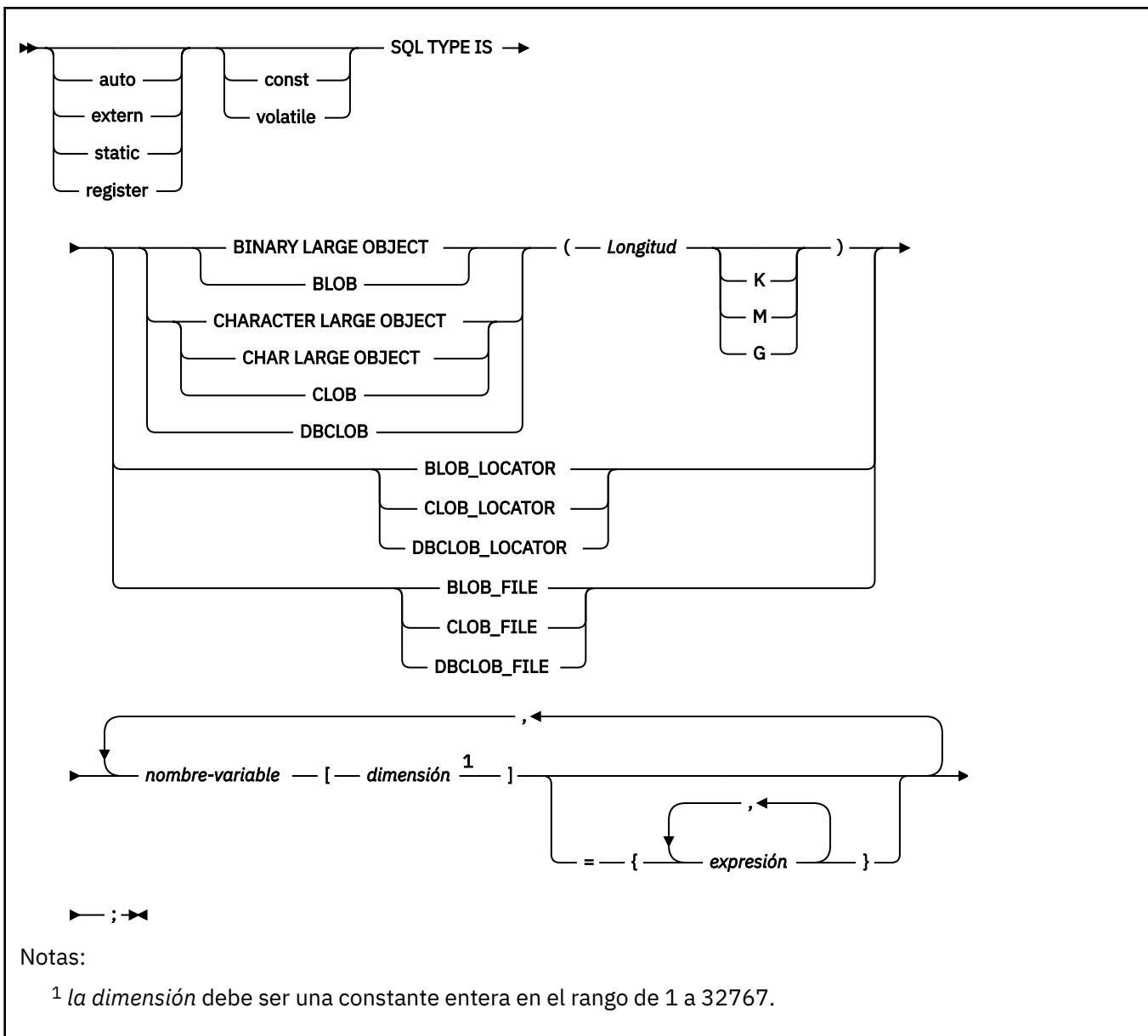
El siguiente ejemplo muestra declaraciones válidas e inválidas de matrices de variables de host gráficas que utilizan la forma estructurada VARGRAPHIC.

```
EXEC SQL BEGIN DECLARE SECTION;
/* valid declaration of host-variable array vgraph */
struct VARGRAPH {
    short len;
    sqldbchar d[10];
} vgraph[20];

/* invalid declaration of host-variable array vgraph */
struct VARGRAPH vgraph[20];
```

LOB, localizador y matrices de variables de referencia de archivos

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host BLOB, CLOB y DBCLOB, localizadores y variables de referencia de archivos.

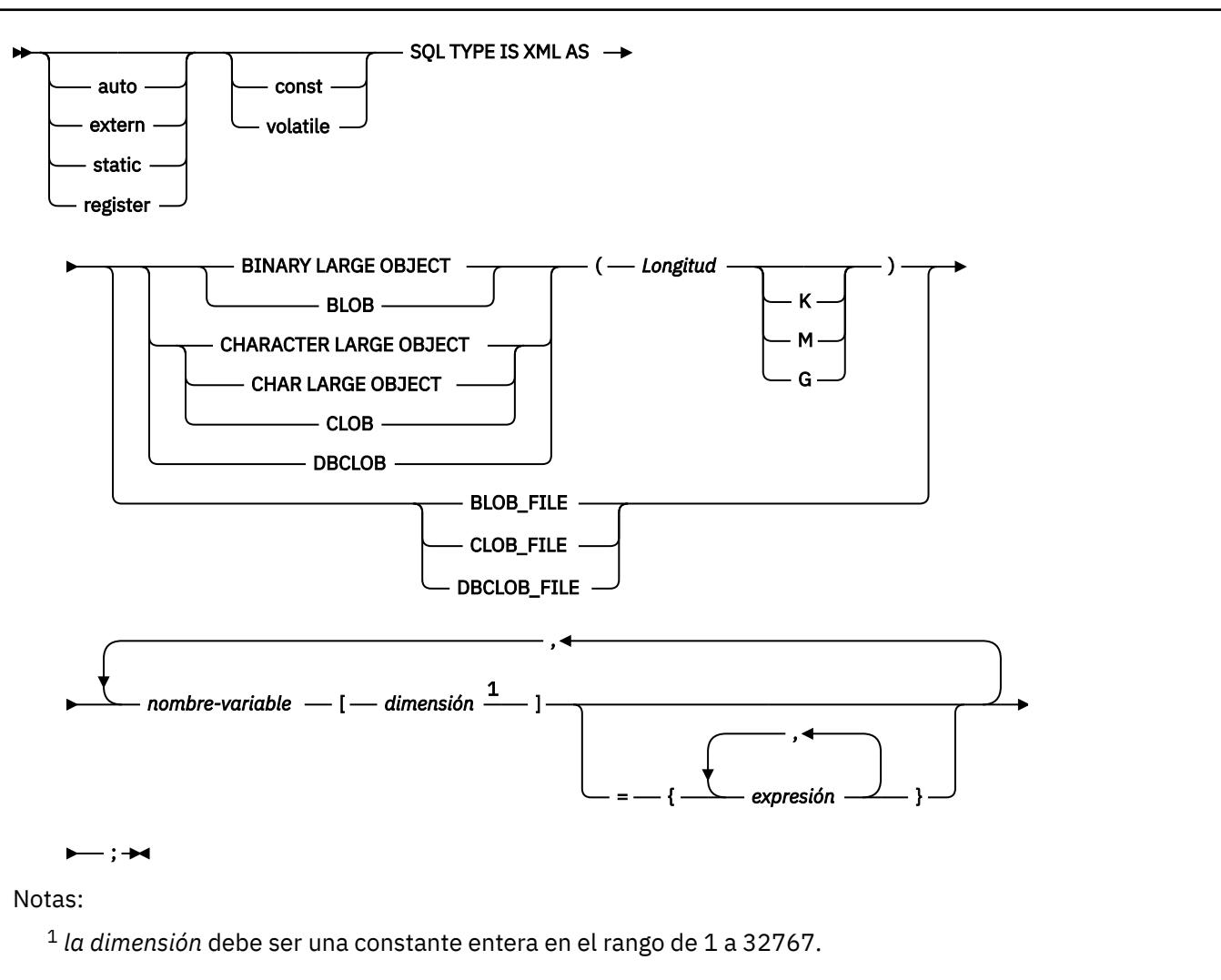


Notas:

¹ la dimensión debe ser una constante entera en el rango de 1 a 32767.

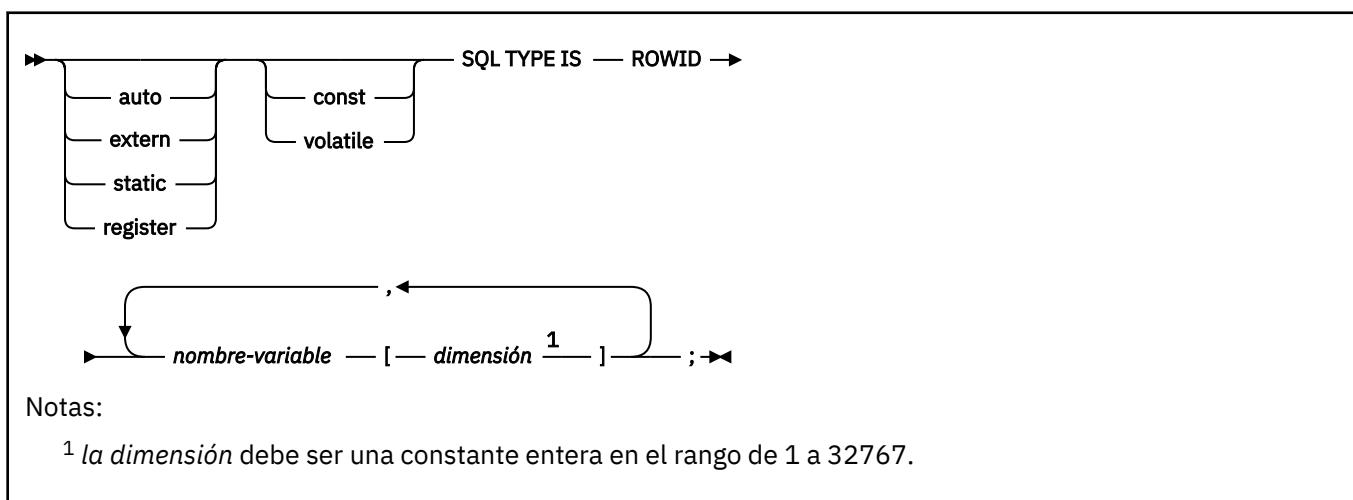
Variables de matriz de referencia de archivos y host XML

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host BLOB, CLOB y DBCLOB y matrices de variables de referencia de archivos para tipos de datos XML.



Matrices de variables ROWID

El siguiente diagrama muestra la sintaxis para declarar matrices de variables ROWID.



Conceptos relacionados

Utilización de matrices de variables de host en sentencias SQL

Utilice matrices de variables de lenguaje de host en sentencias de SQL incorporado para representar valores que el programa no conoce hasta que se ejecuta la consulta. Las matrices de variables de host son útiles para almacenar un conjunto de valores recuperados o para pasar un conjunto de valores que se van a insertar en una tabla.

Matrices de variables de host

Puede utilizar matrices de variables de host para pasar una matriz de datos entre Db2 y su aplicación.

Una matriz de variables de host es una matriz de datos que se declara en el lenguaje de host para ser utilizada dentro de una instrucción SQL.

Matrices de variables de lenguaje principal en PL/I, C, C++ y COBOL (Db2 SQL)

Tareas relacionadas

Inserción de varias filas de datos desde matrices de variables de host

Utilice las variables de host en la sentencia INSERT cuando no conozca al menos algunos de los valores que se han de insertar hasta que el programa se ejecuta.

Almacenamiento de datos LOB en tablas de Db2

Db2 maneja los datos LOB de manera diferente a otros tipos de datos. Como resultado, a veces es necesario realizar acciones adicionales al definir columnas LOB e insertar los datos LOB.

Recuperación de varias filas de datos en matrices de variables de host

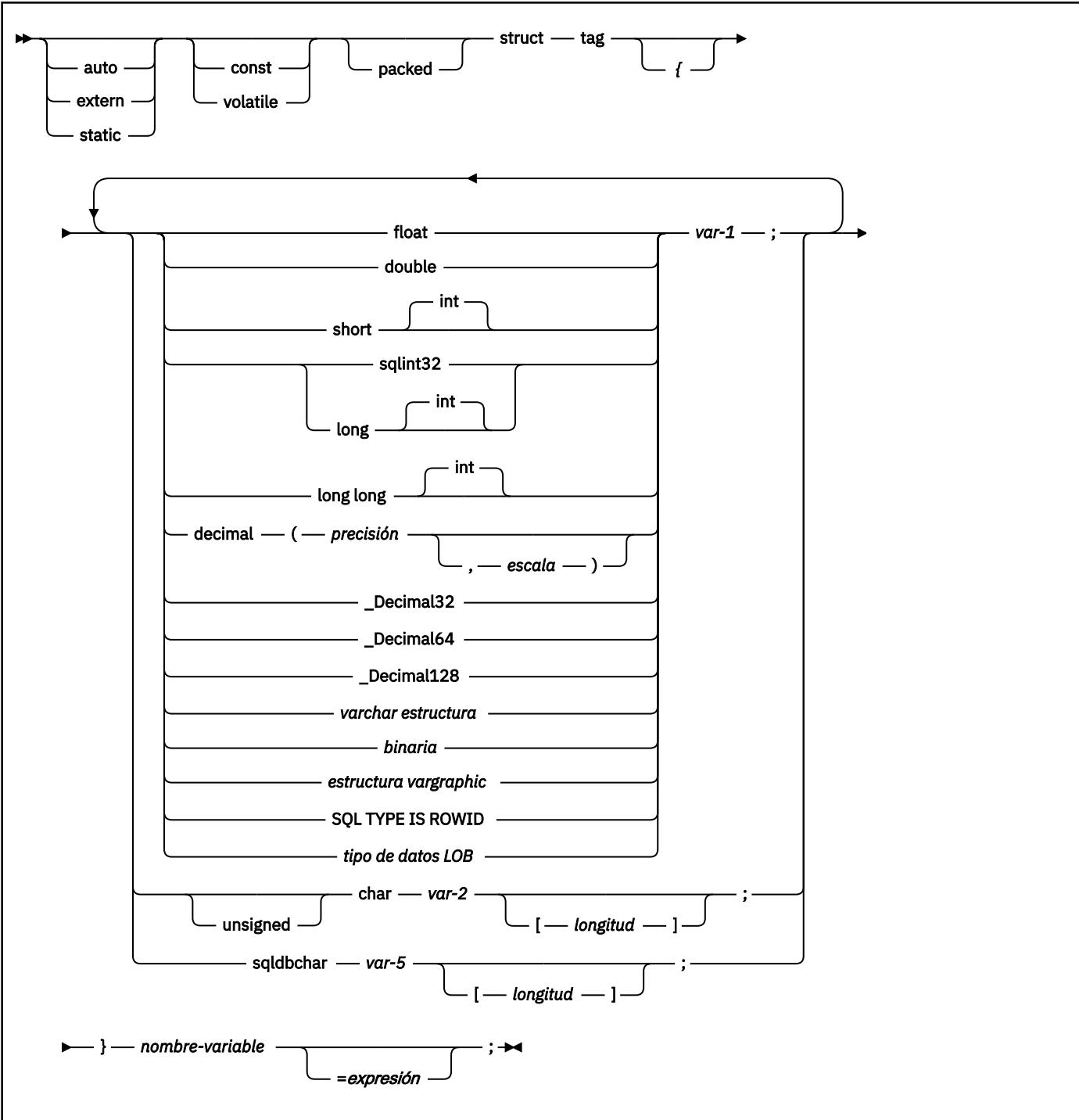
Si sabe que la consulta devuelve varias filas, puede especificar matrices de variables de host para almacenar los valores de columna recuperados.

Estructuras de host en C y C++

Una estructura de host C contiene un grupo ordenado de campos de datos.

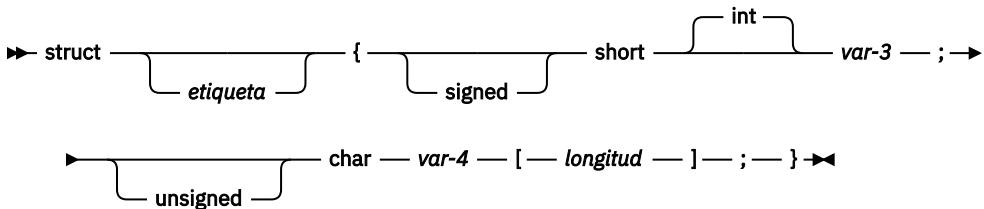
Estructuras host

El siguiente diagrama muestra la sintaxis para declarar estructuras de host.



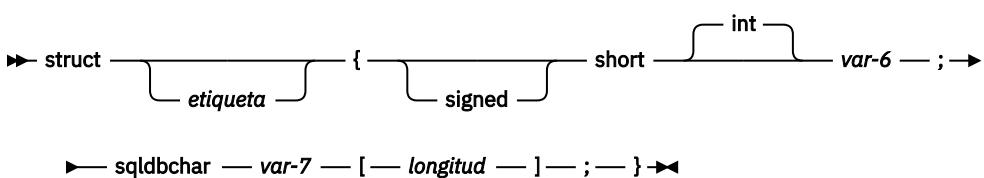
Estructuras VARCHAR

El siguiente diagrama muestra la sintaxis de las estructuras VARCHAR que se utilizan en las declaraciones de estructuras de host.



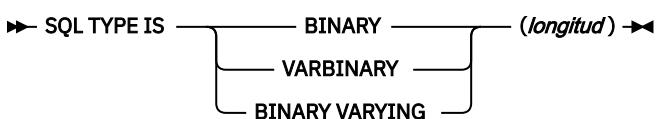
Estructuras VARGRÁFICAS

El siguiente diagrama muestra la sintaxis de las estructuras VARGRAPHIC que se utilizan en las declaraciones de estructuras de host.



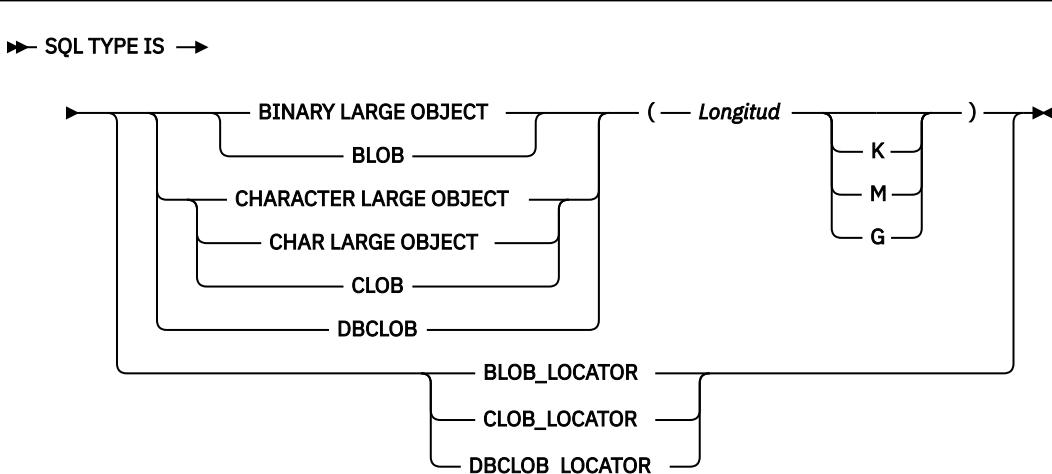
Estructuras binarias

El siguiente diagrama muestra la sintaxis de las estructuras binarias que se utilizan en las declaraciones de estructuras de host.



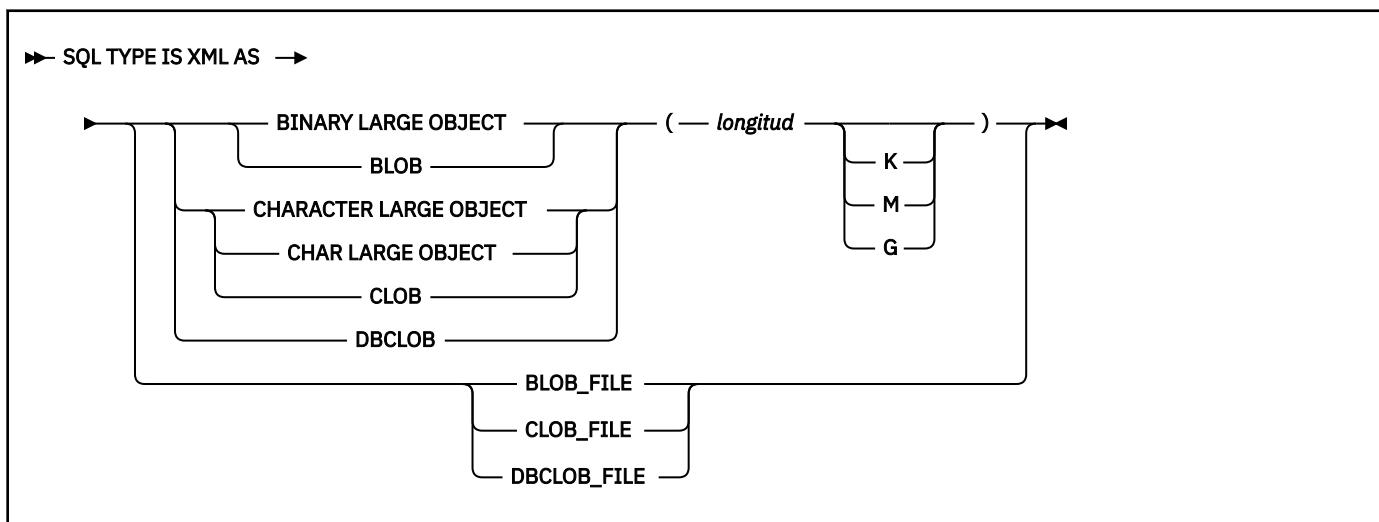
tipos de datos LOB

El siguiente diagrama muestra la sintaxis de los tipos de datos LOB que se utilizan en las declaraciones de estructuras de host.



Tipos de datos LOB para datos XML

El siguiente diagrama muestra la sintaxis de los tipos de datos LOB que se utilizan en las declaraciones de estructuras de host para datos XML.



Ejemplo

En el siguiente ejemplo, la estructura de host se denomina target y contiene los campos c1, c2 y c3. c1 y c3 son matrices de caracteres, y c2 es una variable de host que equivale al tipo de datos SQL VARCHAR. La estructura de host de destino puede formar parte de otra estructura de host, pero debe ser el nivel más profundo de la estructura anidada.

```
struct {char c1[3];
        struct {short len;
                char data[5];
            }c2;
        char c3[2];
    }target;
```

Conceptos relacionados

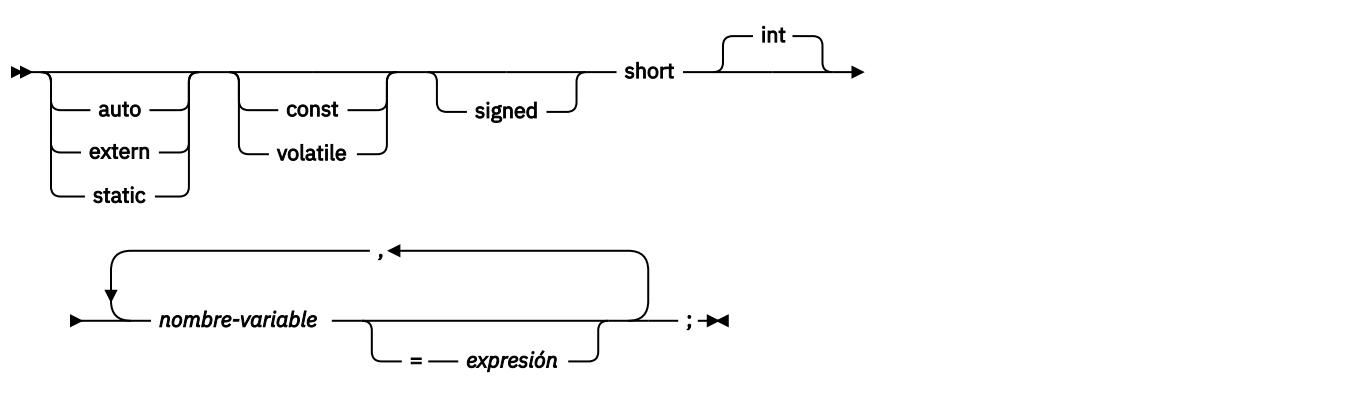
Estructuras host

Utilice las estructuras host para pasar un grupo de variables host entre Db2 y su aplicación.

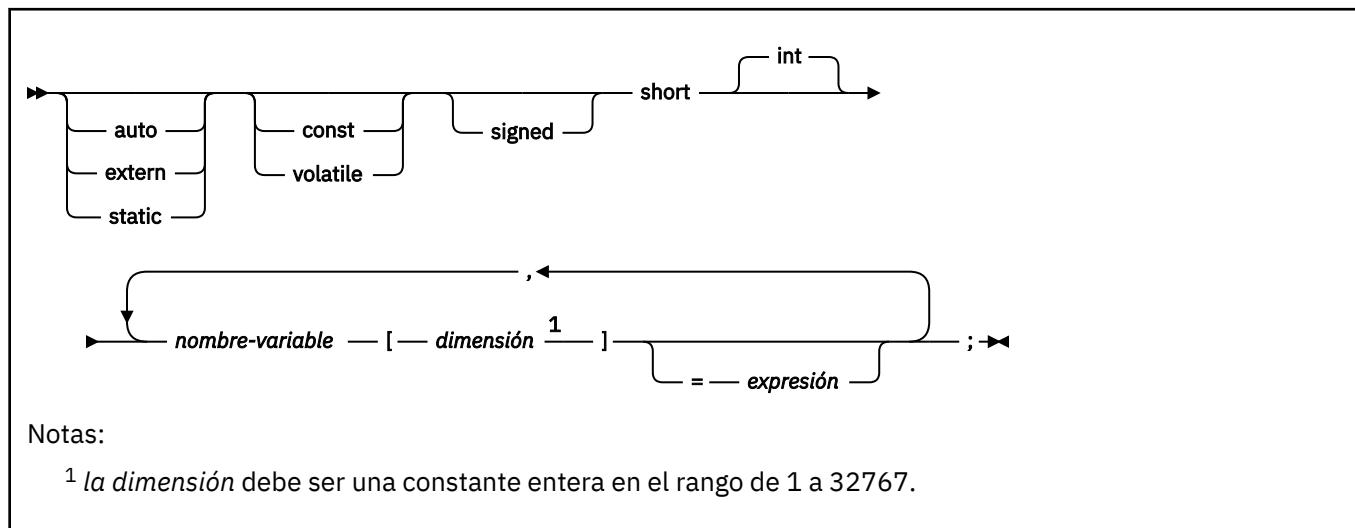
Variables de indicador, matrices de indicador y matrices de indicador de estructura de host en C y C++

Una variable indicadora es un entero de 2 bytes (entero corto). Una matriz de variable indicadora es una matriz de enteros de 2 bytes (entero corto). Declare las variables indicadoras de la misma forma que las variables host. Puede mezclar las declaraciones de dos tipos de variables.

El siguiente diagrama muestra la sintaxis para declarar una variable indicadora en C y C++.



El siguiente diagrama muestra la sintaxis para declarar una matriz de indicadores o una matriz de indicadores de estructura de host en C y C++.



Ejemplo

El siguiente ejemplo muestra una sentencia FETCH con las declaraciones de las variables de host que se necesitan para la sentencia FETCH y sus variables indicadoras asociadas.

```

EXEC SQL FETCH CLS_CURSOR INTO :ClsCd,
                           :Day :DayInd,
                           :Bgn :BgnInd,
                           :End :EndInd;
    
```

Puede declarar estas variables de la siguiente manera:

```

EXEC SQL BEGIN DECLARE SECTION;
char  ClsCd[8];
char  Bgn[9];
char  End[9];
short Day, DayInd, BgnInd, EndInd;
EXEC SQL END DECLARE SECTION;
    
```

Conceptos relacionados

Variables de indicación, matrices y estructuras

Una variable de indicación se asocia con una variable host concreta. Cada variable de indicación contiene un valor entero pequeño que indica alguna información sobre la variable host asociada. Las estructuras y matrices de indicador tienen el mismo propósito para las estructuras y matrices de variables de host.

Tareas relacionadas

[Inserción de valores nulos en columnas utilizando las matrices o variables indicadoras](#)

Si necesita insertar valores nulos en una columna, utilizar una matriz o variable indicadora es una forma fácil de hacerlo. Una variable o matriz de indicador está asociada a una variable de host o matriz de variables de lenguaje de host determinada.

Referencia a variables host de puntero en programas C

Si utiliza el coprocesador de Db2, puede hacer referencia a cualquier variable host de puntero declarada en las sentencias SQL.

Procedimiento

Especifique la variable de host del puntero exactamente como se declaró.

La única excepción es cuando se hace referencia a punteros a matrices de caracteres terminadas en cero. En este caso, no es necesario incluir los paréntesis que formaban parte de la declaración.

ejemplos

Ejemplos: Referencias de variables de host de puntero escalar

Tabla 99. Ejemplo de referencias a variables de host de puntero escalar

Declaración	Descripción	Referencia
short *hvshortp;	hvshortp es una variable de host puntero que apunta a dos bytes de almacenamiento.	EXEC SQL set :hvshortp=123;
double *hvdoubp;	hvdoubp es una variable de host puntero que apunta a ocho bytes de almacenamiento.	EXEC SQL set :hvdoubp=456;
char (*hvcharpn) [20];	hvcharpn es una variable de host puntero que apunta a una matriz de caracteres terminada en cero de hasta 20 bytes.	EXEC SQL set :hvcharpn='nul_terminated';

Ejemplo: Referencia de variable de host de puntero de carácter delimitado

Supongamos que su programa declara la siguiente variable de host de puntero de carácter acotado:

```
struct {  
    unsigned long len;  
    char * data;  
} hvbcharp;
```

El siguiente ejemplo hace referencia a esta variable de host de puntero de carácter acotado:

```
hvcharp.len = dynlen; a  
hvcharp.data = (char *) malloc (hvcharp.len); b  
EXEC SQL set :hvcharp = 'data buffer with length'; c
```

Nota:

a

dynlen puede ser una constante de tiempo de compilación o una variable con un valor que se asigna en tiempo de ejecución.

b

El almacenamiento se asigna dinámicamente para hvcharp.data.

c

La instrucción SQL hace referencia al nombre de la estructura, no a un elemento dentro de la estructura.

Ejemplos: Referencias de variables de host de puntero de matriz

Tabla 100. Ejemplo de referencias a variables de host de puntero de matriz

Declaración	Descripción	Referencia
short * hvarrp1[6]	hvarrp1 es una matriz de 6 punteros que apuntan a dos bytes de almacenamiento cada uno.	EXEC SQL set :hvarrp1[n]=123;
double * hvarrp2[3]	hvarrp2 es una matriz de 3 punteros que apuntan a 8 bytes de almacenamiento cada uno.	EXEC SQL set :hvarrp2[n]=456;
struct { unsigned long len; char * data; } hvbarrp3[5];	hvbarrp3 es una matriz de 5 punteros de caracteres delimitados.	EXEC SQL set :hvbarrp3[n] = 'data buffer with length'

Ejemplo: Referencia de variable de host de matriz de estructura

Supongamos que su programa declara el siguiente puntero a la estructura `tbl_struct`:

```
struct tbl_struct *ptr_tbl_struct =  
(struct tbl_struct *) malloc (sizeof (struct tbl_struct) * n);
```

Para hacer referencia a estos datos en sentencias SQL, utilice el puntero como se muestra en el siguiente ejemplo. Asume que `tbl_sel_cur` es un cursor declarado.

```
for (L_col_cnt = 0; L_col_cnt < n; L_con_cnt++)  
{ ...  
    EXEC SQL FETCH tbl_sel_cur INTO :ptr_tbl_struct [L_col_cnt]  
    ...  
}
```

Tareas relacionadas

Declaración de variables host de puntero en programas C

Si utiliza el coprocesador de Db2, puede utilizar variables host de puntero con almacenamiento asignado estáticamente o dinámicamente. Estas variables host de puntero pueden apuntar a datos numéricos, datos no numéricos o a una estructura.

Declaración de variables host de puntero en programas C

Si utiliza el coprocesador de Db2, puede utilizar variables host de puntero con almacenamiento asignado estáticamente o dinámicamente. Estas variables host de puntero pueden apuntar a datos numéricos, datos no numéricos o a una estructura.

Acerca de esta tarea

Puede declarar los siguientes tipos de variables de host puntero:

Variable de host del puntero escalar

Una variable anfitriona que apunta a datos escalares numéricos o no numéricos.

Variable de host del puntero de matriz

Una variable anfitriona que es una matriz de punteros.

Estructura matriz variable de host

Una variable anfitriona que apunta a una estructura.

Procedimiento

Incluya un asterisco (*) en cada declaración de variable para indicar que la variable es un puntero.

Restricciones:

- No puede utilizar variables de host puntero que apunten a datos de caracteres de longitud desconocida. Por ejemplo, no especifique la siguiente declaración: `char * hvcharpu`. En su lugar, especifique la longitud de los datos utilizando una variable de host de puntero de carácter acotado. *Una variable de host de puntero de carácter delimitado* es una variable de host que se declara como una estructura con los siguientes elementos:
 - Un campo de 4 bytes que contiene la longitud del área de almacenamiento.
 - Un puntero al área de almacenamiento dinámico no numérico.
- No puede utilizar punteros sin tipificar. Por ejemplo, no especifique la siguiente declaración: `void * untypedprt`.

ejemplos

Ejemplo: Declaraciones de variables de host de puntero escalar

Tabla 101. Ejemplo de declaraciones de variables de host de puntero escalar

Declaración	Descripción
<code>short *hvshortp;</code>	hvshortp es una variable de host puntero que apunta a dos bytes de almacenamiento.
<code>double *hvdoubp;</code>	hvdoubp es una variable de host puntero que apunta a ocho bytes de almacenamiento.
<code>char (*hvcharpn) [20];</code>	hvcharpn es una variable de host puntero que apunta a una matriz de caracteres terminada en cero de hasta 20 bytes.

Ejemplo: Declaración de variable de host de puntero de carácter delimitado

El siguiente código de ejemplo declara una variable de host de puntero de carácter acotado llamada hvbcharp con dos elementos: len y data.

```
struct {
  unsigned long len;
  char * data;
} hvbcharp;
```

Ejemplo: declaraciones de variables de host de puntero de matriz

Tabla 102. Ejemplo de declaraciones de variables de host de puntero de matriz

Declaración	Descripción
<code>short * hvarrp1[6]</code>	hvarrp1 es una matriz de 6 punteros que apuntan a dos bytes de almacenamiento cada uno.
<code>double * hvarrp2[3]</code>	hvarrp2 es una matriz de 3 punteros que apuntan a 8 bytes de almacenamiento cada uno.
<code>struct { unsigned long len; char * data; } hvarrp3[5];</code>	hvarrp3 es una matriz de 5 punteros de caracteres delimitados.

Ejemplo: Declaración de variable de host de matriz de estructura

El siguiente código de ejemplo declara una estructura de tabla llamada `tbl_struct`.

```
struct tbl_struct
{
  char colname[20];
  small int colno;
  small int coltype;
```

```

    small int collen;
};


```

El siguiente código de ejemplo declara un puntero a la estructura `tbl_struct`. El almacenamiento se asigna dinámicamente para un máximo de `n` filas.

```

struct tbl_struct *ptr_tbl_struct =
  (struct tbl_struct *) malloc (sizeof (struct tbl_struct) * n);


```

Tareas relacionadas

[Referencia a variables host de puntero en programas C](#)

Si utiliza el coprocesador de Db2, puede hacer referencia a cualquier variable host de puntero declarada en las sentencias SQL.

SQL equivalente y tipos de datos C

Cuando declara variables host en los programas C, el precompilador utiliza tipos de datos SQL equivalentes. Cuando recupere datos de un tipo de datos SQL concreto en una variable host, tendrá que asegurarse de que la variable host sea de un tipo de datos equivalente.

La siguiente tabla describe el tipo de datos SQL y los valores base `SQLTYPE` y `SQLLEN` que el precompilador utiliza para las variables de host en las sentencias SQL.

Tabla 103. Tipos de datos SQL, valores `SQLLEN` y valores `SQLTYPE` que el precompilador utiliza para variables de host en programas C

Tipo de datos de la variable de host C	SQLTYPE de la variable de host ¹	SQLLEN de la variable de host	Tipos de datos SQL
<code>short int</code>	500	2	<code>SMALLINT</code>
<code>long int</code>	496	4	<code>ENTERO</code>
<code>long long</code> <code>long long int</code> <code>sqlint64</code>	492	8	<code>BIGINT</code> ⁵
<code>dec imal(p,s)2</code>	484	p en el byte 1, s en el byte 2	<code>DECIMAL(p,s)2</code>
<code>_Decimal32</code>	996/997	4	<code>DECFL</code> OAT(16) ⁷ , 8
<code>_Decimal64</code>	996/997	8	<code>DECFLOAT(16)</code> 8
<code>_Decimal128</code>	996/997	16	<code>DECFL</code> O AT(34)8
<code>flotante</code>	480	4	<code>FLOAT</code> (precisión simple)
<code>doble</code>	480	8	<code>FLOAT</code> (doble precisión)
<code>TIPO SQL ES BINARIO(n), 1<=n <=255</code>	912	<i>n</i>	<code>BINARY(n)</code>
<code>TIPO SQL ES VARBINARIO(n), 1<=n <=32704</code>	908	<i>n</i>	<code>VARBINARY(n)</code>

Tabla 103. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas C (continuación)

Tipo de datos de la variable de host C	SQLTYPE de la variable de host ¹	SQLLEN de la variable de host	Tipos de datos SQL
Formulario de un solo carácter	452	1	CHAR(1)
Formulario de caracteres terminados en NUL	460	<i>n</i>	VARCHAR (<i>n-1</i>)
Estructurado VARCHAR formulario $1 \leq n \leq 2\,55$	448	<i>n</i>	VARCHAR(<i>n</i>)
Formulario estructurado VARCHAR $n > 255$	456	<i>n</i>	VARCHAR(<i>n</i>)
Formulario de gráfico único	468	1	GRAPHIC (1)
NUL-terminated forma gráfica	400	<i>n</i>	VARGRÁFICO (<i>n-1</i>)
VARGRAPHIC forma estructurada $1 \leq n < 128$	464	<i>n</i>	VARGRAPHIC (<i>n</i>)
VARGRAPHIC forma estructurada $n > 127$	472	<i>n</i>	VARGRAPHIC (<i>n</i>)
• TIPO_SQL_ES RESULT_SET _LOCALIZADOR	972	4	Localizador de conjunto de resultados3
TIPO_SQL_ES Tabla como <i>nombre-tabla</i> AS LOCATOR	976	4	Localizador de mesas3
TIPO_SQL_ES BLOB_LOCATOR	960	4	Localizador BL OB3
TIPO_SQL_ES CLOB_LOCATOR	964	4	Localizador CLOB3
TIPO_SQL_ES DBCLOB_LOCATOR	968	4	Localizador DBCLOB3
TIPO_SQL_ES BLOB(<i>n</i>) $1 \leq n \leq 214748364$	404	<i>n</i>	BLOB (<i>n</i>)

Tabla 103. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas C (continuación)

Tipo de datos de la variable de host C	SQLTYPE de la variable de host ¹	SQLLEN de la variable de host	Tipos de datos SQL
TIPO SQL ES <i>CLOB(n) 1≤n≤2147483647</i>	408	<i>n</i>	CLOB (<i>n</i>)
TIPO SQL ES <i>DBCLOB(n) 1≤n≤1073741823</i>	412	<i>n</i>	<i>DBCLOB(n)</i> ⁴
SQL TYPE IS XML AS <i>BLOB(n)</i>	404	0	XML
SQL TYPE IS XML AS <i>CLOB(n)</i>	408	0	XML
SQL TYPE IS XML AS <i>DBCLOB(n)</i>	412	0	XML
TIPO SQL ES <i>BLOB_FILE</i>	916/917	267	Referencia de archivo BLOB ³
TIPO SQL ES <i>CLOB_FILE</i>	920/921	267	Referencia de archivo CLOB ³
EL TIPO SQL ES <i>DBCLOB_FILE</i>	924/925	267	Referencia de archivo DBCLOB ³
SQL TYPE IS XML AS <i>BLOB_FILE</i>	916/917	267	Referencia de archivo XML BLOB ³
SQL TYPE IS XML AS <i>CLOB_FILE</i>	920/921	267	Referencia de archivo XML CLOB ³
SQL TYPE IS XML AS <i>DBCLOB_FILE</i>	924/925	267	Referencia de archivo XML DBCLOB ³
EL TIPO SQL ES <i>ROWID</i>	904	40	ROWID

Tabla 103. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas C (continuación)

Tipo de datos de la variable de host C	SQLTYPE de la variable de host1	SQLLEN de la variable de host	Tipos de datos SQL
Notas:			
1. Si una variable de host incluye una variable de indicador, el valor SQLTYPE es el valor SQLTYPE base más 1.			
2. <i>p</i> es <i>la precisión</i> ; en terminología SQL, este es el número total de dígitos. En C, esto se llama <i>tamaño</i> .			
<i>s</i> es <i>la escala</i> ; en terminología SQL, es el número de dígitos a la derecha del punto decimal. En C, esto se llama <i>precisión</i> .			
C++ no admite el tipo de datos decimal.			
3. No utilice este tipo de datos como tipo de columna.			
4. <i>n</i> es el número de caracteres de doble byte.			
5. No existe un equivalente exacto. Utilice DECIMAL(19,0).			
6. El tipo de datos C long se asigna al tipo de datos SQL BIGINT.			
7. La variable de host DFP con una longitud de 4 es compatible, mientras que la columna DFP solo se puede definir con una longitud de 8 (DECFLOAT(16)) o 16 (DECFLOAT(34)).			
8. Para utilizar el tipo de datos de host de coma flotante decimal, debe hacer lo siguiente:			
<ul style="list-style-type: none"> • Utilice z/OS 1.10 o posterior (z/OS V1R10 XL C/C++). • Compilar con la opción de compilador C/C++, DFP. • Especifique la opción del compilador SQL para habilitar el coprocesador de optimización de consultas (Db2). • Especifique la opción del compilador C/C++, ARCH(7). Es requerido por la opción del compilador DFP si el tipo DFP se utiliza en el código fuente. • Especifique la opción del compilador 'DEFINE(__STDC_WANT_DEC_FP__)' porque DFP no forma parte oficialmente del estándar de lenguaje C/C++. 			

La siguiente tabla muestra las variables de host C equivalentes para cada tipo de datos SQL. Utilice esta tabla para determinar el tipo de datos C para las variables de host que defina para recibir la salida de la base de datos. Por ejemplo, si recupera datos TIMESTAMP, puede definir una variable de forma de carácter terminado en NUL o de forma estructurada VARCHAR

Esta tabla muestra las conversiones directas entre los tipos de datos SQL y los tipos de datos C. Sin embargo, varios tipos de datos SQL son compatibles. Cuando realiza asignaciones o comparaciones de datos que tienen tipos de datos compatibles, Db2 convierte esos tipos de datos compatibles.

Tabla 104. Equivalentes de variables de host C que puede utilizar al recuperar datos de un tipo de datos SQL concreto

Tipos de datos SQL	Variable de host C equivalente	Notas
SMALLINT	short int	
ENTERO	long int	
DECIMAL(<i>p,s</i>) o NUMERIC(<i>p,s</i>)	decimal	Puede utilizar el tipo de datos double si su compilador C no tiene un tipo de datos decimal; sin embargo, double no es un equivalente exacto.
REAL o FLOAT(<i>n</i>)	flotante	1<=n <=21

Tabla 104. Equivalentes de variables de host C que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host C equivalente	Notas
DOBLE PRECISIÓN o <i>FLOTANTE(n)</i>	doble	$2 \leq n \leq 53$
DECFLOAT(16)	_Decminal32	
DECFLOAT(34)	_Decimal128	
BIGINT	long long, long long int, y sqlint64	
BINARY(<i>n</i>)	TIPO SQL ES BINARIO(<i>n</i>)	$1 \leq n \leq 255$ Si los datos pueden contener caracteres NUL (\0), es posible que ciertas funciones de las bibliotecas C y C++ no manejen los datos correctamente. Asegúrese de que su aplicación maneja los datos correctamente.
VARBINARY(<i>n</i>)	TIPO SQL ES VARBINARIO(<i>n</i>)	$1 \leq n \leq 32704$
CHAR(1)	forma de un solo carácter	
CHAR (<i>n</i>)	sin equivalente exacto	Si $n > 1$, utilice la forma de carácter terminada en NUL
VARCHAR(<i>n</i>)	Formulario de caracteres terminados en NUL	Si los datos pueden contener caracteres NUL (\0), utilice el formulario estructurado VARCHAR. Deje al menos $n+1$ para acomodar el terminador NUL.
	Formulario estructurado VARCHAR	
GRAPHIC (1)	forma de gráfico único	
GRAPHIC (<i>n</i>)	sin equivalente exacto	Si $n > 1$, utilice el formato gráfico terminado en NUL. <i>n</i> es el número de caracteres de doble byte.
VARGRAPHIC (<i>n</i>)	Forma gráfica terminada en NUL	Si los datos pueden contener valores NUL gráficos (\0\0), utilice el formulario estructurado VARGRAPHIC. Deje al menos $n+1$ para acomodar el terminador NUL. <i>n</i> es el número de caracteres de doble byte.
	Formulario estructurado VARGRAPHIC	<i>n</i> es el número de caracteres de doble byte.
FECHA	Formulario de caracteres terminados en NUL	Si está utilizando una rutina de salida de fecha, esa rutina determina la longitud. De lo contrario, deje al menos 11 caracteres para acomodar el terminador NUL.
	Formulario estructurado VARCHAR	Si está utilizando una rutina de salida de fecha, esa rutina determina la longitud. De lo contrario, deje al menos 10 caracteres.

Tabla 104. Equivalentes de variables de host C que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host C equivalente	Notas
HORA	Formulario de caracteres terminados en NUL	Si está utilizando una rutina de salida de tiempo, la duración la determina dicha rutina. De lo contrario, la longitud debe ser de al menos 7; para incluir segundos, la longitud debe ser de al menos 9 para acomodar el terminador NUL.
	Formulario estructurado VARCHAR	Si está utilizando una rutina de salida de tiempo, la duración la determina dicha rutina. De lo contrario, la duración debe ser de al menos 6; para incluir segundos, la duración debe ser de al menos 8.
TIMESTAMP	Formulario de caracteres terminados en NUL	La longitud debe ser de al menos 20. Para incluir microsegundos, la longitud debe ser 27. Si la duración es inferior a 27, se produce un truncamiento en la parte de los microsegundos.
	Formulario estructurado VARCHAR	La longitud debe ser de al menos 19. Para incluir microsegundos, la longitud debe ser 26. Si la duración es inferior a 26, se produce un truncamiento en la parte de los microsegundos.
TIMESTAMP(0)	Formulario de caracteres terminados en NUL	La longitud debe ser de al menos 20.
	Formulario estructurado VARCHAR	La longitud debe ser de al menos 19.
MARCA DE TIEMPO(p) $p > 0$	Formulario de caracteres terminados en NUL	La longitud debe ser de al menos 20. Para incluir fracciones de segundo, la duración debe ser $21+x$, donde x es el número de fracciones de segundo que se van a incluir; si x es menor que p , se produce un truncamiento en la parte de las fracciones de segundo.
	Formulario estructurado VARCHAR	La longitud debe ser de al menos 19. Para incluir fracciones de segundo, la longitud debe ser $20+x$, donde x es el número de fracciones de segundo que se van a incluir; si x es menor que p , se produce un truncamiento en la parte de las fracciones de segundo.
TIMESTAMP (0) WITH TIME ZONE	Formulario de caracteres terminados en NUL	La longitud debe ser de al menos 26.
	Formulario estructurado VARCHAR	La longitud debe ser de al menos 25.
FECHA Y HORA (p) CON ZONA HORARIA	Formulario de caracteres terminados en NUL	La longitud debe ser de al menos $27+p$.
	Formulario estructurado VARCHAR	La longitud debe ser de al menos $26+p$.

Tabla 104. Equivalentes de variables de host C que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host C equivalente	Notas
Localizador de conjunto de resultados	TIPO SQL ES RESULT_SET_LOCATOR	Utilice este tipo de datos solo para recibir conjuntos de resultados. No utilice este tipo de datos como tipo de columna.
Localizador de tablas	SQL TYPE IS TABLE LIKE <i>nombre-tabla</i> AS LOCATOR	Utilice este tipo de datos solo en una función definida por el usuario o en un procedimiento almacenado para recibir filas de una tabla de transición. No utilice este tipo de datos como tipo de columna.
localizador de BLOB	TIPO SQL ES BLOB_LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas BLOB. No utilice este tipo de datos como tipo de columna.
localizador de CLOB	TIPO SQL ES CLOB_LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas CLOB. No utilice este tipo de datos como tipo de columna.
localizador de DBCLOB	TIPO SQL ES DBCLOB_LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas DBCLOB. No utilice este tipo de datos como tipo de columna.
BLOB (<i>n</i>)	TIPO SQL ES BLOB(<i>n</i>)	$1 \leq n \leq 214748364\ 7$
CLOB (<i>n</i>)	TIPO SQL ES CLOB(<i>n</i>)	$1 \leq n \leq 214748364\ 7$
DBCLOB (<i>n</i>)	TIPO SQL ES DBCLOB(<i>n</i>)	<i>n</i> es el número de caracteres de doble byte. $1 \leq n \leq 10737418\ 23$
XML	SQL TYPE IS XML AS BLOB(<i>n</i>)	$1 \leq n \leq 214748364\ 7$
XML	SQL TYPE IS XML AS CLOB(<i>n</i>)	$1 \leq n \leq 214748364\ 7$
XML	SQL TYPE IS XML AS DBCLOB(<i>n</i>)	<i>n</i> es el número de caracteres de doble byte. $1 \leq n \leq 10737418\ 23$
Referencia de archivo BLOB	TIPO SQL ES BLOB_FILE	Utilice este tipo de datos solo para manipular datos en columnas BLOB. No utilice este tipo de datos como tipo de columna.
Referencia de archivo CLOB	TIPO SQL ES CLOB_FILE	Utilice este tipo de datos solo para manipular datos en columnas CLOB. No utilice este tipo de datos como tipo de columna.
Referencia de archivo DBCLOB	EL TIPO SQL ES DBCLOB_FILE	Utilice este tipo de datos solo para manipular datos en columnas DBCLOB. No utilice este tipo de datos como tipo de columna.

Tabla 104. Equivalentes de variables de host C que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host C equivalente	Notas
Referencia de archivo XML BLOB	SQL TYPE IS XML AS BLOB_FILE	Utilice este tipo de datos solo para manipular datos XML como archivos BLOB. No utilice este tipo de datos como tipo de columna.
Referencia de archivo XML CLOB	SQL TYPE IS XML AS CLOB_FILE	Utilice este tipo de datos solo para manipular datos XML como archivos CLOB. No utilice este tipo de datos como tipo de columna.
Referencia de archivo XML DBCLOB	SQL TYPE IS XML AS DBCLOB_FILE	Utilice este tipo de datos solo para manipular datos XML como archivos DBCLOB. No utilice este tipo de datos como tipo de columna.
ROWID	EL TIPO SQL ES ROWID	

La siguiente tabla muestra las definiciones del lenguaje C que se deben utilizar en los procedimientos almacenados en C y en las funciones definidas por el usuario, cuando los tipos de datos de los parámetros en las definiciones de rutina son LOB, ROWID o localizadores. Para otros tipos de datos de parámetros, las definiciones del lenguaje C son las mismas que las de la anterior sección " [Tabla 104 en la página 647](#) ".

Tabla 105. Declaraciones equivalentes en lenguaje C para LOB, ROWID y localizadores en definiciones de rutinas definidas por el usuario

Tipo de datos SQL en la definición 1	Declaración C
localizador de tablas	unsigned long
localizador de BLOB	
localizador de CLOB	
localizador de DBCLOB	
BL OB (n)	<pre>struct {unsigned long length; char data[n]; } var;</pre>
CLOB(n)	<pre>struct {unsigned long length; char var_data[n]; } var;</pre>
DBCL OB (n)	<pre>struct {unsigned long length; sqldbchar data[n]; } var;</pre>
ROWID	<pre>struct { short int length; char data[40]; } var;</pre>

Tabla 105. Declaraciones equivalentes en lenguaje C para LOB, ROWID y localizadores en definiciones de rutinas definidas por el usuario (continuación)

Tipo de datos SQL en la definición 1	Declaración C
VARCHAR(<i>n</i>) ²	<p>Si se especifica o implica PARAMETER VARCHAR NULTERM:</p> <pre>char data[n+1];</pre> <p>Si se especifica PARAMETER VARCHAR STRUCTURE:</p> <pre>struct {short len; char data[n]; } var;</pre>

Nota:

1. El archivo SQLUDF, que se encuentra en el conjunto de datos DSN1210. SDSNC.H, incluye el typedef sqldbchar. El uso de sqldbchar le permite manipular datos DBCS y Unicode UTF-16 en el mismo formato en el que se almacenan en Db2. sqldbchar también facilita la migración de aplicaciones a otras plataformas de e Db2 .
2. Esta fila no se aplica a VARCHAR(*n*) PARA DATOS DE BIT. Los DATOS BIT siempre se transmiten en una representación estructurada.

Conceptos relacionados

[Compatibilidad de SQL y tipos de datos de lenguaje](#)

Los tipos de datos de variable host que se utilizan en sentencias SQL deben ser compatibles con los tipos de datos de las columnas con las que tiene intención de utilizarlos.

[Declaraciones de variable de referencia de archivo LOB, variable host LOB y localizador LOB](#)

Cuando escribe aplicaciones para manipular datos LOB, necesita declarar variables hosto para contener los datos LOB o el localizador LOB. De lo contrario, debe declarar variables de referencia de archivo LOB para apuntar a datos LOB.

[Tipos de datos de variable host para datos XML en aplicaciones de SQL incorporado \(Db2 Programming for XML\)](#)

Gestión de códigos de error de SQL en aplicaciones C y C++

Las aplicaciones C y C++ pueden solicitar más información sobre los códigos de error de SQL utilizando la subrutina DSNTIAR o emitiendo una sentencia GET DIAGNOSTICS.

Procedimiento

Para solicitar información sobre errores SQL en aplicaciones C y C++, utilice los siguientes métodos:

- Puede utilizar la subrutina DSNTIAR para convertir un código de retorno SQL en un mensaje de texto. DSNTIAR toma datos de SQLCA, los formatea en un mensaje y coloca el resultado en un área de salida de mensajes que usted proporciona en su programa de aplicación. Para conceptos y más información sobre el comportamiento de DSNTIAR, consulte “[Visualización de campos SQLCA invocando DSNTIAR](#)” en la página 558.

Sintaxis DSNTIAR

DSNTIAR tiene la siguiente sintaxis:

```
rc = DSNTIAR(&sqlca, &message, &lrecl);
```

Parámetros para DSNTIAR

Los parámetros DSNTIAR tienen los siguientes significados:

&sqlca

Un área de comunicación SQL.

&mensaje

Un área de salida, en formato VARCHAR, en la que DSNTIAR coloca el texto del mensaje. La primera semicifra contiene la longitud del área restante; su valor mínimo es 240.

Las líneas de salida de texto, cada una con la longitud especificada en *&lrecl*, se colocan en esta área. Por ejemplo, podría especificar el formato del área de salida como:

```
#define data_len 132
#define data_dim 10
int length_of_line = data_len ;
struct error_struct {
    short int error_len;
    char error_text[data_dim][data_len];
} error_message = {data_dim * data_len};
:
rc = DSNTIAR(&sqlca, &error_message, &length_of_line);
```

donde *error_message* es el nombre del área de salida del mensaje, *data_dim* es el número de líneas en el área de salida del mensaje y *data_len* es la longitud de cada línea.

&lrecl

Una palabra completa que contiene la longitud de registro lógica de los mensajes de salida, en el rango de 72 a 240.

Para informar a su compilador de que DSNTIAR es un programa en lenguaje ensamblador, incluya una de las siguientes declaraciones en su aplicación.

Para C, incluya:

```
#pragma linkage (DSNTIAR,OS)
```

Para C++, incluya una declaración similar a esta:

```
extern "OS" short int DSNTIAR(struct sqlca *sqlca,
                           struct error_struct *error_message,
                           int *data_len);
```

En el programa de ejemplo C de Db2 DSN8BD3 y en el programa de ejemplo C++ DSN8BE3 aparecen ejemplos de llamadas a DSNTIAR desde una aplicación. Ambos están en la biblioteca DSN8C10.SDSENSAMP. Visite “[Aplicaciones de ejemplo suministradas con Db2 for z/OS](#)” en la página 1091 para obtener instrucciones sobre cómo acceder e imprimir el código fuente de los programas de muestra.

- Si su CICS solicitud requiere CICS almacenamiento, debe utilizar la subrutina DSNTIAC en lugar de DSNTIAR.

Sintaxis DSNTIAC

DSNTIAC tiene la siguiente sintaxis:

```
rc = DSNTIAC(&eib, &commarea, &sqlca, &message, &lrecl);
```

Parámetros para DSNTIAC

DSNTIAC tiene parámetros adicionales, que debe utilizar para llamadas a rutinas que utilizan CICS comandos.

&eib

Bloque de interfaz EXEC

&commarea

área de comunicación

Para obtener más información sobre estos parámetros, consulte la guía de programación de aplicaciones correspondiente para CICS. Las descripciones de los parámetros restantes son las

mismas que las de DSNTIAR. Tanto DSNTIAC como DSNTIAR formatean el SQLCA de la misma manera.

Debe definir DSNTIA1 en el CSD. Si carga DSNTIAR o DSNTIAC, también debe definirlos en el CSD. Para ver un ejemplo de declaraciones de generación de entradas CSD para su uso con DSNTIAC, consulte el trabajo DSNTEJ5A.

El código fuente ensamblador para DSNTIAC y el trabajo DSNTEJ5A, que ensambla y edita enlaces DSNTIAC, están en *el prefijo* del conjunto de datos.SDSENSAMP.

- También puede utilizar el campo de elemento de condición MESSAGE_TEXT de la instrucción GET DIAGNOSTICS para convertir un código de retorno SQL en un mensaje de texto.

Los programas que requieren un soporte de mensajes token largos deben codificar la instrucción GET DIAGNOSTICS en lugar de DSNTIAR. Para obtener más información sobre GET DIAGNOSTICS, consulte “[Comprobación de la ejecución de sentencias SQL utilizando la instrucción GET DIAGNOSTICS](#)” en la página 563.

Tareas relacionadas

[Manejo de códigos de error de SQL](#)

Los programas de aplicación pueden solicitar más información sobre los códigos de error SQL en Db2.

Referencia relacionada

[GET DIAGNOSTICS declaración \(Db2 SQL\)](#)

Aplicaciones COBOL que emiten sentencias SQL

Puede codificar sentencias SQL en determinadas secciones del programa COBOL.

Las secciones permitidas se muestran en la siguiente tabla.

Tabla 106. Instrucciones SQL permitidas para secciones de programa COBOL

Sentencia SQL	Sección del programa
BEGIN DECLARE SECTION END DECLARE SECTION	SECCIÓN DE TRABAJO-ALMACENAMIENTO1 o SECCIÓN DE ENLACE
INCLUIR SQLCA	SECCIÓN DE TRABAJO-ALMACENAMIENTO1 o SECCIÓN DE ENLACE
INCLUIR nombre-archivo-texto	DIVISIÓN DE PROCEDIMIENTOS o DIVISIÓN DE DATOS2
DECLARE TABLE DECLARE CURSOR	DIVISIÓN DE DATOS o DIVISIÓN DE PROCEDIMIENTOS
DECLARE VARIABLE	SECCIÓN 1 : TRABAJO-ALMACENAMIENTO
Otros	PROCEDURE DIVISION

Notas:

1. Si utiliza el coprocesador Db2 , puede utilizar la SECCIÓN DE ALMACENAMIENTO LOCAL siempre que la SECCIÓN DE ALMACENAMIENTO DE TRABAJO aparezca en la tabla.
2. Al incluir declaraciones de variables de host, la instrucción INCLUDE debe estar en la SECCIÓN WORKING-STORAGE o en la SECCIÓN LINKAGE.

No se pueden poner instrucciones SQL en la sección DECLARATIVES de un programa COBOL.

Cada instrucción SQL en un programa COBOL debe comenzar con EXEC SQL y terminar con END-EXEC. Si utiliza el precompilador Db2 , las palabras clave EXEC y SQL deben aparecer en una línea, pero el resto de la instrucción puede aparecer en líneas posteriores. Si utiliza el coprocesador de EXEC y SQL (Db2),

las palabras clave EXEC y SQL pueden estar en líneas diferentes. No incluya ningún símbolo entre las dos palabras clave EXEC y SQL, excepto en los comentarios COBOL, incluidas las líneas de depuración. No incluya comentarios SQL entre las palabras clave EXEC y SQL.

Si la instrucción SQL aparece entre dos instrucciones COBOL, el punto después de END-EXEC es opcional y podría no ser apropiado. Si la declaración aparece en un SI.A continuación, conjunto de sentencias COBOL, omita el punto final para evitar terminar inadvertidamente la sentencia IF.

Podría codificar una instrucción UPDATE en un programa COBOL de la siguiente manera:

```
EXEC SQL  
  UPDATE DSN8C10_DEPT  
    SET MGRNO = :MGR-NUM  
    WHERE DEPTNO = :INT-DEPT  
END-EXEC.
```

Comentarios

Puede incluir líneas de comentario COBOL (* en la columna 7) en instrucciones SQL siempre que pueda utilizar un espacio en blanco.

Además, puede incluir comentarios SQL ('--) en cualquier instrucción SQL incrustada. Debe haber un espacio antes de los dos guiones ('--) que comienzan el comentario. Para obtener más información, consulte [Comentarios SQL \(Db2 SQL\)](#).

Restricciones: Si utiliza el precompilador Db2 , tenga en cuenta las siguientes restricciones:

- No puede incluir líneas de comentario COBOL entre las palabras clave EXEC y SQL. El precompilador trata las líneas de depuración COBOL y las líneas de expulsión de página (/ en la columna 7) como líneas de comentario. El coprocesador COBOL (Db2) trata las líneas de depuración basándose en las reglas COBOL, que dependen de la configuración del modo WITH DEBUGGING.
- No puede utilizar comentarios en línea COBOL que estén identificados por un indicador de comentario flotante (*>). Los comentarios en línea COBOL se interpretan correctamente solo cuando se utiliza el coprocesador de optimización de memoria (Db2).

Para una sentencia SQL INCLUDE, el precompilador Db2 trata cualquier texto que siga al punto después de END-EXEC, y en la misma línea que END-EXEC, como un comentario. El coprocesador COBOL (Db2) trata este texto como parte de la sintaxis del programa COBOL.

Líneas de depuración

El precompilador de Db2 ignora la «D» de la columna 7 en las líneas de depuración y la trata como un espacio en blanco. El coprocesador COBOL (Db2) sigue las reglas del lenguaje COBOL en lo que respecta a las líneas de depuración.

Continuación para sentencias SQL

Las reglas para continuar una constante de cadena de caracteres de una línea a la siguiente en una instrucción SQL incrustada en un programa COBOL son las mismas que para continuar un literal no numérico en COBOL. Sin embargo, puede utilizar una comilla o un apóstrofe como primer carácter no en blanco en el área B de la línea de continuación. La misma regla se aplica a la continuación de identificadores delimitados y no depende de la opción de delimitador de cadena.

Para cumplir con el estándar SQL, delimite una constante de cadena de caracteres con un apóstrofe y utilice una comilla como primer carácter no en blanco en el área B de la línea de continuación para una constante de cadena de caracteres.

Las líneas continuas de una instrucción SQL pueden estar en las columnas 8-72 cuando se utiliza el precompilador " Db2 " y en las columnas 12-72 cuando se utiliza el coprocesador " Db2 ".

Delimitadores

Delimite una instrucción SQL en su programa COBOL con la palabra clave de inicio " EXEC SQL " y un " END-EXEC " como se muestra en el siguiente código de ejemplo:

```
EXEC SQL  
  SQL-statement  
END-EXEC.
```

COPIAR

Si utiliza el precompilador Db2 , no utilice una instrucción COBOL COPY dentro de las declaraciones de variables de host. Si utiliza el coprocesador COBOL (Db2), puede utilizar COBOL COPY.

REPLACE

Si utiliza el precompilador Db2 , la instrucción REPLACE no tiene efecto en las instrucciones SQL. Afecta únicamente a las sentencias COBOL que genera el precompilador.

Si utiliza el coprocesador Db2 , la instrucción REPLACE sustituye cadenas de texto en instrucciones SQL, así como en instrucciones COBOL generadas.

Declaración de tablas y vistas

Su programa COBOL debe incluir la instrucción DECLARE TABLE para describir cada tabla y ver los accesos del programa. Puede utilizar el generador de declaraciones de Db2 (DCLGEN) para generar las declaraciones DECLARE TABLE. Debe incluir a los miembros de DCLGEN en la DIVISIÓN DE DATOS.

SQL dinámico en un programa COBOL

En general, los programas COBOL pueden manejar fácilmente sentencias SQL dinámicas. Los programas COBOL pueden manejar sentencias SELECT si los tipos de datos y el número de campos devueltos son fijos. Si desea utilizar instrucciones SELECT de lista variable, utilice un SQLDA.

Incluir código

Para incluir sentencias SQL o declaraciones de variables de host COBOL de un miembro de un conjunto de datos particionado, utilice la siguiente sentencia SQL en el código fuente donde desee incluir las sentencias:

```
EXEC SQL INCLUDE member-name END-EXEC.
```

Si está utilizando el precompilador Db2 , no puede anidar sentencias SQL INCLUDE. En este caso, no utilice verbos COBOL para incluir sentencias SQL o declaraciones de variables de host, y no utilice la sentencia SQL INCLUDE para incluir CICS código relacionado con el preprocesador. En general, si está utilizando el precompilador Db2 , utilice la instrucción SQL INCLUDE solo para la codificación relacionada con SQL. Si utiliza el coprocesador COBOL Db2 , no se aplican estas restricciones.

Utilice el par de palabras clave 'EXEC SQL' y 'END-EXEC' para incluir únicamente sentencias SQL. No se permiten instrucciones COBOL, como COPY o REPLACE.

Márgenes

Debe codificar las sentencias SQL que comienzan con EXEC SQL en las columnas 12-72. De lo contrario, el precompilador de Db2 no reconoce la instrucción SQL.

Nombres

Puede utilizar cualquier nombre COBOL válido para una variable de host. No utilice nombres de entrada externos ni nombres de planes de acceso que empiecen por «DSN», ni nombres de variables de host que empiecen por «SQL». Estos nombres están reservados para Db2.

Números de secuencia

Las sentencias fuente que genera el precompilador de Db2 no incluyen números de secuencia.

Etiquetas de declaración

Puede preceder las instrucciones SQL ejecutables en la DIVISIÓN PROCEDIMIENTO con un nombre de párrafo.

WHENEVER, sentencia

El objetivo de la cláusula GOTO en una instrucción SQL WHENEVER debe ser un nombre de sección o un nombre de párrafo no cualificado en la DIVISIÓN PROCEDIMIENTO.

Consideraciones especiales sobre COBOL

Las siguientes consideraciones se aplican a los programas escritos en COBOL:

- En un programa COBOL que utiliza elementos en una estructura multinivel como nombres de variables de host, el precompilador Db2 genera los nombres de dos niveles más bajos.
- El uso de las opciones DYNAM y NODYNAM del compilador COBOL depende del entorno operativo.

TSO y IMS : Puede especificar la opción DYNAM al compilar un programa COBOL si utiliza las siguientes directrices. IMS y Db2 comparten un nombre de alias común, DSNHLI, para el módulo de interfaz de idioma. Debe hacer lo siguiente cuando concatene sus bibliotecas:

- Si utiliza IMS con la opción DYNAM de COBOL, asegúrese de concatenar primero la IMS biblioteca primero.
- Si ejecuta su programa de aplicación solo en , asegúrese de concatenar primero la biblioteca Db2 .

CICS, CAF y RRSAF : Debe especificar la opción NODYNAM cuando compile un programa COBOL que incluya CICS instrucciones o se traduce mediante un CICS o el integrado CICS traductor. En estos casos, no puede especificar la opción DYNAM. Si su CICS programa tiene una subrutina que no está traducida por un CICS traductor independiente o el traductor integrado, pero contiene instrucciones SQL, puede especificar la opción DYNAM CICS, pero contiene sentencias SQL, puede especificar la opción DYNAM. Sin embargo, en este caso, debe concatenar las CICS bibliotecas antes de las bibliotecas de Db2 .

Puede compilar procedimientos almacenados COBOL con la opción DYNAM o la opción NODYNAM. Si utiliza DYNAM, asegúrese de que el módulo de interfaz de idioma de Db2 correcto se carga dinámicamente realizando una de las siguientes acciones:

- Utilice la opción de precompilador ATTACH(RRSAF).
- Copie el módulo DSNRLI en una biblioteca de carga que esté concatenada delante de las bibliotecas de Db2 . Utilice el nombre de miembro DSNHLI.
- Para evitar truncar los valores numéricos, utilice uno de los siguientes métodos:
 - Utilice el tipo de datos " COMP-5 " para variables de host enteras binarias.
 - Especifique la opción del compilador COBOL:
 - TRUNC(OPT) o TRUNC(STD) si está seguro de que los datos que la aplicación mueve a cada variable binaria no tienen una precisión mayor que la definida en la cláusula PICTURE de la variable binaria.
 - TRUNC(BIN) si la precisión de los datos que se mueven a cada variable binaria puede exceder el valor en la cláusula PICTURE.

Db2 asigna valores a variables de host enteras binarias como si hubiera especificado la opción del compilador COBOL TRUNC(BIN) o utilizando el tipo de datos COMP-5.

- Si está utilizando el precompilador Db2 y su programa COBOL contiene varios puntos de entrada o se llama varias veces, la cláusula USING de la sentencia de entrada que se ejecuta antes de que se ejecute la primera sentencia SQL debe contener el SQLCA y todas las entradas de la sección de vinculación que cualquier sentencia SQL utiliza como variables de host.
- Si utiliza el precompilador Db2 , no deben aparecer directivas de compilación entre la instrucción PROCEDURE DIVISION y DECLARATIVES.
- No utilice constantes figurativas COBOL (como CERO y ESPACIO), caracteres simbólicos, modificación de referencia ni subíndices en las sentencias SQL.
- Observe las reglas para nombrar identificadores SQL, como se describe en [Identificadores en SQL \(Db2 SQL\)](#). Sin embargo, solo para COBOL, los nombres de los identificadores SQL pueden seguir las reglas para nombrar palabras COBOL, como se describe en [Palabras de COBOL con caracteres de un solo byte \(COBOL\) \(Guía de programación de Enterprise COBOL for z/OS\)](#). Sin embargo, los nombres no deben exceder la longitud permitida para el objeto Db2 .
- Envuelve los guiones utilizados como operadores de resta con espacios. Db2 suele interpretar un guion sin espacios a su alrededor como parte del nombre de una variable anfitriona.
- Si incluye una instrucción SQL en un párrafo COBOL PERFORM ... THRU y también especifica la instrucción SQL WHENEVER ... GO, el compilador COBOL devuelve el mensaje de advertencia IGYOP3094. Ese mensaje podría indicar un problema. No se recomienda este uso.

- Si utiliza el precompilador Db2 , todas las sentencias SQL y las variables de host a las que hagan referencia deben estar dentro del primer programa cuando se utilicen programas anidados o compilación por lotes.
- Si utiliza el precompilador Db2 , sus programas COBOL deben tener una DATA DIVISION y una PROCEDURE DIVISION. Tanto las divisiones como la SECCIÓN DE ALMACENAMIENTO DE TRABAJO deben estar presentes en los programas que contengan sentencias SQL. Sin embargo, si sus programas COBOL requieren la LOCAL-STORAGE SECTION, entonces se debe utilizar el coprocesador Db2 en lugar del precompilador Db2 .
- El precompilador COBOL (Db2) genera esta variable COBOL:

```
DSN-TMP2 PIC S9(18) COMP-3
```

El coprocesador COBOL (Db2) genera esta variable COBOL:

```
SQL---SCRVALD DS 10P PIC S9(18) COMP-3
```

Si especifica la opción COBOL RULES(NOEVENPACK), el compilador COBOL genera la advertencia IGYDS1348-W, porque esas variables tienen un número par de dígitos decimales empaquetados.

PSPI Si su programa utiliza el precompilador Db2 y utiliza parámetros que están definidos en LINKAGE SECTION como variables de host para Db2 y la dirección del parámetro de entrada puede cambiar en invocaciones posteriores de su programa, su programa debe restablecer la variable SQL-INIT-FLAG. Esta marca la genera el precompilador de Db2 . Restablecer esta marca indica que el almacenamiento debe inicializarse cuando se ejecute la siguiente instrucción SQL. Para restablecer el indicador, inserte la instrucción MOVE ZERO TO SQL-INIT-FLAG en la DIVISIÓN PROCEDIMIENTO del programa llamado, antes de cualquier instrucción SQL ejecutable que utilice las variables del host. Si utiliza el coprocesador COBOL Db2 , el programa llamado no necesita restablecer SQL-INIT-FLAG. **PSPI**

Manejo de códigos de error de SQL

Las aplicaciones Cobol pueden solicitar más información sobre errores SQL en Db2. Para obtener más información, consulte “[Gestión de códigos de error de SQL en aplicaciones Cobol](#)” en la página 721.

Conceptos relacionados

[Ejemplos de aplicaciones suministrados con Db2 for z/OS](#)

Db2 proporciona ejemplos de aplicaciones para ayudarle con las técnicas de programación e Db2 es y las prácticas de codificación dentro de cada uno de los cuatro entornos: batch, TSO, IMS, y CICS. Las aplicaciones de ejemplo contienen varias aplicaciones que pueden aplicarse a la gestión de una empresa.

[DCLGEN \(generador de declaraciones\)](#)

El programa debe declarar las tablas y vistas a las que accede. El generador de declaraciones de Db2, DCLGEN, crea estas sentencias DECLARE para programas C, COBOL y PL/I programs, y así no necesita codificar las sentencias él mismo. DCLGEN genera también las estructuras de variable de lenguaje principal correspondientes.

[Utilización de matrices de variables de host en sentencias SQL](#)

Utilice matrices de variables de lenguaje de host en sentencias de SQL incorporado para representar valores que el programa no conoce hasta que se ejecuta la consulta. Las matrices de variables de host son útiles para almacenar un conjunto de valores recuperados o para pasar un conjunto de valores que se van a insertar en una tabla.

[Identificadores en SQL \(Db2 SQL\)](#)

Tareas relacionadas

[Descripción general de las aplicaciones de programación que acceden a datos de Db2 for z/OS](#)

Las aplicaciones que interactúan con Db2, en primer lugar se deben conectar a Db2. Entonces pueden leer, añadir o modificar datos o manipular objetos de Db2.

[Inclusión de SQL dinámico en el programa](#)

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

[Comprobación de la ejecución de sentencias SQL utilizando la instrucción GET DIAGNOSTICS](#)

Una forma de comprobar si una sentencia SQL se ha ejecutado correctamente es solicitar a Db2 que devuelva información de diagnóstico sobre la última sentencia SQL ejecutada.

Definición de áreas de descriptor de SQL (SQLDA)

Si su programa incluye ciertas sentencias SQL, debe definir al menos *un área de descriptor SQL (SQLDA)*. Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Visualización de campos SQLCA invocando DSNTIAR

Si utiliza el SQLCA para comprobar si una sentencia SQL se ha ejecutado correctamente, el programa necesita leer los datos de los campos SQLCA adecuados. Una forma sencilla de leer estos campos es utilizar la subrutina de ensamblador DSNTIAR.

Establecimiento de límites para el uso de recursos de sistema utilizando el recurso de límite de recursos (Db2 Performance)

Ejemplos de programación de COBOL

Puede escribir programas de Db2 en COBOL. Estos programas pueden acceder a un subsistema Db2 local o remoto y pueden ejecutar sentencias de SQL estático o dinámico. Esta información contiene varios ejemplos de programación de ese tipo.

Para preparar y ejecutar estas aplicaciones, utilice el JCL de *prefijo.SDSNSAMP* como modelo para su JCL.

Referencia relacionada

[Ejemplos de programación de Assembler, C, C++, COBOL, PL/I y REXX \(Db2 Programming samples\)](#)

Programa de SQL dinámico COBOL de ejemplo

Puede codificar sentencias SELECT de lista variable dinámicas en un programa COBOL. *Las sentencias SELECT de lista variable* son sentencias para las que no se conoce el número o los tipos de datos de las columnas que se van a devolver al escribir el programa.

Conceptos introductorios

[Envío de sentencias SQL a Db2 \(Introducción a Db2 para z/OS\)](#)

[Aplicaciones de SQL dinámico \(Introducción a Db2 para z/OS\)](#)

“[Inclusión de SQL dinámico en el programa](#)” en la página 524 describe tres variaciones de sentencias SQL dinámicas:

- Declaraciones no SELECT
- Instrucciones SELECT de lista fija

En este caso, usted conoce el número de columnas devueltas y sus tipos de datos cuando escribe el programa.

- Declaraciones SELECT de lista variable.

En este caso, **no se conoce** el número de columnas devueltas ni sus tipos de datos al escribir el programa.

Esta sección documenta una técnica de codificación de instrucciones SELECT de listas variables en COBOL.

Este programa de ejemplo no admite tipos de datos BLOB, CLOB o DBCLOB.

Punteros y variables basadas en el programa COBOL de muestra

COBOL tiene una instrucción de tipo PUNTERO y una instrucción SET que proporcionan punteros y variables basadas.

La sentencia SET establece un puntero desde la dirección de un área en la sección de enlace u otro puntero; la sentencia también puede establecer la dirección de un área en la sección de enlace.

UNLDBCU2 “Ejemplo del programa COBOL de muestra” en la página 660 proporciona estos usos de la instrucción SET. La instrucción SET no permite el uso de una dirección en la sección WORKING-STORAGE.

Asignación de almacenamiento para el programa COBOL de muestra

COBOL no proporciona un medio para asignar el almacenamiento principal dentro de un programa. Puede lograr el mismo fin teniendo un programa inicial que asigne el almacenamiento y luego llame a un segundo programa que manipule el puntero. (COBOL no permite manipular directamente el puntero porque es probable que se produzcan errores y fallos)

El programa inicial es extremadamente sencillo. Incluye una sección de almacenamiento funcional que asigna la máxima cantidad de almacenamiento necesaria. Este programa llama entonces al segundo programa, pasando el área o áreas en la instrucción CALL. El segundo programa define el área en la sección de enlace y puede entonces utilizar punteros dentro del área.

Si necesita asignar partes de almacenamiento, el mejor método es utilizar índices o subíndices. Puede utilizar subíndices para operaciones aritméticas y de comparación.

Ejemplo del programa COBOL de muestra

El siguiente ejemplo muestra un ejemplo del programa inicial UNLDBCUI que asigna el almacenamiento y llama al segundo programa UNLDBCUI2. UNLDBCUI2 y, a continuación, define las áreas de almacenamiento pasadas en su sección de vinculación e incluye la cláusula USING en su instrucción PROCEDURE DIVISION.

Definir los punteros y, a continuación, redefinirlos como numéricos, permite realizar algunas manipulaciones de los punteros que no se pueden realizar directamente. Por ejemplo, no puede añadir la longitud de la columna al puntero de registro, pero puede añadir la longitud de la columna al valor numérico que redefine el puntero.

El siguiente ejemplo es el programa inicial que asigna almacenamiento.

```
**** UNLDBCUI- DB2 SAMPLE BATCH COBOL UNLOAD PROGRAM ****
*
* MODULE NAME = UNLDBCUI
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
*                   UNLOAD PROGRAM
*                   BATCH
*                   IBM ENTERPRISE COBOL FOR Z/OS
*
* COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1987
* REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
*
* STATUS = VERSION 1 RELEASE 3, LEVEL 0
*
* FUNCTION = THIS MODULE PROVIDES THE STORAGE NEEDED BY
*             UNLDBCUI2 AND CALLS THAT PROGRAM.
*
* NOTES =
*     DEPENDENCIES = ENTERPRISE COBOL FOR Z/OS IS REQUIRED.
*                   SEVERAL NEW FACILITIES ARE USED.
*
* RESTRICTIONS =
*     THE MAXIMUM NUMBER OF COLUMNS IS 750,
*     WHICH IS THE SQL LIMIT.
*
*     DATA RECORDS ARE LIMITED TO 32700 BYTES,
*     INCLUDING DATA, LENGTHS FOR VARCHAR DATA,
*     AND SPACE FOR NULL INDICATORS.
*
* MODULE TYPE = IBM ENTERPRISE COBOL PROGRAM
*   PROCESSOR = ENTERPRISE COBOL FOR Z/OS
*   MODULE SIZE = SEE LINK EDIT
* ATTRIBUTES = REENTRANT
*
* ENTRY POINT = UNLDBCUI
*   PURPOSE = SEE FUNCTION
*   LINKAGE = INVOKED FROM DSN RUN
*   INPUT = NONE
*   OUTPUT = NONE
```

```

*      EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION
*
*      EXIT-ERROR =
*          RETURN CODE = NONE
*          ABEND CODES = NONE
*          ERROR-MESSAGES = NONE
*
*      EXTERNAL REFERENCES =
*          ROUTINES/SERVICES =
*              UNLDBCU2 - ACTUAL UNLOAD PROGRAM
*
*          DATA-AREAS      =    NONE
*          CONTROL-BLOCKS =    NONE
*
*          TABLES = NONE
*          CHANGE-ACTIVITY = NONE
*
*      *PSEUDOCODE*
*
*          PROCEDURE
*              CALL UNLDBC2.
*          END.
*-----
*/
IDENTIFICATION DIVISION.
*-----
PROGRAM-ID.    UNLDBC1
*
ENVIRONMENT DIVISION.
*
CONFIGURATION SECTION.
DATA DIVISION.
*
WORKING-STORAGE SECTION.
*
01 WORKAREA-IND.
    02 WORKIND PIC S9(4) COMP-5 OCCURS 750 TIMES.
01 RECWORK.
    02 RECWORK-LEN PIC S9(8) COMP-5 VALUE 32700.
    02 RECWORK-CHAR PIC X(1) OCCURS 32700 TIMES.
*
PROCEDURE DIVISION.
*
    CALL 'UNLDBC2' USING WORKAREA-IND RECWORK.
    GOBACK.

```

El siguiente ejemplo es el programa llamado que realiza la manipulación del puntero.

```

***** UNLDBC2- DB2 SAMPLE BATCH COBOL UNLOAD PROGRAM *****
*
*      MODULE NAME = UNLDBC2
*
*      DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
*                          UNLOAD PROGRAM
*                          BATCH
*                          ENTERPRISE COBOL FOR Z/OS
*
*      COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1987
*      REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
*
*      STATUS = VERSION 1 RELEASE 3, LEVEL 0
*
*      FUNCTION = THIS MODULE ACCEPTS A TABLE NAME OR VIEW NAME
*                  AND UNLOADS THE DATA IN THAT TABLE OR VIEW.
*      READ IN A TABLE NAME FROM SYSIN.
*      PUT DATA FROM THE TABLE INTO DD SYSREC01.
*      WRITE RESULTS TO SYSPRINT.
*
*      NOTES =
*          DEPENDENCIES = IBM ENTERPRISE COBOL FOR Z/OS
*                         IS REQUIRED.
*
*      RESTRICTIONS =
*          THE SQLDA IS LIMITED TO 33016 BYTES.
*          THIS SIZE ALLOWS FOR THE DB2 MAXIMUM
*          OF 750 COLUMNS.
*
*          DATA RECORDS ARE LIMITED TO 32700 BYTES,
*          INCLUDING DATA, LENGTHS FOR VARCHAR DATA,

```

```

* AND SPACE FOR NULL INDICATORS.
*
* TABLE OR VIEW NAMES ARE ACCEPTED, AND ONLY
* ONE NAME IS ALLOWED PER RUN.
*
* MODULE TYPE = ENTERPRISE COBOL FOR Z/OS
* PROCESSOR = DB2 PRECOMPILE, COBOL COMPILER
* MODULE SIZE = SEE LINK EDIT
* ATTRIBUTES = REENTRANT
*
* ENTRY POINT = UNLDBC2
* PURPOSE = SEE FUNCTION
* LINKAGE =
* CALL 'UNLDBC2' USING WORKAREA-IND RECWORK.
*
* INPUT = SYMBOLIC LABEL/NAME = WORKAREA-IND
* DESCRIPTION = INDICATOR VARIABLE ARRAY
* 01 WORKAREA-IND.
*   02 WORKIND PIC S9(4) COMP-5 OCCURS 750 TIMES.★
*
* SYMBOLIC LABEL/NAME = RECWORK
* DESCRIPTION = WORK AREA FOR OUTPUT RECORD
* 01 RECWORK.
*   02 RECWORK-LEN PIC S9(8) COMP.
*   02 RECWORK-CHAR PIC X(1) OCCURS 32700 TIMES.★
*
* SYMBOLIC LABEL/NAME = SYSIN
* DESCRIPTION = INPUT REQUESTS - TABLE OR VIEW
*
* OUTPUT = SYMBOLIC LABEL/NAME = SYSPRINT
* DESCRIPTION = PRINTED RESULTS
*
* SYMBOLIC LABEL/NAME = SYSREC01
* DESCRIPTION = UNLOADED TABLE DATA
*
* EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION
* EXIT-ERROR =
*   RETURN CODE = NONE
*   ABEND CODES = NONE
*   ERROR-MESSAGES =
*     DSNT490I SAMPLE COBOL DATA UNLOAD PROGRAM RELEASE 3.0★
*       - THIS IS THE HEADER, INDICATING A NORMAL ★
*       - START FOR THIS PROGRAM. ★
*     DSNT493I SQL ERROR, SQLCODE = NNNNNNNN ★
*       - AN SQL ERROR OR WARNING WAS ENCOUNTERED ★
*       - ADDITIONAL INFORMATION FROM DSNTIAR ★
*       - FOLLOWS THIS MESSAGE. ★
*     DSNT495I SUCCESSFUL UNLOAD XXXXXXXX ROWS OF ★
*       TABLE TTTTTTTT ★
*       - THE UNLOAD WAS SUCCESSFUL. XXXXXXXX IS ★
*       - THE NUMBER OF ROWS UNLOADED. TTTTTTTT ★
*       - IS THE NAME OF THE TABLE OR VIEW FROM ★
*       - WHICH IT WAS UNLOADED. ★
*     DSNT496I UNRECOGNIZED DATA TYPE CODE OF NNNNN ★
*       - THE PREPARE RETURNED AN INVALID DATA ★
*       - TYPE CODE. NNNNN IS THE CODE, PRINTED ★
*       - IN DECIMAL. USUALLY AN ERROR IN ★
*       - THIS ROUTINE OR A NEW DATA TYPE. ★
*     DSNT497I RETURN CODE FROM MESSAGE ROUTINE DSNTIAR ★
*       - THE MESSAGE FORMATTING ROUTINE DETECTED ★
*       - AN ERROR. SEE THAT ROUTINE FOR RETURN ★
*       - CODE INFORMATION. USUALLY AN ERROR IN ★
*       - THIS ROUTINE. ★
*     DSNT498I ERROR, NO VALID COLUMNS FOUND ★
*       - THE PREPARE RETURNED DATA WHICH DID NOT ★
*       - PRODUCE A VALID OUTPUT RECORD. ★
*       - USUALLY AN ERROR IN THIS ROUTINE. ★
*     DSNT499I NO ROWS FOUND IN TABLE OR VIEW ★
*       - THE CHOSEN TABLE OR VIEWS DID NOT ★
*       - RETURN ANY ROWS. ★
*       ERROR MESSAGES FROM MODULE DSNTIAR ★
*         - WHEN AN ERROR OCCURS, THIS MODULE ★
*         - PRODUCES CORRESPONDING MESSAGES. ★
* OTHER MESSAGES:
*   THE TABLE COULD NOT BE UNLOADED. EXITING. ★
*
* EXTERNAL REFERENCES =
*   ROUTINES/SERVICES =
*     DSNTIAR - TRANSLATE SQLCA INTO MESSAGES ★
*   DATA-AREAS = NONE ★
*   CONTROL-BLOCKS = ★
*   SQLCA - SQL COMMUNICATION AREA ★

```

```

*      TABLES = NONE
*      CHANGE-ACTIVITY = NONE
*
*      *PSEUDOCODE*
*      PROCEDURE
*          EXEC SQL DECLARE DT CURSOR FOR SEL END-EXEC.
*          EXEC SQL DECLARE SEL STATEMENT END-EXEC.
*          INITIALIZE THE DATA, OPEN FILES.
*          OBTAIN STORAGE FOR THE SQLDA AND THE DATA RECORDS.
*          READ A TABLE NAME.
*          OPEN SYSREC01.
*          BUILD THE SQL STATEMENT TO BE EXECUTED
*          EXEC SQL PREPARE SQL STATEMENT INTO SQLDA END-EXEC.
*          SET UP ADDRESSES IN THE SQLDA FOR DATA.
*          INITIALIZE DATA RECORD COUNTER TO 0.
*          EXEC SQL OPEN DT END-EXEC.
*          DO WHILE SQLCODE IS 0.
*          EXEC SQL FETCH DT USING DESCRIPTOR SQLDA END-EXEC.
*          ADD IN MARKERS TO DENOTE NULLS.
*          WRITE THE DATA TO SYSREC01.
*          INCREMENT DATA RECORD COUNTER.
*          END.
*          EXEC SQL CLOSE DT END-EXEC.
*          INDICATE THE RESULTS OF THE UNLOAD OPERATION.
*          CLOSE THE SYSIN, SYSPRINT, AND SYSREC01 FILES.
*          END.
*-----
*/
IDENTIFICATION DIVISION.
*-----*
PROGRAM-ID.    UNLDBC2
*
ENVIRONMENT DIVISION.
*-----*
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT SYSIN
        ASSIGN TO DA-S-SYSIN.
    SELECT SYSPRINT
        ASSIGN TO UT-S-SYSPRINT.
    SELECT SYSREC01
        ASSIGN TO DA-S-SYSREC01.
*
DATA DIVISION.
*-----*
*
FILE SECTION.
FD    SYSIN
    RECORD CONTAINS 80 CHARACTERS
    BLOCK CONTAINS 0 RECORDS
    LABEL RECORDS ARE OMITTED
    RECORDING MODE IS F.
01 CARDREC           PIC X(80).
*
FD    SYSPRINT
    RECORD CONTAINS 120 CHARACTERS
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS MSGREC
    RECORDING MODE IS F.
01 MSGREC            PIC X(120).
*
FD    SYSREC01
    RECORD CONTAINS 5 TO 32704 CHARACTERS
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS REC01
    RECORDING MODE IS V.
01 REC01.
    02 REC01-LEN PIC S9(8) COMP.
    02 REC01-CHAR PIC X(1) OCCURS 1 TO 32700 TIMES
        DEPENDING ON REC01-LEN.
/
WORKING-STORAGE SECTION.
*-----*
* ***** STRUCTURE FOR INPUT *****
* ***** 01 IOAREA.
*         02 TNAME      PIC X(72).
*         02 FILLER     PIC X(08).
* 01 STMTBUF.

```

```

        49 STMTLEN      PIC S9(4) COMP-5 VALUE 92.
        49 STMTCHAR    PIC X(92).
01 STMTBLD.
        02 FILLER      PIC X(20) VALUE 'SELECT * FROM'.
        02 STMTTAB     PIC X(72).
*
***** REPORT HEADER STRUCTURE *****
01 HEADER.
        02 FILLER PIC X(35)
          VALUE ' DSNT490I SAMPLE COBOL DATA UNLOAD '.
        02 FILLER PIC X(85) VALUE 'PROGRAM RELEASE 3.0'.
01 MSG-SQLEERR.
        02 FILLER PIC X(31)
          VALUE ' DSNT493I SQL ERROR, SQLCODE = '.
        02 MSG-MINUS   PIC X(1).
        02 MSG-PRINT-CODE PIC 9(8).
        02 FILLER PIC X(81) VALUE '           '.
01 MSG-OTHER-ERR.
        02 FILLER PIC X(42)
          VALUE ' THE TABLE COULD NOT BE UNLOADED. EXITING.'.
        02 FILLER PIC X(78) VALUE '           '.
01 UNLOADED.
        02 FILLER PIC X(28)
          VALUE ' DSNT495I SUCCESSFUL UNLOAD '.
        02 ROWS      PIC 9(8).
        02 FILLER PIC X(15) VALUE ' ROWS OF TABLE '.
        02 TABLENAM  PIC X(72) VALUE '           '.
01 BADTYPE.
        02 FILLER PIC X(42)
          VALUE ' DSNT496I UNRECOGNIZED DATA TYPE CODE OF '.
        02 TYPcod    PIC 9(8).
        02 FILLER PIC X(71) VALUE '           '.
01 MSGRETCD.
        02 FILLER PIC X(42)
          VALUE ' DSNT497I RETURN CODE FROM MESSAGE ROUTINE'.
        02 FILLER PIC X(9) VALUE 'DSNTIAR '.
        02 RETCODE    PIC 9(8).
        02 FILLER PIC X(62) VALUE '           '.
01 MSGNOCOL.
        02 FILLER PIC X(120)
          VALUE ' DSNT498I ERROR, NO VALID COLUMNS FOUND'.
01 MSG-NOROW.
        02 FILLER PIC X(120)
          VALUE ' DSNT499I NO ROWS FOUND IN TABLE OR VIEW'.
*****
* WORKAREAS *
77 NOT-FOUND      PIC S9(8) COMP-5 VALUE +100.
*****
* VARIABLES FOR ERROR-MESSAGE FORMATTING *
01 ERROR-MESSAGE.
        02 ERROR-LEN    PIC S9(4) COMP-5 VALUE +960.
        02 ERROR-TEXT   PIC X(120) OCCURS 8 TIMES
          INDEXED BY ERROR-INDEX.
77 ERROR-TEXT-LEN  PIC S9(8) COMP-5 VALUE +120.
*****
* SQL DESCRIPTOR AREA *
01 SQLDA.
        02 SQLDAID     PIC X(8)  VALUE 'SQLDA   '.
        02 SQLDABC     PIC S9(8) COMPUTATIONAL VALUE 33016.
        02 SQLN       PIC S9(4) COMP-5 VALUE 750.
        02 SQLD       PIC S9(4) COMP-5 VALUE 0.
        02 SQLVAR     OCCURS 1 TO 750 TIMES
          DEPENDING ON SQLN.
        03 SQLTYPE     PIC S9(4) COMP-5.
        03 SQLLEN      PIC S9(4) COMP-5.
        03 SQLDATA    POINTER.
        03 SQLIND     POINTER.
        03 SQLNAME.
          49 SQLNAMEL   PIC S9(4) COMP-5.
          49 SQLNAMEC   PIC X(30).
*
* DATA TYPES FOUND IN SQLTYPE, AFTER REMOVING THE NULL BIT
*
77 VARCTYPE      PIC S9(4) COMP-5 VALUE +448.
77 CHARTYPE      PIC S9(4) COMP-5 VALUE +452.
77 VARLTYPE      PIC S9(4) COMP-5 VALUE +456.
77 VARGTYPE      PIC S9(4) COMP-5 VALUE +464.

```

```

77 GTYPE          PIC S9(4) COMP-5 VALUE +468.
77 LVARGTYP      PIC S9(4) COMP-5 VALUE +472.
77 FLOATTYPE     PIC S9(4) COMP-5 VALUE +480.
77 DECTYPE       PIC S9(4) COMP-5 VALUE +484.
77 INTTYPE       PIC S9(4) COMP-5 VALUE +496.
77 HWTYPE        PIC S9(4) COMP-5 VALUE +500.
77 DATETYP       PIC S9(4) COMP-5 VALUE +384.
77 TIMETYP       PIC S9(4) COMP-5 VALUE +388.
77 TIMESTMP      PIC S9(4) COMP-5 VALUE +392.
*
* 01 RECPTR POINTER.
01 RECNUM REDEFINES RECPTR PICTURE S9(8) COMPUTATIONAL.
01 IRECPTR POINTER.
01 IRECNUM REDEFINES IRECPTR PICTURE S9(8) COMPUTATIONAL.
01 I   PICTURE S9(4) COMPUTATIONAL.
01 J   PICTURE S9(4) COMPUTATIONAL.
01 DUMMY PICTURE S9(4) COMPUTATIONAL.
01 MYTYPE PICTURE S9(4) COMPUTATIONAL.
01 COLUMN-IND PICTURE S9(4) COMPUTATIONAL.
01 COLUMN-LEN PICTURE S9(4) COMPUTATIONAL.
01 COLUMN-PREC PICTURE S9(4) COMPUTATIONAL.
01 COLUMN-SCALE PICTURE S9(4) COMPUTATIONAL.
01 INDCOUNT      PIC S9(4) COMPUTATIONAL.
01 ROWCOUNT      PIC S9(4) COMPUTATIONAL.
01 ERR-FOUND PICTURE X(1).
01 WORKAREA2.
    02 WORKINDPTR POINTER OCCURS 750 TIMES.
*****
*  DECLARE CURSOR AND STATEMENT FOR DYNAMIC SQL
*****
*
*           EXEC SQL DECLARE DT CURSOR FOR SEL END-EXEC.
*           EXEC SQL DECLARE SEL STATEMENT END-EXEC.
*
*****
* SQL INCLUDE FOR SQLCA *
*****
*           EXEC SQL INCLUDE SQLCA END-EXEC.
*
77 ONE          PIC S9(4) COMP-5 VALUE +1.
77 TWO          PIC S9(4) COMP-5 VALUE +2.
77 FOUR         PIC S9(4) COMP-5 VALUE +4.
77 QMARK        PIC X(1)  VALUE '?'.
*
LINKAGE SECTION.
01 LINKAREA-IND.
    02 IND  PIC S9(4) COMP-5 OCCURS 750 TIMES.
01 LINKAREA-REC.
    02 REC1-LEN PIC S9(8) COMP.
    02 REC1-CHAR PIC X(1) OCCURS 1 TO 32700 TIMES
        DEPENDING ON REC1-LEN.
01 LINKAREA-QMARK.
    02 INDREC PIC X(1).
/
PROCEDURE DIVISION USING LINKAREA-IND LINKAREA-REC.
*
*****
* SQL RETURN CODE HANDLING *
*****
*           EXEC SQL WHENEVER SQLERROR GOTO DBERROR END-EXEC.
*           EXEC SQL WHENEVER SQLWARNING GOTO DBERROR END-EXEC.
*           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
*
*****
* MAIN PROGRAM ROUTINE *
*****
*           SET IRECPTR TO ADDRESS OF REC1-CHAR(1).
*                           **OPEN FILES
*           MOVE 'N' TO ERR-FOUND.                      **INITIALIZE
*           OPEN INPUT  SYSIN                         ** ERROR FLAG
*
*           OUTPUT SYSPRINT
*           OUTPUT SYSREC01.                         **WRITE HEADER
*           WRITE MSGREC FROM HEADER
*               AFTER ADVANCING 2 LINES.             **GET FIRST INPUT
*           READ SYSIN RECORD INTO IOAREA.          **MAIN ROUTINE
*           PERFORM PROCESS-INPUT THROUGH IND-RESULT.

```

```

*
* PROG-END.
*
*          CLOSE SYSIN                         **CLOSE FILES
*          SYSPRINT
*          SYSREC01.
*          GOBACK.
/
*****                                                 *
*          PERFORMED SECTION:                  *
*          PROCESSING FOR THE TABLE OR VIEW JUST READ   *
*          *                                             *
*****                                                 *
*          PROCESS-INPUT.                         *
*
*          MOVE TNAME TO STMTTAB.
*          MOVE STMTBLD TO STMTCHAR.
*          MOVE +750 TO SQLN.
*          EXEC SQL PREPARE SEL INTO :SQLDA FROM :STMTBUF END-EXEC.
*****                                                 *
*          SET UP ADDRESSES IN THE SQLDA FOR DATA.      *
*          *                                             *
*****                                                 *
*          IF SQLD = ZERO THEN                      *
*              WRITE MSGREC FROM MSGNOCOL           *
*                  AFTER ADVANCING 2 LINES
*              MOVE 'Y' TO ERR-FOUND
*              GO TO IND-RESULT.
*          MOVE ZERO TO ROWCOUNT.
*          MOVE ZERO TO REC1-LEN.
*          SET RECPTR TO IRECPTR.
*          MOVE ONE TO I.
*          PERFORM COLADDR UNTIL I > SQLD.
*****                                                 *
*          SET LENGTH OF OUTPUT RECORD.            *
*          EXEC SQL OPEN DT END-EXEC.             *
*          DO WHILE SQLCODE IS 0.                  *
*              EXEC SQL FETCH DT USING DESCRIPTOR :SQLDA END-EXEC. *
*              ADD IN MARKERS TO DENOTE NULLS.       *
*              WRITE THE DATA TO SYSREC01.          *
*              INCREMENT DATA RECORD COUNTER.        *
*          END.                                     *
*          *                                             *
*****                                                 *
*          **OPEN CURSOR
*          EXEC SQL OPEN DT END-EXEC.
*          PERFORM BLANK-REC.
*          EXEC SQL FETCH DT USING DESCRIPTOR :SQLDA END-EXEC.
*          *                                              **NO ROWS FOUND
*          *                                              **PRINT ERROR MESSAGE
*          IF SQLCODE = NOT-FOUND
*              WRITE MSGREC FROM MSG-NOROW
*                  AFTER ADVANCING 2 LINES
*              MOVE 'Y' TO ERR-FOUND
*          ELSE
*              *                                              **WRITE ROW AND
*              *                                              **CONTINUE UNTIL
*              *                                              **NO MORE ROWS
*              PERFORM WRITE-AND-FETCH
*                  UNTIL SQLCODE IS NOT EQUAL TO ZERO.
*          EXEC SQL WHENEVER NOT FOUND GOTO CLOSED DT    END-EXEC.
*
*          CLOSED DT.
*          EXEC SQL CLOSE DT END-EXEC.
*
*****                                                 *
*          INDICATE THE RESULTS OF THE UNLOAD OPERATION. *
*          *                                             *
*****                                                 *
IND-RESULT.
IF ERR-FOUND = 'N' THEN
    MOVE TNAME TO TABLENAM
    MOVE ROWCOUNT TO ROWS
    WRITE MSGREC FROM UNLOADED
        AFTER ADVANCING 2 LINES
ELSE
    WRITE MSGREC FROM MSG-OTHER-ERR

```

```

        AFTER ADVANCING 2 LINES
MOVE +0012 TO RETURN-CODE
GO TO PROG-END.

*
* WRITE-AND-FETCH.
*   ADD IN MARKERS TO DENOTE NULLS.
*   MOVE ONE TO INDCOUNT.
*   PERFORM NULLCHK UNTIL INDCOUNT = SQLD.
*   MOVE REC1-LEN TO REC01-LEN.
*   WRITE REC01 FROM LINKAREA-REC.
*   ADD ONE TO ROWCOUNT.
*   PERFORM BLANK-REC.
*   EXEC SQL FETCH DT USING DESCRIPTOR :SQLDA END-EXEC.

*
* NULLCHK.
*   IF IND(INDCOUNT) < 0 THEN
*     SET ADDRESS OF LINKAREA-QMARK TO WORKINDPTR(INDCOUNT)
*     MOVE QMARK TO INDREC.
*     ADD ONE TO INDCOUNT.
*****
*   BLANK OUT RECORD TEXT FIRST *
*****
BLANK-REC.
*     MOVE ONE TO J.
*     PERFORM BLANK-MORE UNTIL J > REC1-LEN.

BLANK-MORE.
*     MOVE ' ' TO REC1-CHAR(J).
*     ADD ONE TO J.

*
* COLADDR.
*   SET SQLDATA(I) TO RECPTR.
*****
*
*   DETERMINE THE LENGTH OF THIS COLUMN (COLUMN-LEN)
*   THIS DEPENDS UPON THE DATA TYPE. MOST DATA TYPES HAVE
*   THE LENGTH SET, BUT VARCHAR, GRAPHIC, VARGRAPHIC, AND
*   DECIMAL DATA NEED TO HAVE THE BYTES CALCULATED.
*   THE NULL ATTRIBUTE MUST BE SEPARATED TO SIMPLIFY MATTERS.
*
***** MOVE SQLLEN(I) TO COLUMN-LEN.
*   COLUMN-IND IS 0 FOR NO NULLS AND 1 FOR NULLS
*   DIVIDE SQLTYPE(I) BY TWO GIVING DUMMY REMAINDER COLUMN-IND.
*   MYTYPE IS JUST THE SQLTYPE WITHOUT THE NULL BIT
*   MOVE SQLTYPE(I) TO MYTYPE.
*   SUBTRACT COLUMN-IND FROM MYTYPE.
*   SET THE COLUMN LENGTH, DEPENDENT UPON DATA TYPE
EVALUATE MYTYPE
WHEN CHARTYPE CONTINUE,
WHEN DATETYP CONTINUE,
WHEN TIMETYP CONTINUE,
WHEN TIMESTAMP CONTINUE,
WHEN FLOATTYPE CONTINUE,
WHEN VARCTYPE
  ADD TWO TO COLUMN-LEN,
WHEN VARTYPE
  ADD TWO TO COLUMN-LEN,
WHEN GTYPE
  MULTIPLY COLUMN-LEN BY TWO GIVING COLUMN-LEN,
WHEN VARGTYPE
  PERFORM CALC-VARG-LEN,
WHEN LVARGTYP
  PERFORM CALC-VARG-LEN,
WHEN HWTYP
  MOVE TWO TO COLUMN-LEN,
WHEN INTTYP
  MOVE FOUR TO COLUMN-LEN,
WHEN DECTYP
  PERFORM CALC-DECIMAL-LEN,
WHEN OTHER
  PERFORM UNRECOGNIZED-ERROR,
END-EVALUATE.
ADD COLUMN-LEN TO RECNUM.
ADD COLUMN-LEN TO REC1-LEN.
*****
*
*   IF THIS COLUMN CAN BE NULL, AN INDICATOR VARIABLE IS
*   NEEDED. WE ALSO RESERVE SPACE IN THE OUTPUT RECORD TO
*   NOTE THAT THE VALUE IS NULL.
*
***** MOVE ZERO TO IND(I).

```

```

        IF COLUMN-IND = ONE THEN
            SET SQLIND(I) TO ADDRESS OF IND(I)
            SET WORKINDPTR(I) TO RECPTR
            ADD ONE TO RECNUM
            ADD ONE TO REC1-LEN.
*
*           ADD ONE TO I.
*           PERFORMED PARAGRAPH TO CALCULATE COLUMN LENGTH
*           FOR A DECIMAL DATA TYPE COLUMN
CALC-DECIMAL-LEN.
            DIVIDE COLUMN-LEN BY 256 GIVING COLUMN-PREC
                REMAINDER COLUMN-SCALE.
            MOVE COLUMN-PREC TO COLUMN-LEN.
            ADD ONE TO COLUMN-LEN.
            DIVIDE COLUMN-LEN BY TWO GIVING COLUMN-LEN.
*
*           PERFORMED PARAGRAPH TO CALCULATE COLUMN LENGTH
*           FOR A VARGRAPHIC DATA TYPE COLUMN
CALC-VARG-LEN.
            MULTIPLY COLUMN-LEN BY TWO GIVING COLUMN-LEN.
            ADD TWO TO COLUMN-LEN.
*
*           PERFORMED PARAGRAPH TO NOTE AN UNRECOGNIZED
*           DATA TYPE COLUMN
UNRECOGNIZED-ERROR.
*
*           ERROR MESSAGE FOR UNRECOGNIZED DATA TYPE
*
MOVE SQLTYPE(I) TO TYPcod
MOVE 'Y' TO ERR-FOUND
WRITE MSGREC FROM BADTYPE
    AFTER ADVANCING 2 LINES
GO TO IND-RESULT.

*
*****SQL ERROR OCCURRED - GET MESSAGE *****
* DBERROR.
*
*           **SQL ERROR
MOVE 'Y' TO ERR-FOUND.
MOVE SQLCODE TO MSG-PRINT-CODE.
IF SQLCODE < 0 THEN MOVE '-' TO MSG-MINUS.
WRITE MSGREC FROM MSG-SQLERR
    AFTER ADVANCING 2 LINES.
CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
IF RETURN-CODE = ZERO
    PERFORM ERROR-PRINT VARYING ERROR-INDEX
        FROM 1 BY 1 UNTIL ERROR-INDEX GREATER THAN 8
ELSE
    *
    *           **ERROR FOUND IN DSNTIAR
    *           **PRINT ERROR MESSAGE
    MOVE RETURN-CODE TO RETCODE
    WRITE MSGREC FROM MSGRETC
        AFTER ADVANCING 2 LINES.
    GO TO IND-RESULT.

*
*****PRINT MESSAGE TEXT *****
* ERROR-PRINT.
WRITE MSGREC FROM ERROR-TEXT (ERROR-INDEX)
    AFTER ADVANCING 1 LINE.

```

Conceptos relacionados

[Programar directorios para Db2 12 \(Db2 for z/OS en IBM Documentation \)](#)

Ejemplo de programa COBOL con sentencias CONNECT

Este ejemplo muestra cómo acceder a datos distribuidos mediante el uso de sentencias CONNECT en un programa COBOL.

La siguiente figura contiene un programa COBOL de muestra que utiliza el compromiso de dos fases para acceder a datos distribuidos.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TWOPHASE.
AUTHOR.
REMARKS.
*****
```

```

* MODULE NAME = TWOPHASE
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION USING
*                   TWO PHASE COMMIT AND THE DRDA DISTRIBUTED
*                   ACCESS METHOD WITH CONNECT STATEMENTS
*
* COPYRIGHT = 5665-DB2 (C) COPYRIGHT IBM CORP 1982, 1989
* REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
*
* STATUS = VERSION 5
*
* FUNCTION = THIS MODULE DEMONSTRATES DISTRIBUTED DATA ACCESS
*             USING 2 PHASE COMMIT BY TRANSFERRING AN EMPLOYEE
*             FROM ONE LOCATION TO ANOTHER.
*
*           NOTE: THIS PROGRAM ASSUMES THE EXISTENCE OF THE
*                 TABLE SYSADM.EMP AT LOCATIONS STLEC1 AND
*                 STLEC2.
*
* MODULE TYPE = COBOL PROGRAM
*   PROCESSOR = DB2 PRECOMPILER, ENTERPRISE COBOL FOR Z/OS
*   MODULE SIZE = SEE LINK EDIT
*   ATTRIBUTES = NOT REENTRANT OR REUSABLE
*
* ENTRY POINT =
*   PURPOSE = TO ILLUSTRATE 2 PHASE COMMIT
*   LINKAGE = INVOKE FROM DSN RUN
*   INPUT = NONE
*   OUTPUT =
*             SYMBOLIC LABEL/NAME = SYSPRINT
*             DESCRIPTION = PRINT OUT THE DESCRIPTION OF EACH
*                           STEP AND THE RESULTANT SQLCA
*
* EXIT NORMAL = RETURN CODE 0 FROM NORMAL COMPLETION
*
* EXIT ERROR = NONE
*
* EXTERNAL REFERENCES =
*   ROUTINE SERVICES = NONE
*   DATA-AREAS = NONE
*   CONTROL-BLOCKS =
*             SQLCA - SQL COMMUNICATION AREA
*
* TABLES = NONE
*
* CHANGE-ACTIVITY = NONE
*
*

```

```

* PSEUDOCODE
*
* MAINLINE.
*   Perform CONNECT-TO-SITE-1 to establish
*     a connection to the local connection.
*   If the previous operation was successful Then
*     Do.
*       | Perform PROCESS-CURSOR-SITE-1 to obtain the
*         information about an employee that is
*         transferring to another location.
*       If the information about the employee was obtained
*         successfully Then
*         Do.
*           | Perform UPDATE-ADDRESS to update the information
*             to contain current information about the
*             employee.
*           Perform CONNECT-TO-SITE-2 to establish
*             a connection to the site where the employee is
*             transferring to.
*             If the connection is established successfully
*             Then
*               Do.
*                 | Perform PROCESS-SITE-2 to insert the
*                   employee information at the location
*                   where the employee is transferring to.
*                 End if the connection was established
*                   successfully.
*               End if the employee information was obtained
*                   successfully.
*             End if the previous operation was successful.
*             Perform COMMIT-WORK to COMMIT the changes made to STLEC1

```

```

*      and STLEC2. *
*
*      PROG-END.
*          Close the printer.
*          Return.
*
*      CONNECT-TO-SITE-1.
*          Provide a text description of the following step.
*          Establish a connection to the location where the
*              employee is transferring from.
*          Print the SQLCA out.
*
*      PROCESS-CURSOR-SITE-1.
*          Provide a text description of the following step.
*          Open a cursor that will be used to retrieve information
*              about the transferring employee from this site.
*          Print the SQLCA out.
*          If the cursor was opened successfully Then
*              Do.
*                  | Perform FETCH-DELETE-SITE-1 to retrieve and
*                      delete the information about the transferring
*                          employee from this site.
*                  | Perform CLOSE-CURSOR-SITE-1 to close the cursor.
*          End if the cursor was opened successfully.
*
*      FETCH-DELETE-SITE-1.
*          Provide a text description of the following step.
*          Fetch information about the transferring employee.
*          Print the SQLCA out.
*          If the information was retrieved successfully Then
*              Do.
*                  | Perform DELETE-SITE-1 to delete the employee
*                      at this site.
*          End if the information was retrieved successfully.
*
*      DELETE-SITE-1.
*          Provide a text description of the following step.
*          Delete the information about the transferring employee
*              from this site.
*          Print the SQLCA out.
*
*      CLOSE-CURSOR-SITE-1.
*          Provide a text description of the following step.
*          Close the cursor used to retrieve information about
*              the transferring employee.
*          Print the SQLCA out.
*
*      UPDATE-ADDRESS.
*          Update the address of the employee.
*          Update the city of the employee.
*          Update the location of the employee.
*
*      CONNECT-TO-SITE-2.
*          Provide a text description of the following step.
*          Establish a connection to the location where the
*              employee is transferring to.
*          Print the SQLCA out.
*
*      PROCESS-SITE-2.
*          Provide a text description of the following step.
*          Insert the employee information at the location where
*              the employee is being transferred to.
*          Print the SQLCA out.
*
*      COMMIT-WORK.
*          COMMIT all the changes made to STLEC1 and STLEC2.
*
*****
```

```

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT PRINTER, ASSIGN TO S-OUT1.

DATA DIVISION.
FILE SECTION.
FD  PRINTER
    RECORD CONTAINS 120 CHARACTERS
    DATA RECORD IS PRT-TC-RESULTS
```

```

      LABEL RECORD IS OMITTED.
01  PRT-TC-RESULTS.
03  PRT-BLANK          PIC X(120).

```

WORKING-STORAGE SECTION.

```
*****
* Variable declarations
*****
*****
```

```

01  H-EMPTBL.
05  H-EMPNO    PIC X(6).
05  H-NAME.
   49 H-NAME-LN  PIC S9(4) COMP-5.
   49 H-NAME-DA  PIC X(32).
05  H-ADDRESS.
   49 H-ADDRESS-LN PIC S9(4) COMP-5.
   49 H-ADDRESS-DA PIC X(36).
05  H-CITY.
   49 H-CITY-LN  PIC S9(4) COMP-5.
   49 H-CITY-DA  PIC X(36).
05  H-EMPLLOC  PIC X(4).
05  H-SSNO     PIC X(11).
05  H-BORN     PIC X(10).
05  H-SEX      PIC X(1).
05  H-HIRED    PIC X(10).
05  H-DEPTNO   PIC X(3).
05  H-JOBCODE  PIC S9(3)V COMP-3.
05  H-SRATE    PIC S9(5) COMP.
05  H-EDUC     PIC S9(5) COMP.
05  H-SAL      PIC S9(6)V9(2) COMP-3.
05  H-VALIDCHK PIC S9(6)V COMP-3.

```

```

01  H-EMPTBL-IND-TABLE.
02  H-EMPTBL-IND      PIC S9(4) COMP-5 OCCURS 15 TIMES.

```

```
*****
* Includes for the variables used in the COBOL standard
* language procedures and the SQLCA.
*****
*****
```

```

EXEC SQL INCLUDE COBSVAR END-EXEC.
EXEC SQL INCLUDE SQLCA END-EXEC.

```

```
*****
* Declaration for the table that contains employee information
*****
*****
```

```

EXEC SQL DECLARE SYSADM.EMP TABLE
  (EMPNO  CHAR(6) NOT NULL,
   NAME    VARCHAR(32),
   ADDRESS VARCHAR(36),
   CITY    VARCHAR(36),
   EMPLOC  CHAR(4) NOT NULL,
   SSNO    CHAR(11),
   BORN    DATE,
   SEX     CHAR(1),
   HIRED   CHAR(10),
   DEPTNO  CHAR(3) NOT NULL,
   JOBCODE DECIMAL(3),
   SRATE   SMALLINT,
   EDUC    SMALLINT,
   SAL     DECIMAL(8,2) NOT NULL,
   VALCHK  DECIMAL(6))
END-EXEC.

```

```
*****
* Constants
*****
*****
```

```

77  SITE-1           PIC X(16) VALUE 'STLEC1'.
77  SITE-2           PIC X(16) VALUE 'STLEC2'.
77  TEMP-EMPNO       PIC X(6)  VALUE '080000'.
77  TEMP-ADDRESS-LN  PIC 99   VALUE 15.
77  TEMP-CITY-LN    PIC 99   VALUE 18.

```

```
*****
* Declaration of the cursor that will be used to retrieve
*****
*****
```

```

* information about a transferring employee *
*****



EXEC SQL DECLARE C1 CURSOR FOR
      SELECT EMPNO, NAME, ADDRESS, CITY, EMPLOC,
             SSNO, BORN, SEX, HIRED, DEPTNO, JOBCODE,
             SRATE, EDUC, SAL, VALCHK
        FROM  SYSADM.EMP
       WHERE EMPNO = :TEMP-EMPNO
END-EXEC.

PROCEDURE DIVISION.
A101-HOUSE-KEEPING.
OPEN OUTPUT PRINTER.

***** 
* An employee is transferring from location STLEC1 to STLEC2.   *
* Retrieve information about the employee from STLEC1, delete   *
* the employee from STLEC1 and insert the employee at STLEC2   *
* using the information obtained from STLEC1.                 *
*****



MAINLINE.
  PERFORM CONNECT-TO-SITE-1
  IF SQLCODE IS EQUAL TO 0
    PERFORM PROCESS-CURSOR-SITE-1
    IF SQLCODE IS EQUAL TO 0
      PERFORM UPDATE-ADDRESS
      PERFORM CONNECT-TO-SITE-2
      IF SQLCODE IS EQUAL TO 0
        PERFORM PROCESS-SITE-2.
  PERFORM COMMIT-WORK.

PROG-END.
CLOSE PRINTER.
GOBACK.

***** 
* Establish a connection to STLEC1                               *
*****



CONNECT-TO-SITE-1.

MOVE 'CONNECT TO STLEC1' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
  CONNECT TO :SITE-1
END-EXEC.
PERFORM PTSQLCA.

***** 
* When a connection has been established successfully at STLEC1,* 
* open the cursor that will be used to retrieve information   *
* about the transferring employee.                            *
*****



PROCESS-CURSOR-SITE-1.

MOVE 'OPEN CURSOR C1' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
  OPEN C1
END-EXEC.
PERFORM PTSQLCA.
IF SQLCODE IS EQUAL TO ZERO
  PERFORM FETCH-DELETE-SITE-1
  PERFORM CLOSE-CURSOR-SITE-1.

***** 
* Retrieve information about the transferring employee.        *
* Provided that the employee exists, perform DELETE-SITE-1 to *
* delete the employee from STLEC1.                           *
*****



FETCH-DELETE-SITE-1.

MOVE 'FETCH C1' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
  FETCH C1 INTO :H-EMPTBL:H-EMPTBL-IND
END-EXEC.

```

```

PERFORM PTSQLCA.
IF SQLCODE IS EQUAL TO ZERO
    PERFORM DELETE-SITE-1.

*****
* Delete the employee from STLEC1. *
*****


DELETE-SITE-1.

MOVE 'DELETE EMPLOYEE ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
MOVE 'DELETE EMPLOYEE      ' TO STNAME
EXEC SQL
    DELETE FROM SYSADM.EMP
        WHERE EMPNO = :TEMP-EMPNO
END-EXEC.
PERFORM PTSQLCA.

*****
* Close the cursor used to retrieve information about the      *
* transferring employee.                                     *
*****


CLOSE-CURSOR-SITE-1.

MOVE 'CLOSE CURSOR C1      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    CLOSE C1
END-EXEC.
PERFORM PTSQLCA.

*****
* Update certain employee information in order to make it      *
* current.                                                 *
*****


UPDATE-ADDRESS.

MOVE TEMP-ADDRESS-LN      TO H-ADDRESS-LN.
MOVE '1500 NEW STREET'    TO H-ADDRESS-DA.
MOVE TEMP-CITY-LN         TO H-CITY-LN.
MOVE 'NEW CITY, CA 97804' TO H-CITY-DA.
MOVE 'SJCA'               TO H-EMPLOC.

*****
* Establish a connection to STLEC2                           *
*****


CONNECT-TO-SITE-2.

MOVE 'CONNECT TO STLEC2      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    CONNECT TO :SITE-2
END-EXEC.
PERFORM PTSQLCA.

*****
* Using the employee information that was retrieved from STLEC1 *
* and updated previously, insert the employee at STLEC2.          *
*****


PROCESS-SITE-2.

MOVE 'INSERT EMPLOYEE      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    INSERT INTO SYSADM.EMP VALUES
    (:H-EMPNO,
     :H-NAME,
     :H-ADDRESS,
     :H-CITY,
     :H-EMPLOC,
     :H-SSNO,
     :H-BORN,
     :H-SEX,
     :H-HIRED,

```

```

:H-DEPTNO,
:H-JOBCODE,
:H-SRATE,
:H-EDUC,
:H-SAL,
:H-VALIDCHK)
END-EXEC.
PERFORM PTSQLCA.

*****
* COMMIT any changes that were made at STLEC1 and STLEC2. *
*****

COMMIT-WORK.

MOVE 'COMMIT WORK           ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    COMMIT
END-EXEC.
PERFORM PTSQLCA.

*****
* Include COBOL standard language procedures      *
*****
```

INCLUDE-SUBS.
EXEC SQL INCLUDE COBSSUB END-EXEC.

Ejemplo de programa COBOL que utiliza alias para nombres de tres partes

Puede acceder a los datos distribuidos utilizando alias para nombres de tres partes en un programa COBOL.

El siguiente programa de ejemplo muestra datos de acceso distribuido utilizando alias para nombres de tres partes con confirmación en dos fases.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TWOPHASE.
AUTHOR.
REMARKS.
*****
*
* MODULE NAME = TWOPHASE
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION USING
*                   TWO PHASE COMMIT AND DRDA WITH
*                   ALIASES FOR THREE-PART NAMES
*
* FUNCTION = THIS MODULE DEMONSTRATES DISTRIBUTED DATA ACCESS
*             USING 2 PHASE COMMIT BY TRANSFERRING AN EMPLOYEE
*             FROM ONE LOCATION TO ANOTHER.
*
* NOTE: THIS PROGRAM ASSUMES THE EXISTENCE OF THE
*       TABLE SYSADM.ALLEMPLOYEES AT LOCATIONS STLEC1
*       AND STLEC2.
*
* MODULE TYPE = COBOL PROGRAM
*   PROCESSOR = DB2 PRECOMPILER, ENTERPRISE COBOL FOR Z/OS
*   MODULE SIZE = SEE LINK EDIT
*   ATTRIBUTES = NOT REENTRANT OR REUSABLE
*
* ENTRY POINT =
*   PURPOSE = TO ILLUSTRATE 2 PHASE COMMIT
*   LINKAGE = INVOKE FROM DSN RUN
*   INPUT = NONE
*   OUTPUT =
*           SYMBOLIC LABEL/NAME = SYSPRINT
*           DESCRIPTION = PRINT OUT THE DESCRIPTION OF EACH
*                         STEP AND THE RESULTANT SQLCA
*
* EXIT NORMAL = RETURN CODE 0 FROM NORMAL COMPLETION
*
* EXIT ERROR = NONE
*
* EXTERNAL REFERENCES =
*   ROUTINE SERVICES = NONE
*   DATA-AREAS = NONE
```

```

*      CONTROL-BLOCKS      =
*          SQLCA - SQL COMMUNICATION AREA
*
*      TABLES = NONE
*
*      CHANGE-ACTIVITY = NONE
*
*
*
*      PSEUDOCODE
*
*      MAINLINE.
*          Perform PROCESS-CURSOR-SITE-1 to obtain the information
*              about an employee that is transferring to another
*                  location.
*          If the information about the employee was obtained
*              successfully Then
*              Do.
*                  | Perform UPDATE-ADDRESS to update the information to
*                      contain current information about the employee.
*                  | Perform PROCESS-SITE-2 to insert the employee
*                      information at the location where the employee is
*                          transferring to.
*              End if the employee information was obtained
*                  successfully.
*          Perform COMMIT-WORK to COMMIT the changes made to STLEC1
*              and STLEC2.
*
*      PROG-END.
*          Close the printer.
*          Return.
*
*      PROCESS-CURSOR-SITE-1.
*          Provide a text description of the following step.
*          Open a cursor that will be used to retrieve information
*              about the transferring employee from this site.
*          Print the SQLCA out.
*          If the cursor was opened successfully Then
*              Do.
*                  | Perform FETCH-DELETE-SITE-1 to retrieve and
*                      delete the information about the transferring
*                          employee from this site.
*                  | Perform CLOSE-CURSOR-SITE-1 to close the cursor.
*              End if the cursor was opened successfully.
*
*      FETCH-DELETE-SITE-1.
*          Provide a text description of the following step.
*          Fetch information about the transferring employee.
*          Print the SQLCA out.
*          If the information was retrieved successfully Then
*              Do.
*                  | Perform DELETE-SITE-1 to delete the employee
*                      at this site.
*              End if the information was retrieved successfully.
*
*      DELETE-SITE-1.
*          Provide a text description of the following step.
*          Delete the information about the transferring employee
*              from this site.
*          Print the SQLCA out.
*
*      CLOSE-CURSOR-SITE-1.
*          Provide a text description of the following step.
*          Close the cursor used to retrieve information about
*              the transferring employee.
*          Print the SQLCA out.
*
*      UPDATE-ADDRESS.
*          Update the address of the employee.
*          Update the city of the employee.
*          Update the location of the employee.
*
*      PROCESS-SITE-2.
*          Provide a text description of the following step.
*          Insert the employee information at the location where
*              the employee is being transferred to.
*          Print the SQLCA out.
*
*      COMMIT-WORK.
*          COMMIT all the changes made to STLEC1 and STLEC2.
*
*****

```

```

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT PRINTER, ASSIGN TO S-OUT1.

DATA DIVISION.
FILE SECTION.
FD PRINTER
  RECORD CONTAINS 120 CHARACTERS
  DATA RECORD IS PRT-TC-RESULTS
  LABEL RECORD IS OMITTED.
01 PRT-TC-RESULTS.
  03 PRT-BLANK          PIC X(120).

WORKING-STORAGE SECTION.

*****
* Variable declarations
*****
01 H-EMPTBL.
  05 H-EMPNO    PIC X(6).
  05 H-NAME.
    49 H-NAME-LN  PIC S9(4) COMP-5.
    49 H-NAME-DA  PIC X(32).
  05 H-ADDRESS.
    49 H-ADDRESS-LN PIC S9(4) COMP-5.
    49 H-ADDRESS-DA PIC X(36).
  05 H-CITY.
    49 H-CITY-LN   PIC S9(4) COMP-5.
    49 H-CITY-DA   PIC X(36).
  05 H-EMPLLOC  PIC X(4).
  05 H-SSNO     PIC X(11).
  05 H-BORN      PIC X(10).
  05 H-SEX       PIC X(1).
  05 H-HIRED     PIC X(10).
  05 H-DEPTNO   PIC X(3).
  05 H-JOBCODE  PIC S9(3)V COMP-3.
  05 H-SRATE    PIC S9(5) COMP.
  05 H-EDUC     PIC S9(5) COMP.
  05 H-SAL      PIC S9(6)V9(2) COMP-3.
  05 H-VALIDCHK PIC S9(6)V COMP-3.
01 H-EMPTBL-IND-TABLE.
  02 H-EMPTBL-IND    PIC S9(4) COMP-5 OCCURS 15 TIMES.

*****
* Includes for the variables used in the COBOL standard
* language procedures and the SQLCA.
*****
EXEC SQL INCLUDE COBSVAR END-EXEC.
EXEC SQL INCLUDE SQLCA END-EXEC.

*****
* Declaration for the table that contains employee information
*****
EXEC SQL DECLARE SYSADM.ALLEMPLOYEES TABLE
  (EMPNO  CHAR(6) NOT NULL,
   NAME    VARCHAR(32),
   ADDRESS VARCHAR(36),
   CITY    VARCHAR(36),
   EMPLLOC CHAR(4) NOT NULL,
   SSNO    CHAR(11),
   BORN    DATE,
   SEX     CHAR(1),
   HIRED   CHAR(10),
   DEPTNO  CHAR(3) NOT NULL,
   JOBCODE DECIMAL(3),
   SRATE   SMALLINT,
   EDUC    SMALLINT,
   SAL     DECIMAL(8,2) NOT NULL,
   VALCHK  DECIMAL(6))
END-EXEC.

*****
* Constants
*****
77 TEMP-EMPNO          PIC X(6) VALUE '080000'.

```

```

77 TEMP-ADDRESS-LN          PIC 99      VALUE 15.
77 TEMP-CITY-LN            PIC 99      VALUE 18.

*****
* Declaration of the cursor that will be used to retrieve      *
* information about a transferring employee                      *
* EC1EMP is the alias for STLEC1.SYSADM.ALLEMPLOYEES           *
*****


EXEC SQL DECLARE C1 CURSOR FOR
    SELECT EMPNO, NAME, ADDRESS, CITY, EMPLOC,
           SSNO, BORN, SEX, HIRED, DEPTNO, JOBCODE,
           SRATE, EDUC, SAL, VALCHK
      FROM EC1EMP
     WHERE EMPNO = :TEMP-EMPNO
END-EXEC.

PROCEDURE DIVISION.
A101-HOUSE-KEEPING.
    OPEN OUTPUT PRINTER.

***** 
* An employee is transferring from location STLEC1 to STLEC2.   *
* Retrieve information about the employee from STLEC1, delete   *
* the employee from STLEC1 and insert the employee at STLEC2   *
* using the information obtained from STLEC1.                  *
*****


MAINLINE.
    PERFORM PROCESS-CURSOR-SITE-1
    IF SQLCODE IS EQUAL TO 0
        PERFORM UPDATE-ADDRESS
        PERFORM PROCESS-SITE-2.
    PERFORM COMMIT-WORK.

PROG-END.
    CLOSE PRINTER.
    GOBACK.

***** 
* Open the cursor that will be used to retrieve information   *
* about the transferring employee.                            *
*****


PROCESS-CURSOR-SITE-1.

MOVE 'OPEN CURSOR C1      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    OPEN C1
END-EXEC.
PERFORM PTSQLCA.
IF SQLCODE IS EQUAL TO ZERO
    PERFORM FETCH-DELETE-SITE-1
    PERFORM CLOSE-CURSOR-SITE-1.

***** 
* Retrieve information about the transferring employee.       *
* Provided that the employee exists, perform DELETE-SITE-1 to *
* delete the employee from STLEC1.                          *
*****


FETCH-DELETE-SITE-1.

MOVE 'FETCH C1      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    FETCH C1 INTO :H-EMPTBL:H-EMPTBL-IND
END-EXEC.          PERFORM PTSQLCA.
IF SQLCODE IS EQUAL TO ZERO
    PERFORM DELETE-SITE-1.

***** 
* Delete the employee from STLEC1.                         *
*****


DELETE-SITE-1.

MOVE 'DELETE EMPLOYEE ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
MOVE 'DELETE EMPLOYEE      ' TO STNAME
EXEC SQL
    DELETE FROM EC1EMP

```

```

        WHERE EMPNO = :TEMP-EMPNO
END-EXEC.
PERFORM PTSQLCA.

*****
* Close the cursor used to retrieve information about the      *
* transferring employee.                                     *
*****
CLOSE-CURSOR-SITE-1.

MOVE 'CLOSE CURSOR C1      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    CLOSE C1
END-EXEC.
PERFORM PTSQLCA.

*****
* Update certain employee information in order to make it      *
* current.                                                 *
*****
UPDATE-ADDRESS.
MOVE TEMP-ADDRESS-LN      TO H-ADDRESS-LN.
MOVE '1500 NEW STREET'    TO H-ADDRESS-DA.
MOVE TEMP-CITY-LN         TO H-CITY-LN.
MOVE 'NEW CITY, CA 97804' TO H-CITY-DA.
MOVE 'SJCA'               TO H-EMPLOC.
*****
* Using the employee information that was retrieved from STLEC1 *
* and updated previously, insert the employee at STLEC2.          *
* EC2EMP is the alias for STLEC2.SYSADM.ALLEMPLOYEES            *
*****
PROCESS-SITE-2.

MOVE 'INSERT EMPLOYEE      ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    INSERT INTO EC2EMP VALUES
    (:H-EMPNO,
     :H-NAME,
     :H-ADDRESS,
     :H-CITY,
     :H-EMPLOC,
     :H-SSNO,
     :H-BORN,
     :H-SEX,
     :H-HIRED,
     :H-DEPTNO,
     :H-JOBCODE,
     :H-SRATE,
     :H-EDUC,
     :H-SAL,
     :H-VALIDCHK)
END-EXEC.
PERFORM PTSQLCA.

*****
* COMMIT any changes that were made at STLEC1 and STLEC2.      *
*****
COMMIT-WORK.

MOVE 'COMMIT WORK          ' TO STNAME
WRITE PRT-TC-RESULTS FROM STNAME
EXEC SQL
    COMMIT
END-EXEC.
PERFORM PTSQLCA.

*****
* Include COBOL standard language procedures                  *
*****
INCLUDE-SUBS.
EXEC SQL INCLUDE COBSSUB END-EXEC.

```

Ejemplo de procedimiento almacenado COBOL con una convención de vinculación GENERAL WITH NULLS

Puede llamar a un procedimiento almacenado que utilice la convención de vinculación GENERAL WITH NULLS desde un programa COBOL.

Este ejemplo de procedimiento almacenado hace lo siguiente:

- Busca en la tabla de catálogo Db2 SYSIBM.SYSROUTINES una fila que coincida con los parámetros de entrada del programa cliente. Los dos parámetros de entrada contienen valores para NAME y SCHEMA.
- Busca en la tabla SYSTABLES del catálogo Db2 todas las tablas en las que el valor de CREATOR coincide con el valor del parámetro de entrada SCHEMA. El procedimiento almacenado utiliza un cursor para devolver los nombres de las tablas.

La convención de vinculación para este procedimiento almacenado es GENERAL WITH NULLS.

Los parámetros de salida de este procedimiento almacenado contienen el SQLCODE de la operación SELECT y el valor de la columna RUNOPTS recuperado de la tabla SYSIBM.SYSROUTINES.

La declaración CREATE PROCEDURE para este procedimiento almacenado podría tener este aspecto:

```
CREATE PROCEDURE GETPRML(Procnm CHAR(18) IN, Schema CHAR(8) IN,
    Outcode INTEGER OUT, Parmlst VARCHAR(254) OUT)
LANGUAGE COBOL
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME "GETPRML"
COLLID GETPRML
ASUTIME NO LIMIT
PARAMETER STYLE GENERAL WITH NULLS
STAY RESIDENT NO
RUN OPTIONS "MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)"
WLM ENVIRONMENT SAMPPROG
PROGRAM TYPE MAIN
SECURITY DB2
RESULT SETS 2
COMMIT ON RETURN NO;
```

El siguiente ejemplo es un procedimiento almacenado COBOL con la convención de vinculación GENERAL WITH NULLS.

```
CBL RENT
IDENTIFICATION DIVISION.
PROGRAM-ID. GETPRML.
AUTHOR. EXAMPLE.
DATE-WRITTEN. 03/25/98.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
DATA DIVISION.
FILE SECTION.
*
* WORKING-STORAGE SECTION.
*
* EXEC SQL INCLUDE SQLCA END-EXEC.
*
*****{*}
* DECLARE A HOST VARIABLE TO HOLD INPUT SCHEMA
*****{*}
01 INSCHEMA PIC X(8).
*****{*}
* DECLARE CURSOR FOR RETURNING RESULT SETS
*****{*}
*
* EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
*      SELECT NAME FROM SYSIBM.SYSTABLES WHERE CREATOR=:INSCHEMA
*      END-EXEC.
*
LINKAGE SECTION.
*****{*}
* DECLARE THE INPUT PARAMETERS FOR THE PROCEDURE
*****{*}
01 PROCNM PIC X(18).
```

```

01 SCHEMA PIC X(8).
*****
* DECLARE THE OUTPUT PARAMETERS FOR THE PROCEDURE
*****
01 OUT-CODE PIC S9(9) USAGE BINARY.
01 PARMST.
 49 PARMLST-LEN PIC S9(4) USAGE BINARY.
 49 PARMLST-TEXT PIC X(254).
*****
* DECLARE THE STRUCTURE CONTAINING THE NULL
* INDICATORS FOR THE INPUT AND OUTPUT PARAMETERS.
*****
01 IND-PARM.
 03 PROCNM-IND PIC S9(4) USAGE BINARY.
 03 SCHEMA-IND PIC S9(4) USAGE BINARY.
 03 OUT-CODE-IND PIC S9(4) USAGE BINARY.
 03 PARMLST-IND PIC S9(4) USAGE BINARY.

PROCEDURE DIVISION USING PROCNM, SCHEMA,
OUT-CODE, PARMLST, IND-PARM.
*****
* If any input parameter is null, return a null value
* for PARMLST and set the output return code to 9999.
*****
IF PROCNM-IND < 0 OR
SCHEMA-IND < 0
MOVE 9999 TO OUT-CODE
MOVE 0 TO OUT-CODE-IND
MOVE -1 TO PARMLST-IND
ELSE
*****
* Issue the SQL SELECT against the SYSIBM.SYSROUTINES
* DB2 catalog table.
*****
EXEC SQL
  SELECT RUNOPTS INTO :PARMLST
  FROM SYSIBM.SYSROUTINES
  WHERE NAME=:PROCNM AND
SCHEMA=:SCHEMA
END-EXEC
MOVE 0 TO PARMLST-IND
*****
* COPY SQLCODE INTO THE OUTPUT PARAMETER AREA
*****
MOVE SQLCODE TO OUT-CODE
MOVE 0 TO OUT-CODE-IND.
*
*****
* OPEN CURSOR C1 TO CAUSE DB2 TO RETURN A RESULT SET
* TO THE CALLER.
*****
EXEC SQL OPEN C1
END-EXEC.
PROG-END.
GOBACK.

```

Ejemplo de procedimiento almacenado COBOL con una convención de vinculación GENERAL

Puede llamar a un procedimiento almacenado que utilice la convención de vinculación GENERAL desde un programa COBOL.

Este ejemplo de procedimiento almacenado hace lo siguiente:

- Busca en la tabla de catálogo SYSROUTINES una fila que coincida con los parámetros de entrada del programa cliente. Los dos parámetros de entrada contienen valores para NAME y SCHEMA.
- Busca en la tabla SYSTABLES del catálogo Db2 todas las tablas en las que el valor de CREATOR coincide con el valor del parámetro de entrada SCHEMA. El procedimiento almacenado utiliza un cursor para devolver los nombres de las tablas.

Este procedimiento almacenado puede devolver un valor NULL para las variables de host de salida.

La convención de vinculación para este procedimiento almacenado es GENERAL.

Los parámetros de salida de este procedimiento almacenado contienen el SQLCODE de la operación SELECT y el valor de la columna RUNOPTS recuperado de la tabla SYSROUTINES.

La declaración CREATE PROCEDURE para este procedimiento almacenado podría tener este aspecto:

```
CREATE PROCEDURE GETPRML(Procnm CHAR(18) IN, Schema CHAR(8) IN,
    OutCode INTEGER OUT, Parmlst VARCHAR(254) OUT)
LANGUAGE COBOL
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME "GETPRML"
COLLID GETPRML
ASUTIME NO LIMIT
PARAMETER STYLE GENERAL
STAY RESIDENT NO
RUN OPTIONS "MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)"
WLM ENVIRONMENT SAMPPROG
PROGRAM TYPE MAIN
SECURITY DB2
RESULT SETS 2
COMMIT ON RETURN NO;
```

```
CBL RENT
IDENTIFICATION DIVISION.
PROGRAM-ID. GETPRML.
AUTHOR. EXAMPLE.
DATE-WRITTEN. 03/25/98.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
DATA DIVISION.
FILE SECTION.

WORKING-STORAGE SECTION.

    EXEC SQL INCLUDE SQLCA END-EXEC.
*****
*  DECLARE A HOST VARIABLE TO HOLD INPUT SCHEMA
*****
01  INSCHEMA PIC X(8).

*****
*  DECLARE CURSOR FOR RETURNING RESULT SETS
*****
*
    EXEC SQL DECLARE C1 CURSOR WITH RETURN FOR
        SELECT NAME FROM SYSIBM.SYSTABLES WHERE CREATOR=:INSCHEMA
    END-EXEC.

*
LINKAGE SECTION.
*****
*  DECLARE THE INPUT PARAMETERS FOR THE PROCEDURE
*****
01  Procnm  PIC X(18).
01  Schema  PIC X(8).
*****
*  DECLARE THE OUTPUT PARAMETERS FOR THE PROCEDURE
*****
01  Out-Code PIC S9(9) USAGE BINARY.
01  Parmlst.
    49 Parmlst-LEN  PIC S9(4) USAGE BINARY.
    49 Parmlst-Text PIC X(254).

PROCEDURE DIVISION USING Procnm, Schema,
    Out-Code, Parmlst.

*****
* Issue the SQL SELECT against the SYSIBM.SYSROUTINES
* DB2 catalog table.
*****
EXEC SQL
    SELECT Runopts INTO :Parmlst
        FROM SYSIBM.ROUTINES
        WHERE Name=:Procnm AND
            Schema=:Schema
    END-EXEC.
```

```
*****
* COPY SQLCODE INTO THE OUTPUT PARAMETER AREA
*****
MOVE SQLCODE TO OUT-CODE.
*****
* OPEN CURSOR C1 TO CAUSE DB2 TO RETURN A RESULT SET
* TO THE CALLER.
*****
EXEC SQL OPEN C1
END-EXEC.
PROG-END.
GOBACK.
```

Ejemplo de programa COBOL que llama a un procedimiento almacenado

Puede llamar al procedimiento almacenado GETPRML que utiliza la convención de vinculación GENERAL WITH NULLS desde un programa COBOL en un sistema z/OS .

Dado que el procedimiento almacenado devuelve conjuntos de resultados, este programa comprueba si hay conjuntos de resultados y recupera el contenido de los mismos. La siguiente figura contiene el ejemplo de programa COBOL que llama al procedimiento almacenado GETPRML.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CALPRML.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT REPOUT
         ASSIGN TO UT-S-SYSPRINT.

DATA DIVISION.
FILE SECTION.
FD REPOUT
   RECORD CONTAINS 127 CHARACTERS
   LABEL RECORDS ARE OMITTED
   DATA RECORD IS REPREC.
01 REPREC          PIC X(127).

WORKING-STORAGE SECTION.
*****
* MESSAGES FOR SQL CALL
*****
01 SQLREC.
   02 BADMSG    PIC X(34) VALUE
      ' SQL CALL FAILED DUE TO SQLCODE = '.
   02 BADCODE    PIC +9(5) USAGE DISPLAY.
   02 FILLER     PIC X(80) VALUE SPACES.
01 ERMREC.
   02 ERRMSG    PIC X(12) VALUE ' SQLERRMC = '.
   02 ERRCODE    PIC X(70).
   02 FILLER     PIC X(38) VALUE SPACES.
01 CALLREC.
   02 CALLMSG   PIC X(28) VALUE
      ' GETPRML FAILED DUE TO RC = '.
   02 CALLCODE   PIC +9(5) USAGE DISPLAY.
   02 FILLER     PIC X(42) VALUE SPACES.
01 RSLTREC.
   02 RSLTMSG   PIC X(15) VALUE
      ' TABLE NAME IS '.
   02 TBLNAME   PIC X(18) VALUE SPACES.
   02 FILLER     PIC X(87) VALUE SPACES.

*****
* WORK AREAS
*****
01 PROCNM          PIC X(18).
01 SCHEMA          PIC X(8).
01 OUT-CODE        PIC S9(9) USAGE COMP-5.
01 PARMST.
   49 PARMLEN       PIC S9(4) USAGE COMP-5.
   49 PARMTXT      PIC X(254).
01 PARMBUF REDEFINES PARMST.
   49 PARBLEN       PIC S9(4) USAGE COMP-5.
   49 PARMARRY      PIC X(127) OCCURS 2 TIMES.
01 NAME.
```

```

49 NAMELEN      PIC S9(4) USAGE COMP-5.
49 NAMETXT     PIC X(18).
77 PARMIND      PIC S9(4) COMP-5.
77 I            PIC S9(4) COMP-5.
77 NUMLINES     PIC S9(4) COMP-5.
*****
* DECLARE A RESULT SET LOCATOR FOR THE RESULT SET   *
* THAT IS RETURNED.                                *
*****                                                 *
01 LOC          USAGE SQL TYPE IS
                RESULT-SET-LOCATOR VARYING.

*****
* SQL INCLUDE FOR SQLCA
* EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
*-----*
PROG-START.
    OPEN OUTPUT REPOUT.
*        OPEN OUTPUT FILE
*        MOVE 'DSN8EP2'      TO PROCNM.
*        INPUT PARAMETER -- PROCEDURE TO BE FOUND
*        MOVE SPACES TO SCHEMA.
*        INPUT PARAMETER -- SCHEMA IN SYSROUTINES
*        MOVE -1 TO PARMIND.
*        THE PARMLST PARAMETER IS AN OUTPUT PARM.
*        MARK PARMLST PARAMETER AS NULL, SO THE DB2
*        REQUESTER DOES NOT HAVE TO SEND THE ENTIRE
*        PARMLST VARIABLE TO THE SERVER. THIS
*        HELPS REDUCE NETWORK I/O TIME, BECAUSE
*        PARMLST IS FAIRLY LARGE.
* EXEC SQL
*     CALL GETPRML (:PROCNM,
*                  :SCHEMA,
*                  :OUT-CODE,
*                  :PARMLST INDICATOR :PARMIND)
* END-EXEC.

* MAKE THE CALL
* IF SQLCODE NOT EQUAL TO +466 THEN
*     IF CALL RETURNED BAD SQLCODE
*         MOVE SQLCODE TO BADCODE
*         WRITE REPREC FROM SQLREC
*         MOVE SQLERRMC TO ERRMCODE
*         WRITE REPREC FROM ERRMREC
*     ELSE
*         PERFORM GET-PARMS
*         PERFORM GET-RESULT-SET.
* PROG-END.
*     CLOSE REPOUT.
*     CLOSE OUTPUT FILE
*     GOBACK.
* PARMVRT.
*     MOVE SPACES TO REPREC.
*     WRITE REPREC FROM PARMARRY(I)
*           AFTER ADVANCING 1 LINE.
* GET-PARMS.
*     IF THE CALL WORKED,
*     IF OUT-CODE NOT EQUAL TO 0 THEN
*         DID GETPRML HIT AN ERROR?
*         MOVE OUT-CODE TO CALLCODE
*         WRITE REPREC FROM CALLREC
*     ELSE
*         EVERYTHING WORKED
*         DIVIDE 127 INTO PARMLEN GIVING NUMLINES ROUNDED
*         FIND OUT HOW MANY LINES TO PRINT
*         PERFORM PARMVRT VARYING I
*             FROM 1 BY 1 UNTIL I GREATER THAN NUMLINES.
* GET-RESULT-SET.
*****
* ASSUME YOU KNOW THAT ONE RESULT SET IS RETURNED,  *
* AND YOU KNOW THE FORMAT OF THAT RESULT SET.       *
* ALLOCATE A CURSOR FOR THE RESULT SET, AND FETCH   *
* THE CONTENTS OF THE RESULT SET.                   *
*****                                                 *
* EXEC SQL ASSOCIATE LOCATORS (:LOC)
*           WITH PROCEDURE GETPRML
* END-EXEC.
* LINK THE RESULT SET TO THE LOCATOR

```

```

EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :LOC
END-EXEC.
*
      LINK THE CURSOR TO THE RESULT SET
      PERFORM GET-ROWS VARYING I
      FROM 1 BY 1 UNTIL SQLCODE EQUAL TO +100.
GET-ROWS.
      EXEC SQL FETCH C1 INTO :NAME
      END-EXEC.
      MOVE NAME TO TBLNAME.
      WRITE REPREC FROM RSLTREC
      AFTER ADVANCING 1 LINE.

```

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en COBOL

Los programas en COBOL que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Acerca de esta tarea

Si especifica la opción de procesamiento SQL STDSQL(YES), no defina un SQLCA. Si lo hace, Db2 ignora su SQLCA y la definición de su SQLCA provoca errores en tiempo de compilación. Si especifica la opción de procesamiento SQL STDSQL(NO), incluya un SQLCA explícitamente.

Para los programas COBOL, cuando especifique STDSQL(YES), debe declarar una variable SQLCODE. Db2 declara un área SQLCA para usted en la SECCIÓN WORKING-STORAGE. Db2 controla la estructura y la ubicación de la SQLCA.

Si su aplicación contiene instrucciones SQL y no incluye un área de comunicaciones SQL (SQLCA), debe declarar variables de host SQLCODE y SQLSTATE individuales. Su programa puede utilizar estas variables para comprobar si una instrucción SQL se ha ejecutado correctamente.

Procedimiento

Elija una de estas acciones:

Opción	Descripción
Para definir el área de comunicaciones SQL:	<p>a. Codifique el SQLCA directamente en el programa o utilice la siguiente instrucción SQL INCLUDE para solicitar una declaración SQLCA estándar:</p> <pre>EXEC SQL INCLUDE SQLCA</pre> <p>Puede especificar INCLUDE SQLCA o una declaración para SQLCODE dondequiero que pueda especificar un nivel 77 o una entrada de descripción de registro en la SECCIÓN DE ALMACENAMIENTO DE TRABAJO.</p> <p>Db2 establece los valores SQLCODE y SQLSTATE en el SQLCA después de que se ejecute cada instrucción SQL. Su aplicación debe comprobar estos valores para determinar si la última instrucción SQL se ha realizado correctamente.</p>
Para declarar las variables de host SQLCODE y SQLSTATE:	<p>a. Declare la variable SQLCODE dentro de una declaración BEGIN DECLARE SECTION y una declaración END DECLARE SECTION en las declaraciones de su programa como PIC S9(9) COMP-5.</p> <p>Cuando utilice el precompilador de SQL (Db2), puede declarar una variable SQLCODE independiente en la WORKING-STORAGE SECTION o en la LINKAGE SECTION. Cuando se utiliza el coprocesador de SQL (Db2), se puede declarar una variable SQLCODE independiente en la WORKING-STORAGE SECTION, LINKAGE SECTION o LOCAL-STORAGE SECTION.</p>

Opción	Descripción
	<p>b. Declare la variable SQLSTATE dentro de una instrucción BEGIN DECLARE SECTION y una instrucción END DECLARE SECTION en sus declaraciones de programa como PICTURE X(5).</p> <p>Restricción: No declare una variable SQLSTATE como un elemento de una estructura.</p> <p>Requisito: Después de declarar las variables SQLCODE y SQLSTATE, asegúrese de que todas las sentencias SQL del programa estén dentro del ámbito de la declaración de estas variables.</p>

Tareas relacionadas

[Comprobación de la ejecución de sentencias SQL](#)

Después de ejecutar una sentencia SQL, el programa debe comprobar si hay errores antes de confirmar los datos y manejar los errores que representan.

[Comprobación de la ejecución de sentencias SQL utilizando SQLCA](#)

Un modo de comprobar si una sentencia SQL se ha ejecutado correctamente es utilizar el área de comunicación SQL (SQLCA). Esta área se distingue de la comunicación con Db2.

[Comprobación de la ejecución de sentencias SQL utilizando SQLCODE y SQLSTATE](#)

Siempre que se ejecuta una sentencia SQL, los campos SQLCODE y SQLSTATE de SQLCA reciben un código de retorno.

[Definición de elementos que su programa puede utilizar para comprobar si una sentencia SQL se ha ejecutado correctamente](#)

Si su programa contiene sentencias SQL, el programa debe definir determinada infraestructura para poder comprobar si las sentencias se han ejecutado correctamente. También puede incluir un área de comunicación SQL (SQLCA), que contenga variables SQLCODE y SQLSTATE, o declarar variables host SQLCODE y SQLSTATE.

Definición de áreas de descriptor de SQL (SQLDA) en COBOL

Si su programa incluye determinadas sentencias SQL, debe definir al menos un área de descriptor SQL (SQLDA). Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Procedimiento

Realice una de las acciones siguientes:

- Codifique las declaraciones SQLDA directamente en su programa. Cuando utilice el precompilador Db2 , debe colocar las declaraciones SQLDA en la WORKING-STORAGE SECTION o LINKAGE SECTION de su programa, siempre que pueda especificar una entrada de descripción de registro en esa sección. Cuando utilice el coprocesador Db2 , debe colocar las declaraciones SQLDA en la WORKING-STORAGE SECTION, LINKAGE SECTION o LOCAL-STORAGE SECTION de su programa, siempre que pueda especificar una entrada de descripción de registro en esa sección.
- Llamar a una subrutina escrita en C, PL/I o lenguaje ensamblador y que utilice la sentencia INCLUDE SQLDA para definir el SQLDA. La subrutina también puede incluir instrucciones SQL para cualquier función SQL dinámica que necesite.

Restricción:

- Debe colocar las declaraciones SQLDA antes de la primera instrucción SQL que haga referencia al descriptor de datos, a menos que utilice la opción de procesamiento TWOPASS SQL.

Tareas relacionadas

[Definición de áreas de descriptor de SQL \(SQLDA\)](#)

Si su programa incluye ciertas sentencias SQL, debe definir al menos *un área de descriptor SQL (SQLDA)*. Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Referencia relacionada

[Área de descriptor de SQL \(SQLDA\) \(Db2 SQL\)](#)

Declaración de variables host y variables de indicador en COBOL

Puede utilizar variables de host, matrices de variables de host y estructuras de host en instrucciones SQL en su programa para pasar datos entre Db2 y su aplicación.

Procedimiento

Para declarar variables de host, matrices de variables de host y estructuras de host:

1. Declare las variables de acuerdo con las siguientes reglas y directrices:

- Debe declarar explícitamente todas las variables de host y matrices de variables de host que se utilizan en las sentencias SQL en la SECCIÓN DE ALMACENAMIENTO DE TRABAJO o en la SECCIÓN DE ENLACE de la DIVISIÓN DE DATOS de su programa.
- Debe declarar explícitamente cada variable de host y matriz de variables de host antes de utilizarlas en una instrucción SQL.
- Puede especificar OCCURS al definir una estructura de indicador, una matriz de variables de host o una matriz de variables de indicador. No puede especificar OCCURS para ningún otro tipo de variable de host.
- No puede declarar implícitamente ninguna variable de host mediante la tipificación predeterminada o utilizando la sentencia IMPLICIT.
- Si especifica la opción de procesamiento ONEPASS SQL, debe declarar explícitamente cada variable de host y cada matriz de variables de host antes de utilizarlas en una instrucción SQL. Si especifica la opción del precompilador TWOPASS, debe declarar cada variable de host antes de usarla en la instrucción DECLARE CURSOR.
- Si especifica la opción de procesamiento SQL STDSQL(YES), debe preceder las sentencias del lenguaje del host que definen las variables del host y las matrices de variables del host con la sentencia BEGIN DECLARE SECTION y seguir las sentencias del lenguaje del host con la sentencia END DECLARE SECTION. De lo contrario, estas declaraciones son opcionales.
- Asegúrese de que cualquier instrucción SQL que utilice una variable de host o una matriz de variables de host esté dentro del ámbito de la instrucción que declara esa variable o matriz.
- Si utiliza el lenguaje de programación C (Db2 precompilador), asegúrese de que los nombres de las variables de host y de las matrices de variables de host sean únicos dentro del programa, incluso si las variables y las matrices de variables se encuentran en bloques, clases, procedimientos, funciones o subrutinas diferentes. Puede calificar los nombres con un nombre de estructura para hacerlos únicos.

2. Opcional: Definir cualquier variable, matriz y estructura de indicadores asociados.

Tareas relacionadas

[Declaración de variables host y variables de indicación](#)

Puede utilizar variables host y variables de indicación en sentencias SQL del programa para pasar datos entre Db2 y la aplicación.

Variables de host en COBOL

En programas COBOL, puede especificar variables host numéricas, de caracteres, gráficas, binarias, LOB, XML y ROWID. También puede especificar conjuntos de resultados y localizadores de tabla y LOB, y variables de referencia de archivos LOB y XML.

Restricciones:

- Solo algunas de las declaraciones COBOL válidas son declaraciones de variables de host válidas. Si la declaración de una variable no es válida, cualquier instrucción SQL que haga referencia a la variable podría dar lugar al mensaje UNDECLARED HOST VARIABLE.
- No puede utilizar localizadores como tipos de columna.

Los siguientes tipos de datos de localización son tipos de datos COBOL y tipos de datos SQL:

- Localizador de conjunto de resultados
- Localizador de tablas
- localizadores de LOB
- Variables de referencia de archivo LOB

- Una o más entradas REDEFINES pueden seguir a cualquier entrada de descripción de datos de nivel 77. Sin embargo, no puede utilizar los nombres de estas entradas en sentencias SQL. Las entradas con el nombre FILLER se ignoran.

Recomendaciones:

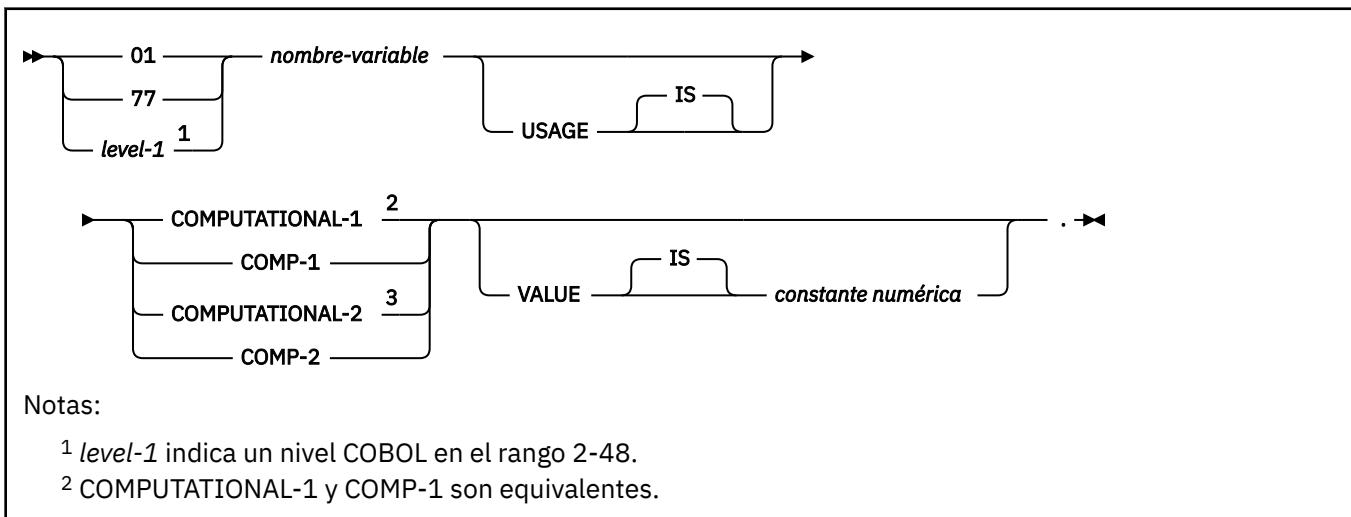
- Tenga cuidado con el desbordamiento. Por ejemplo, supongamos que recupera un valor de columna INTEGER en una variable de host PICTURE S9(4) y el valor de la columna es mayor que 32767 o menor que -32768. Recibirá una advertencia de desbordamiento o un error, dependiendo de si especifica una variable indicadora.
- Tenga cuidado con el truncamiento. Por ejemplo, si recupera un valor de columna CHAR de 80 caracteres en una variable de host PICTURE X(70), los 10 caracteres situados más a la derecha de la cadena recuperada se truncan. Recuperar un valor de columna decimal o de punto flotante de doble precisión en una variable host PIC (S9(8)) COMP elimina cualquier parte fraccionaria del valor. Del mismo modo, recuperar un valor de columna con tipo de datos DECIMAL en una variable decimal COBOL con una precisión inferior podría truncar el valor.
- Si las variables de host de cadena de longitud variable reciben valores cuya longitud es superior a 9999 bytes, compile las aplicaciones en las que utiliza esas variables de host con la opción TRUNC(BIN). TRUNC(BIN) permite que el campo de longitud de la cadena reciba un valor de hasta 32767 bytes.

Variables numéricas del host

Puede especificar las siguientes formas de variables de host numéricas:

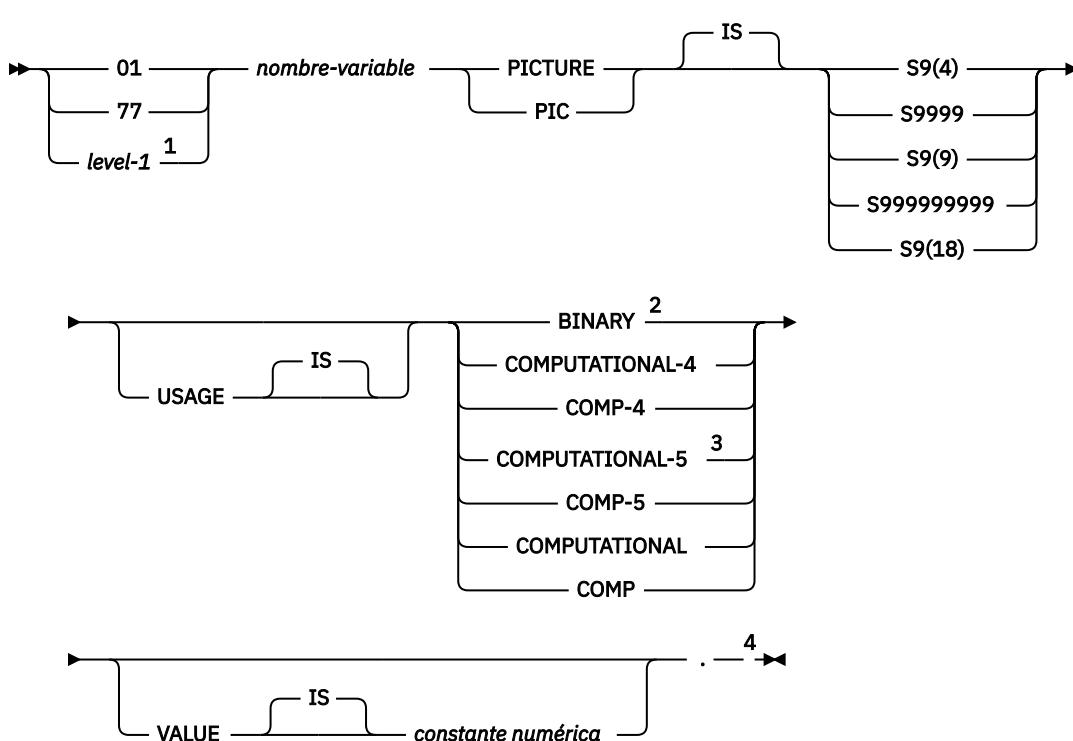
- números de coma flotante
- Números enteros y números enteros pequeños
- Números decimales

El siguiente diagrama muestra la sintaxis para declarar variables de host reales o de punto flotante.



³ COMPUTATIONAL-2 y COMP-2 son equivalentes.

El siguiente diagrama muestra la sintaxis para declarar variables de host enteras y enteras pequeñas.



Notas:

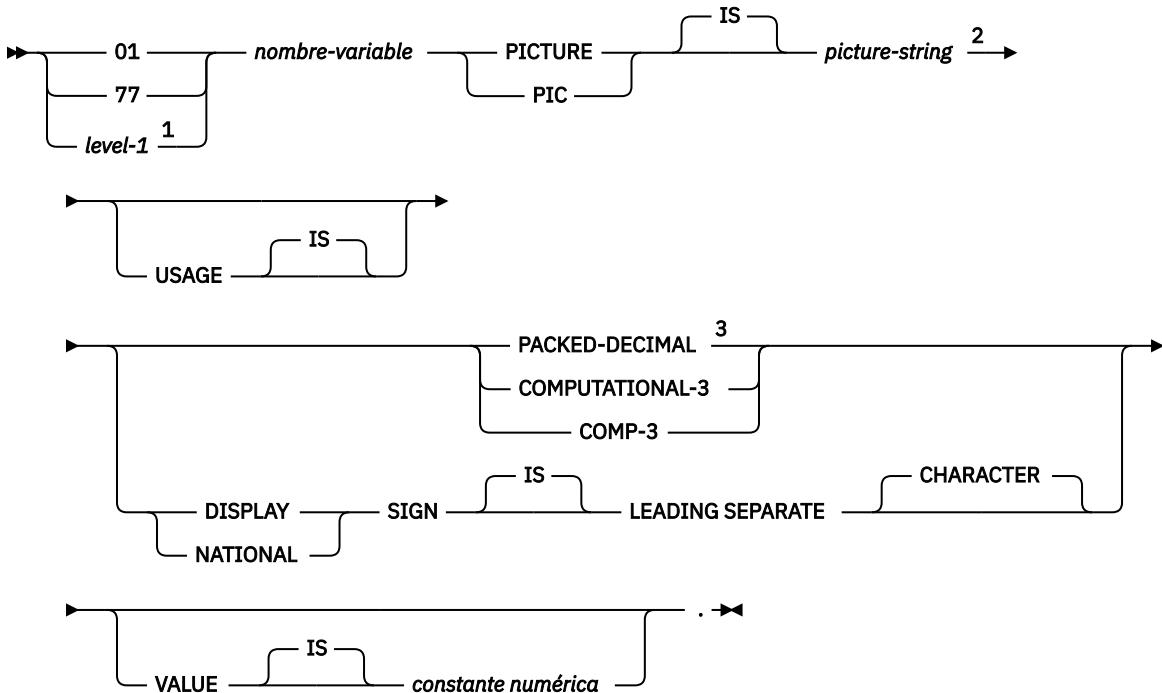
¹ level-1 indica un nivel COBOL en el rango 2-48.

² Los tipos de datos enteros binarios COBOL BINARY, COMPUTATIONAL, COMP, COMPUTATIONAL-4 y COMP-4 son equivalentes. Una aplicación portátil debe codificar BINARY, porque COMP, COMPUTATIONAL-4, COMP-4, COMPUTATIONAL-5 y COMP-5 son extensiones de COBOL (IBM) que no son compatibles con el lenguaje COBOL de la Organización Internacional de Normalización (ISO)/ANSI. Las declaraciones que utilizan COMP-5 en aplicaciones que utilizan la opción de compilación TRUNC(OPT) pueden evitar el truncamiento de datos que no caben en la cláusula de imagen asociada.

³ COMPUTATIONAL-5 (y COMP-5) son equivalentes a los otros tipos de datos enteros binarios COBOL si compila los otros tipos de datos con TRUNC(BIN).

⁴ Se ignora cualquier especificación de escala.

El siguiente diagrama muestra la sintaxis para declarar variables de host decimales.



Notas:

¹ *level-1* indica un nivel COBOL en el rango 2-48.

² La cadena de imágenes asociada con SIGN LEADING SEPARATE debe tener la forma S9(*i*)V9(*d*) (o S9...9V9...9, con *i* y *d* instancias de 9 o S9...9V con *i* instancias de 9).

³ PACKED-DECIMAL, COMPUTATIONAL-3 y COMP-3 son equivalentes. La cadena de imágenes asociada con estos tipos debe tener la forma S9(*i*)V9(*d*) (o S9...9V9...9, con *i* y *d* instancias de 9) o S9(*i*)V.

En COBOL, se declaran los tipos de datos SMALLINT e INTEGER como un número de dígitos decimales. Db2 utiliza el tamaño completo de los enteros (de forma similar al procesamiento con la opción del compilador TRUNC(BIN)) y puede colocar valores más grandes en la variable host de lo que se permitiría en el número especificado de dígitos en la declaración COBOL. Si compila con TRUNC(OPT) o TRUNC(STD), asegúrese de que el tamaño de los números en su aplicación esté dentro del número de dígitos declarado.

Para números enteros pequeños que pueden exceder 9999, utilice S9(4) COMP-5 o compile con TRUNC(BIN). Para números enteros grandes que pueden superar los 999 999 999, utilice S9(10) COMP-3 para obtener el tipo de datos decimal. Si utiliza COBOL para números enteros que superan el COBOL PICTURE, especifique la columna como decimal para asegurarse de que los tipos de datos coincidan y funcionen correctamente.

Si utiliza un compilador COBOL que no admite números decimales de más de 18 dígitos, utilice uno de los siguientes tipos de datos para almacenar valores de más de 18 dígitos:

- Una variable decimal con una precisión menor o igual a 18, si los valores de datos reales encajan. Si recupera un valor decimal en una variable decimal con una escala inferior a la columna de origen en la base de datos, la parte fraccionaria del valor podría truncarse.
- Un entero o una variable de punto flotante, que convierte el valor. Si utiliza una variable entera, perderá la parte fraccionaria del número. Si el número decimal puede exceder el valor máximo de un número entero o si desea conservar un valor fraccionario, utilice una variable de punto flotante. Los números de coma flotante son aproximaciones de números reales. Por lo tanto, cuando asignas un número decimal a una variable de punto flotante, el resultado puede ser diferente del número original.
- Una variable de host de cadena de caracteres. Utilice la función CHAR para recuperar un valor decimal en ella.

Restricción: El tipo de datos SQL DECFLOAT no tiene equivalente en COBOL.

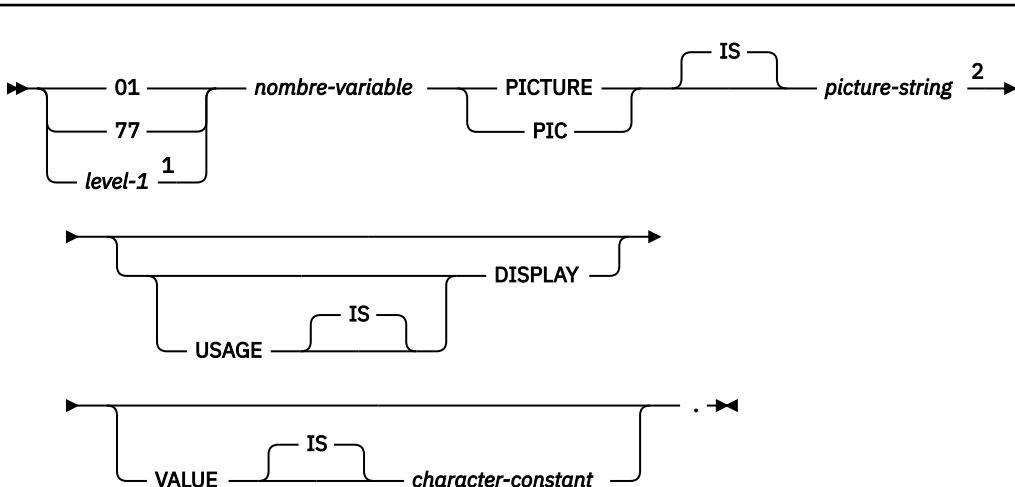
Variables de host de caracteres

Puede especificar las siguientes formas de variables de host de caracteres:

- Series de longitud fija
- Series de longitud variable
- CLOB

Los siguientes diagramas muestran la sintaxis para formularios distintos de CLOB.

El siguiente diagrama muestra la sintaxis para declarar variables de host de caracteres de longitud fija.

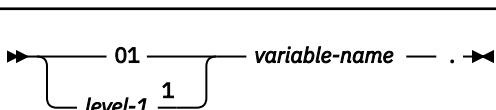


Notas:

¹ level-1 indica un nivel COBOL en el rango 2-48.

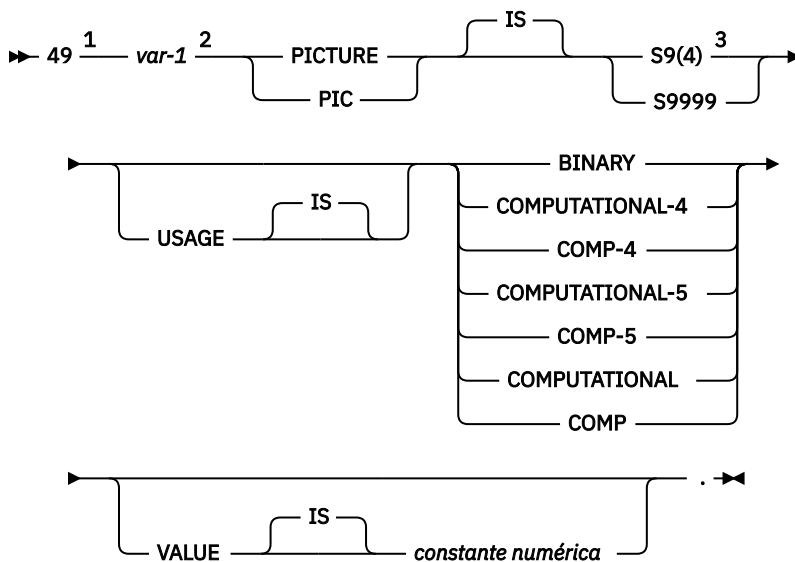
² La cadena de imagen que está asociada con estos formularios debe ser X(m) (o XX...X, con m instancias de X), donde m es hasta la limitación de COBOL. Sin embargo, la longitud máxima del tipo de datos CHAR (cadena de caracteres de longitud fija) en Db2 es de 255 bytes.

Los siguientes diagramas muestran la sintaxis para declarar variables de host de caracteres de longitud variable.



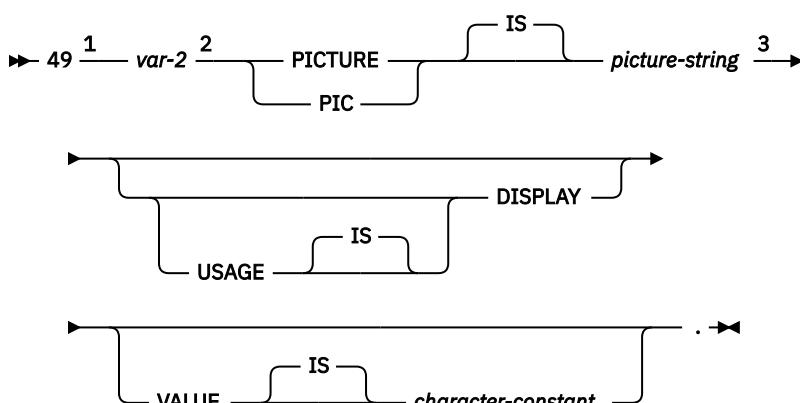
Notas:

¹ level-1 indica un nivel COBOL en el rango 2-48.



Notas:

- ¹ No puedes usar una REDEFINIR intermedia en el nivel 49.
- ² No puede hacer referencia directa a *var-1* como una variable de host.
- ³ Db2 utiliza la longitud completa de la variable binaria (S9(4)) aunque COBOL con TRUNC(STD) reconoce valores de hasta solo 9999. Este comportamiento puede causar errores de truncamiento de datos cuando se ejecutan las sentencias COBOL y podría limitar efectivamente la longitud máxima de las cadenas de caracteres de longitud variable a 9999. Considere la posibilidad de utilizar la opción del compilador TRUNC(BIN) o USAGE COMP-5 para evitar el truncamiento de datos.



Notas:

- ¹ No puedes usar una REDEFINIR intermedia en el nivel 49.
- ² No puede hacer referencia directa a *var-2* como una variable de host.
- ³ Para cadenas de longitud fija, *la cadena de imagen* debe ser $X(m)$ (o XX, con m instancias de X), donde m está *limitado por COBOL*. Sin embargo, la longitud máxima del tipo de datos VARCHAR en Db2 varía en función del tamaño de la página de datos.

Variables de host de caracteres gráficos

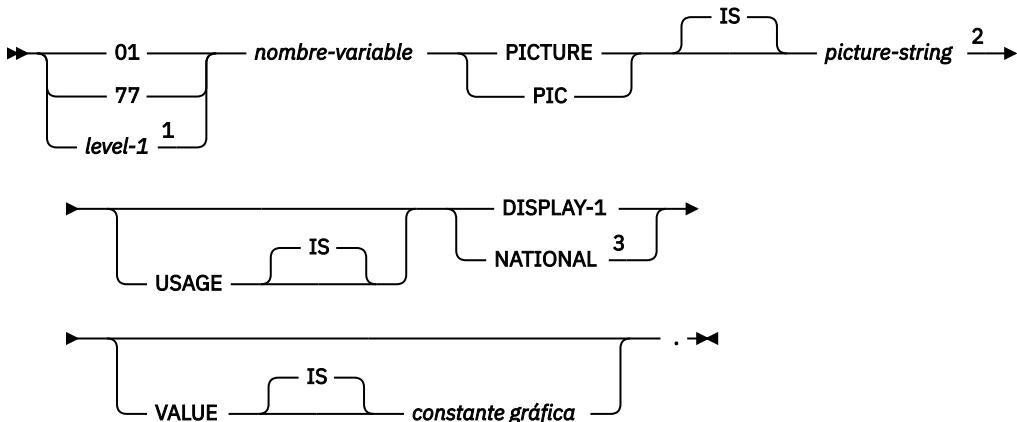
Puede especificar las siguientes formas de variables de host gráficas:

- Series de longitud fija

- Series de longitud variable
- DBCLOB

Los siguientes diagramas muestran la sintaxis para formularios distintos de DBCLOB.

El siguiente diagrama muestra la sintaxis para declarar variables host gráficas de longitud fija.



Notas:

¹ level-1 indica un nivel COBOL en el rango 2-48.

² Para cadenas de longitud fija, la cadena de imagen es G(m) o N(m) (o, m instancias de GG...G o NN...N), donde m es hasta la limitación de COBOL. Sin embargo, la longitud máxima del tipo de datos GRAPHIC (cadena gráfica de longitud fija) en Db2 es de 127 bytes dobles.

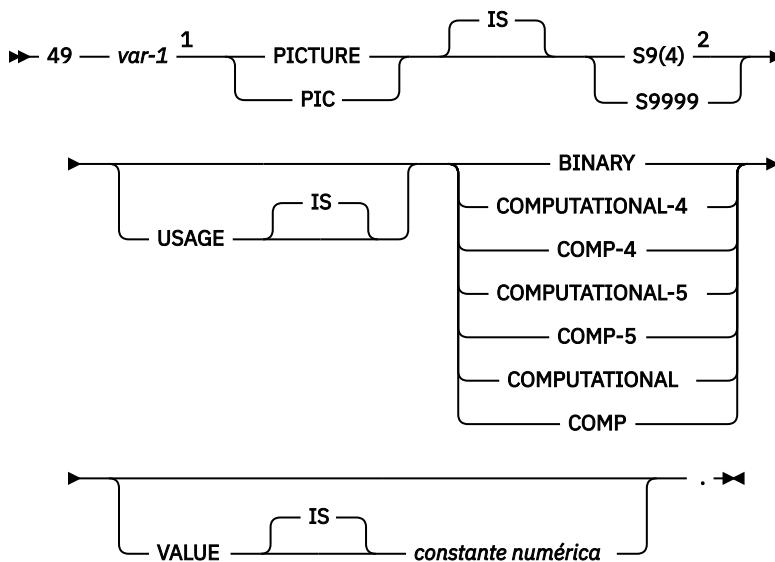
³ Utilice USAGE NATIONAL solo para datos de Unicode UTF-16. En la cadena de imagen para USAGE NATIONAL, debe usar N en lugar de G. USAGE NATIONAL solo es compatible con el coprocesador de la arquitectura Db2 .

Los siguientes diagramas muestran la sintaxis para declarar variables de host gráficas de longitud variable.



Notas:

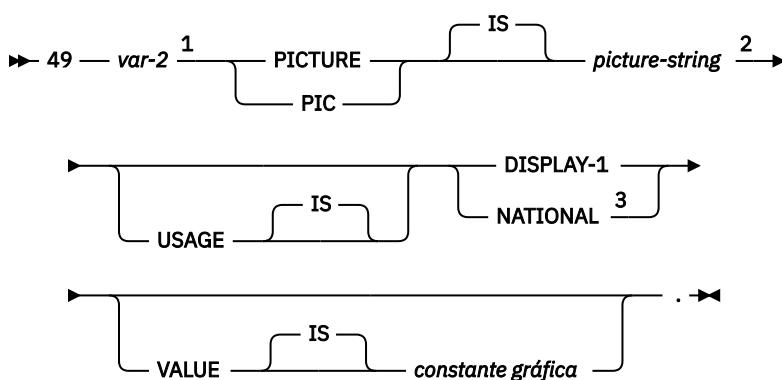
¹ level-1 indica un nivel COBOL en el rango 2-48.



Notas:

¹ No puede hacer referencia directa a `var-1` como una variable de host.

² Db2 utiliza la longitud completa de la variable binaria (`S9(4)`) aunque COBOL con `TRUNC(STD)` reconoce valores de hasta solo 9999. Este comportamiento puede causar errores de truncamiento de datos cuando se ejecutan las sentencias COBOL y podría limitar efectivamente la longitud máxima de las cadenas de caracteres de longitud variable a 9999. Considere la posibilidad de utilizar la opción del compilador `TRUNC(BIN)` o `USAGE COMP-5` para evitar el truncamiento de datos.



Notas:

¹ No puede hacer referencia directa a `var-2` como una variable de host.

² Para cadenas de longitud fija, *la cadena de imagen* es $G(m)$ o $N(m)$ (o, m instancias de GG...G o NN...N), donde m es hasta la limitación de COBOL. Sin embargo, la longitud máxima del tipo de datos **VARGRÁFICOS** en Db2 varía en función del tamaño de la página de datos.

³ Utilice `USAGE NATIONAL` solo para datos de Unicode UTF-16. En *la cadena de imagen* para `USAGE NATIONAL`, debe usar `N` en lugar de `G`. `USAGE NATIONAL` solo es compatible con el coprocesador de la arquitectura Db2 .

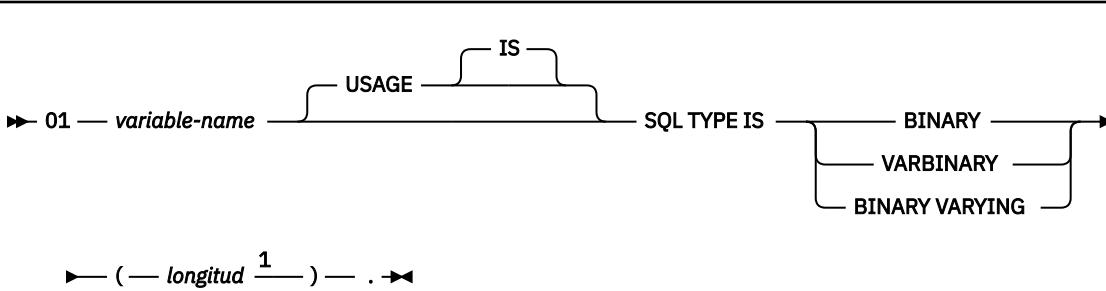
Variables binarias de host

Puede especificar las siguientes formas de variables de host binarias:

- Series de longitud fija

- Series de longitud variable
- BLOB

El siguiente diagrama muestra la sintaxis para declarar variables de host BINARY y VARBINARY.



Notas:

¹ Para las variables de host BINARIAS, la longitud debe estar en el rango de 1 a 255. Para las variables de host VARBINARY, la longitud debe estar en el formato "range1–32704".

COBOL no tiene variables que se correspondan con los tipos binarios SQL BINARY y VARBINARY. Para crear variables de host que puedan utilizarse con estos tipos de datos, utilice la cláusula SQL TYPE IS. El precompilador SQL reemplaza esta declaración con una estructura de lenguaje COBOL en el miembro de origen de salida.

Cuando se hace referencia a una variable de host BINARIA o VARBINARIA en una instrucción SQL, se debe utilizar la variable que se especifica en la declaración SQL TYPE. Cuando haga referencia a la variable de host en una instrucción de lenguaje de host, debe utilizar la variable que genera Db2 .

Ejemplos de declaraciones de variables binarias

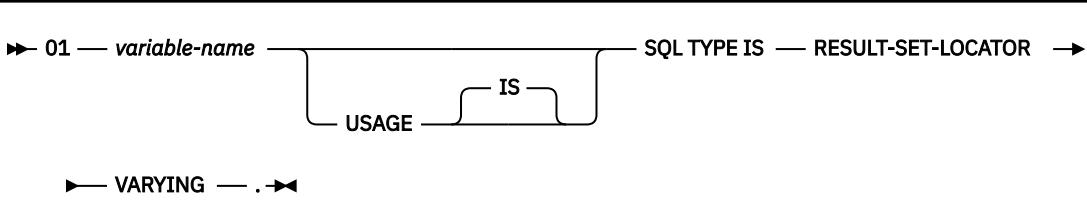
La siguiente tabla muestra ejemplos de variables que genera Db2 cuando se declaran variables de host binarias.

Tabla 107. Ejemplos de declaraciones de variables BINARIAS y VARBINARIAS para COBOL

Declaración variable que se incluye en el programa COBOL	Variable correspondiente que genera Db2 en el miembro de origen de salida
01 BIN-VAR USAGE IS SQL TYPE IS BINARY(10).	01 BIN-VAR PIC X(10).
01 VBIN-VAR USAGE IS SQL TYPE IS VARBINARY(10).	01 VBIN-VAR. 49 VBIN-VAR-LEN PIC S9(4) USAGE BINARY. 49 VBIN-VAR-TEXT PIC X(10).

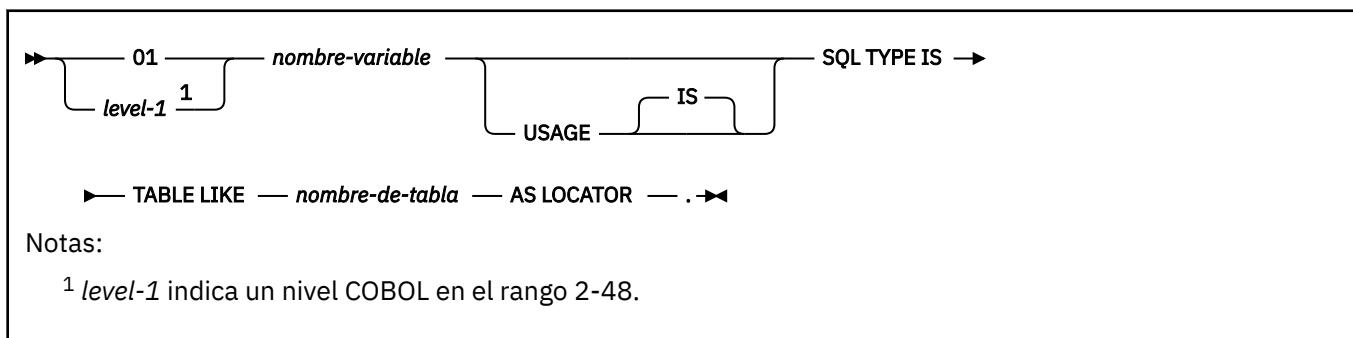
Localizadores de conjuntos de resultados

El siguiente diagrama muestra la sintaxis para declarar localizadores de conjuntos de resultados.



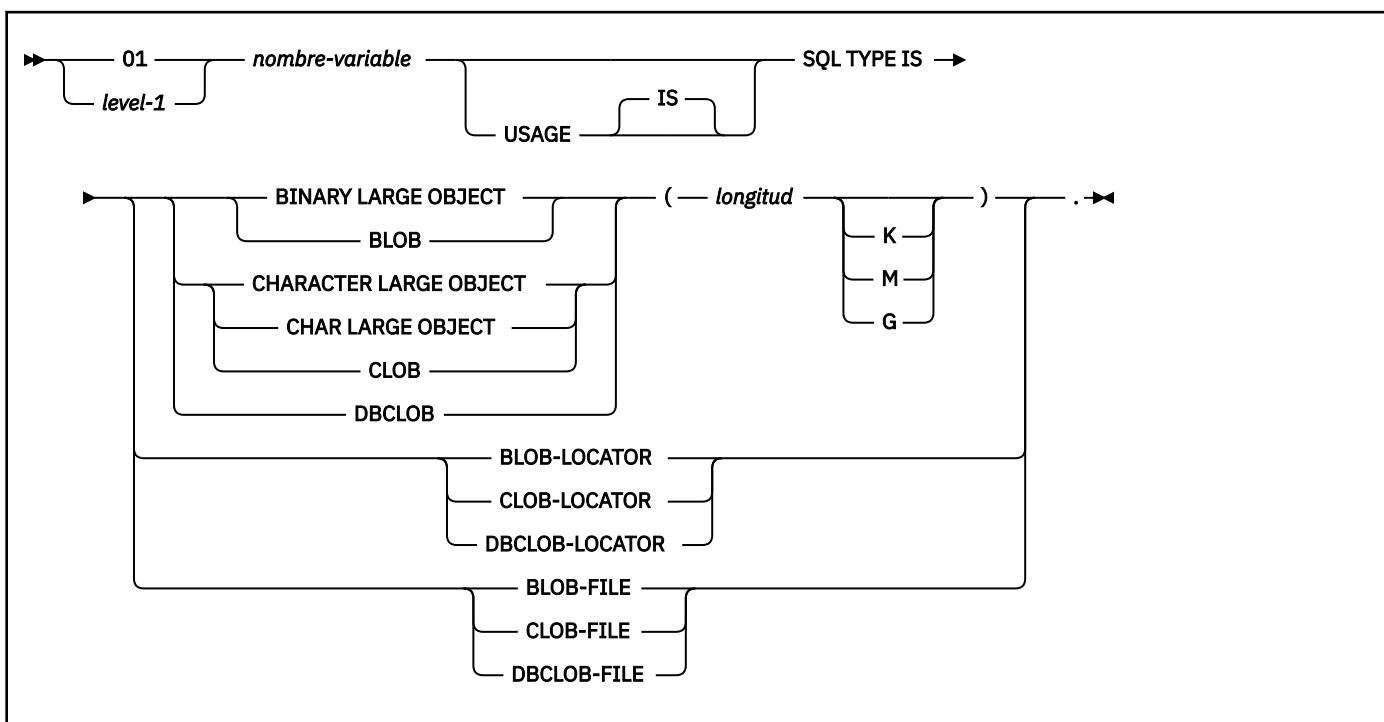
Localizadores de mesas

El siguiente diagrama muestra la sintaxis para declarar localizadores de tablas.



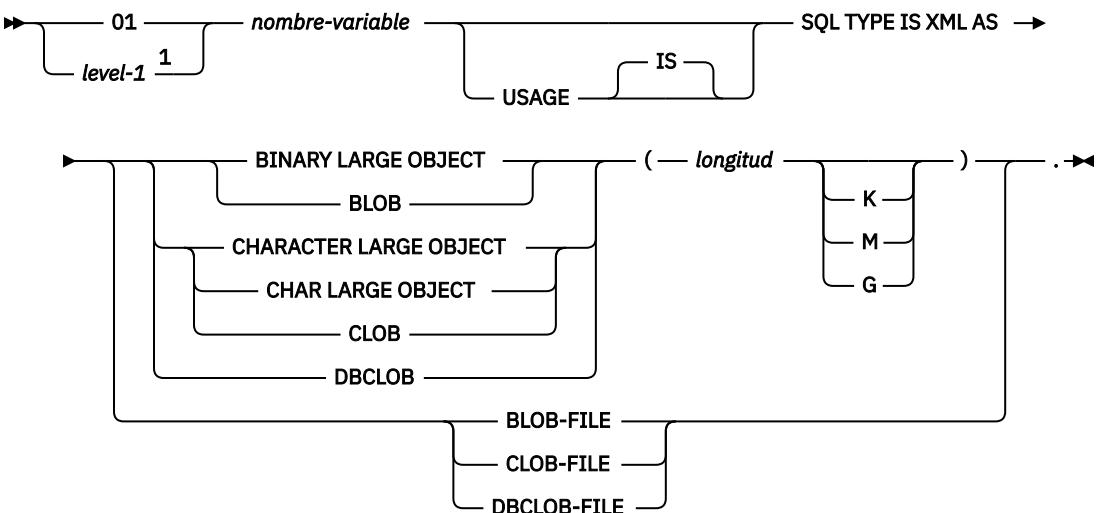
Variables LOB y variables de referencia de archivos

El siguiente diagrama muestra la sintaxis para declarar variables BLOB, CLOB y DBCLOB y variables de referencia de archivos.



Variables de referencia de archivos y host de datos XML

El siguiente diagrama muestra la sintaxis para declarar variables de host BLOB, CLOB y DBCLOB y variables de referencia de archivo para tipos de datos XML.

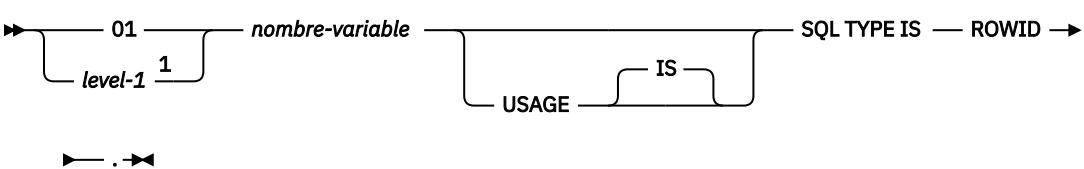


Notas:

¹ *level-1* indica un nivel COBOL en el rango 2-48.

Variables de host ROWID

El siguiente diagrama muestra la sintaxis para declarar variables de host ROWID.



Notas:

¹ *level-1* indica un nivel COBOL en el rango 2-48.

Conceptos relacionados

Variables de host

Utilice variables host para pasar un único elemento de datos entre Db2 y la aplicación.

Tareas relacionadas

Almacenamiento de datos LOB en tablas de Db2

Db2 maneja los datos LOB de manera diferente a otros tipos de datos. Como resultado, a veces es necesario realizar acciones adicionales al definir columnas LOB e insertar los datos LOB.

Referencia relacionada

Límites en Db2 for z/OS (Db2 SQL)

Matrices de variables de host en COBOL

En los programas COBOL, puede especificar matrices de variables de host de tipo numérico, carácter, gráfico, binario, LOB, XML y ROWID. También puede especificar localizadores LOB y variables de referencia de archivos LOB y XML.

Solo se puede hacer referencia a las matrices de variables de lenguaje principal como una referencia simple en los contextos siguientes. En los diagramas de sintaxis, *host-variable-array* designa una referencia a una matriz de variables de host.

- En una sentencia FETCH para una captación de varias filas. Consulte [FETCH declaración \(Db2 SQL\)](#).

- En la forma *FOR n ROWS* de la sentencia *INSERT* con una matriz de variables de host para los datos de origen. Consulte [INSERT declaración \(Db2 SQL\)](#).
- En una sentencia *MERGE* con varias filas de datos de origen. Consulte [MERGE declaración \(Db2 SQL\)](#).
- En una sentencia *EXECUTE* para proporcionar un valor para un marcador de parámetro en una forma dinámica *FOR n ROWS* de la sentencia *INSERT* o una sentencia *MERGE*. Consulte [EXECUTE declaración \(Db2 SQL\)](#).

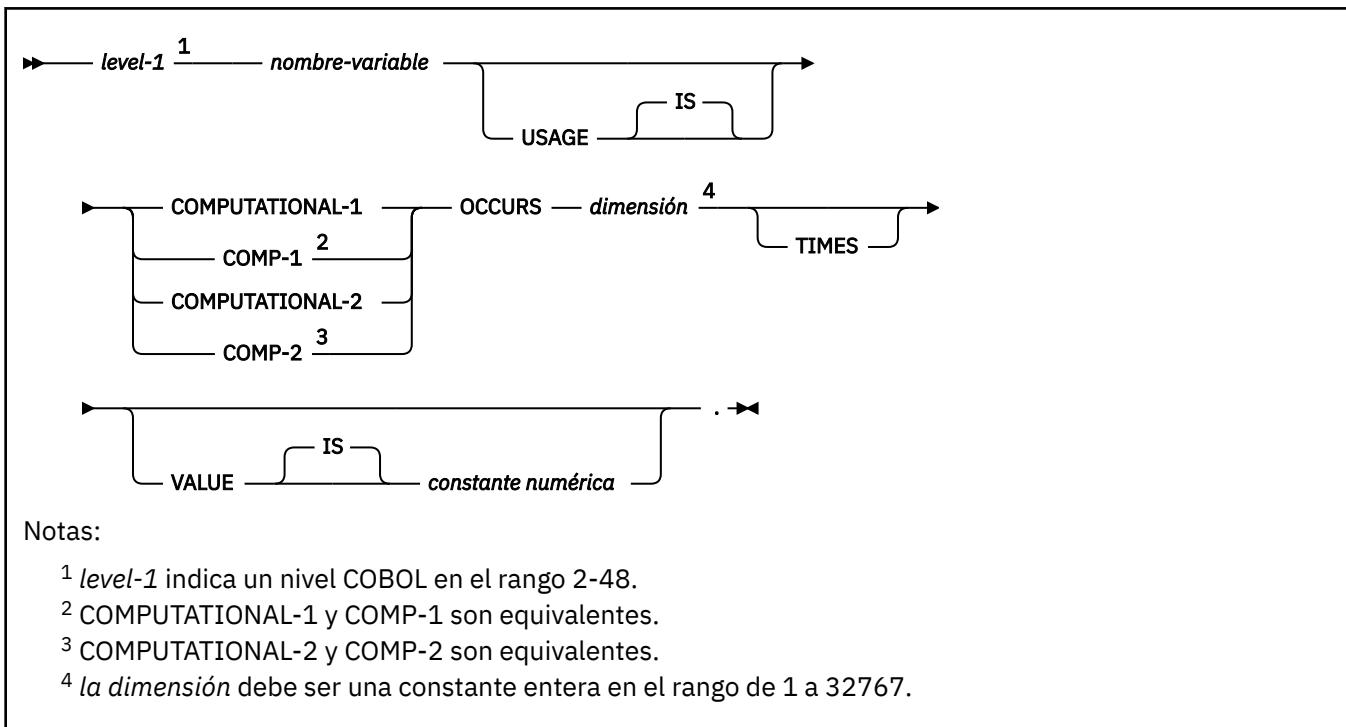
Restricción: Solo algunas de las declaraciones COBOL válidas son declaraciones de matrices de variables de host válidas. Si la declaración de una matriz variable no es válida, cualquier instrucción SQL que haga referencia a la matriz variable podría dar como resultado el mensaje UNDECLARED HOST VARIABLE ARRAY.

Matrices de variables de host numéricas

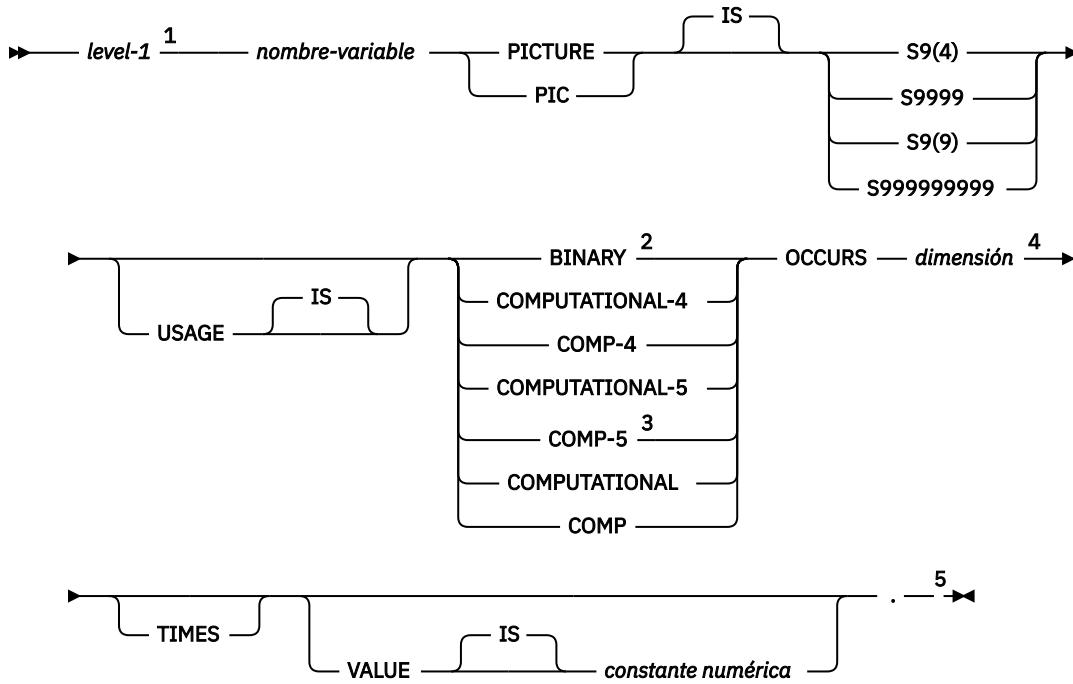
Puede especificar las siguientes formas de matrices de variables de host numéricas:

- números de coma flotante
- Números enteros y números enteros pequeños
- Números decimales

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host de punto flotante.



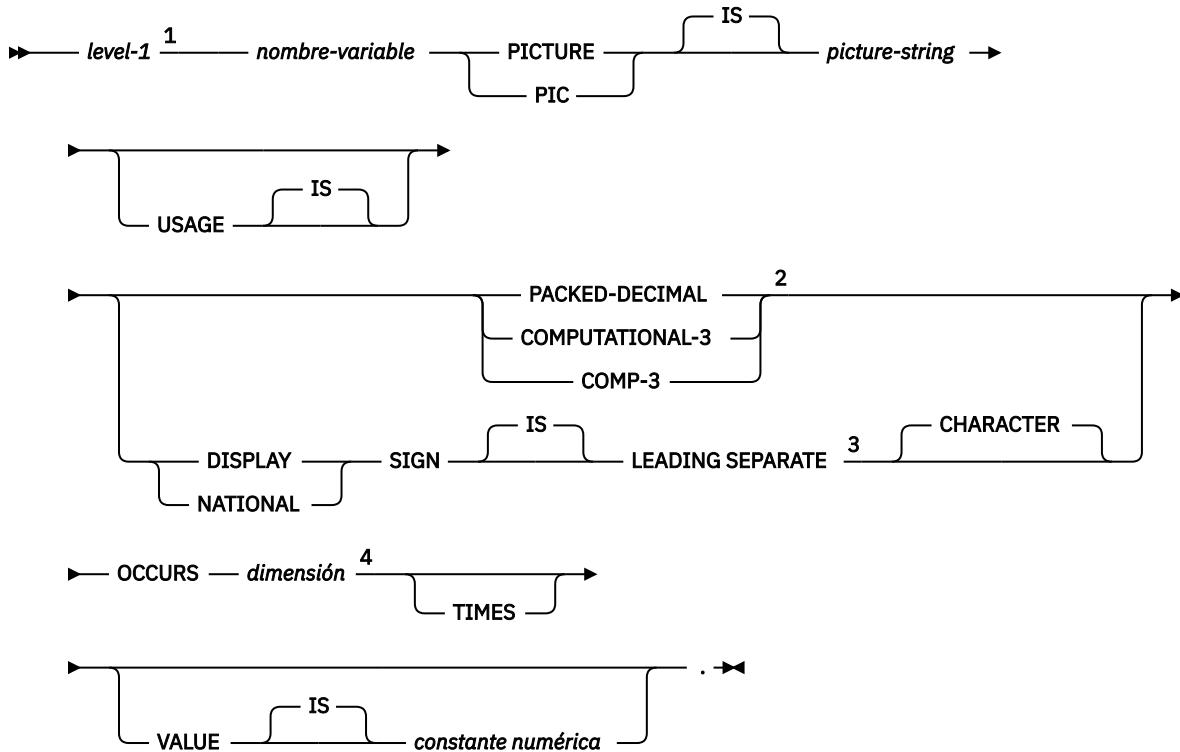
El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host enteras y pequeñas enteras.



Notas:

- ¹ *level-1* indica un nivel COBOL en el rango 2-48.
- ² Los tipos de datos enteros binarios COBOL BINARY, COMPUTATIONAL, COMP, COMPUTATIONAL-4 y COMPUTATIONAL-4 son equivalentes.
- ³ COMPUTATIONAL-5 (y COMP-5) son equivalentes a los otros tipos de datos enteros binarios COBOL si compila los otros tipos de datos con TRUNC(BIN).
- ⁴ *la dimensión* debe ser una constante entera en el rango de 1 a 32767.
- ⁵ Se ignora cualquier especificación de escala.

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host decimales.



Notas:

¹ level-1 indica un nivel COBOL en el rango 2-48.

² PACKED-DECIMAL, COMPUTATIONAL-3 y COMP-3 son equivalentes. La *cadena de imágenes* asociada con estos tipos debe tener la forma S9(*i*)V9(*d*) (o S9...9V9...9, con *i* y *d* instancias de 9) o S9(*i*)V.

³ La *cadena de imágenes* asociada con SIGN LEADING SEPARATE debe tener la forma S9(*i*)V9(*d*) (o S9...9V9...9, con *i* y *d* instancias de 9 o S9...9V con *i* instancias de 9).

⁴ la *dimensión* debe ser una constante entera en el rango de 1 a 32767.

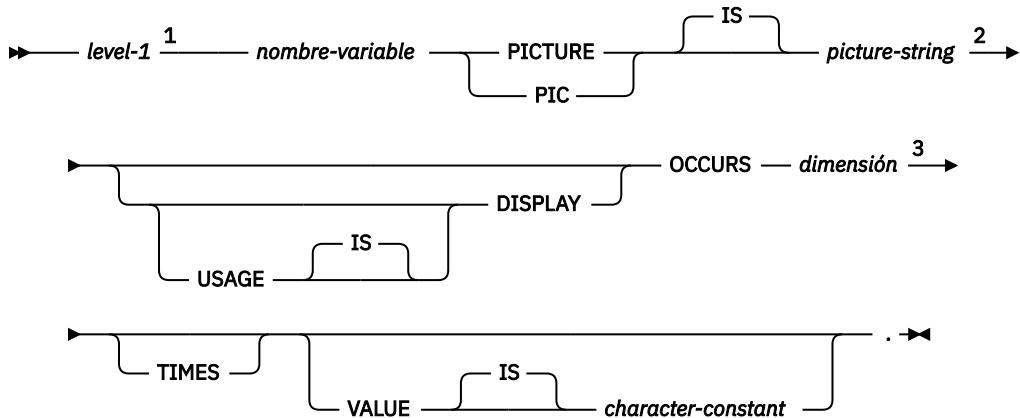
Matrices de variables de host de caracteres

Puede especificar las siguientes formas de matrices de variables de host de caracteres:

- Series de caracteres de longitud fija
- Series de caracteres de longitud variable
- CLOB

Los siguientes diagramas muestran la sintaxis para formularios distintos de CLOB.

El siguiente diagrama muestra la sintaxis para declarar matrices de cadenas de caracteres de longitud fija.



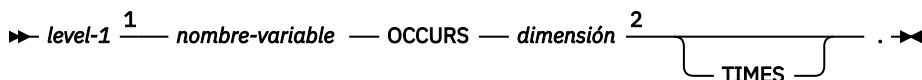
Notas:

¹ *level-1* indica un nivel COBOL en el rango 2-48.

² La cadena de caracteres debe tener la forma $X(m)$ (o $XX\dots X$, con m instancias de X), donde $1 \leq m \leq 32767$ para cadenas de longitud fija. Sin embargo, la longitud máxima del tipo de datos CHAR (cadena de caracteres de longitud fija) en Db2 es de 255 bytes.

³ La dimensión debe ser una constante entera en el rango de 1 a 32767.

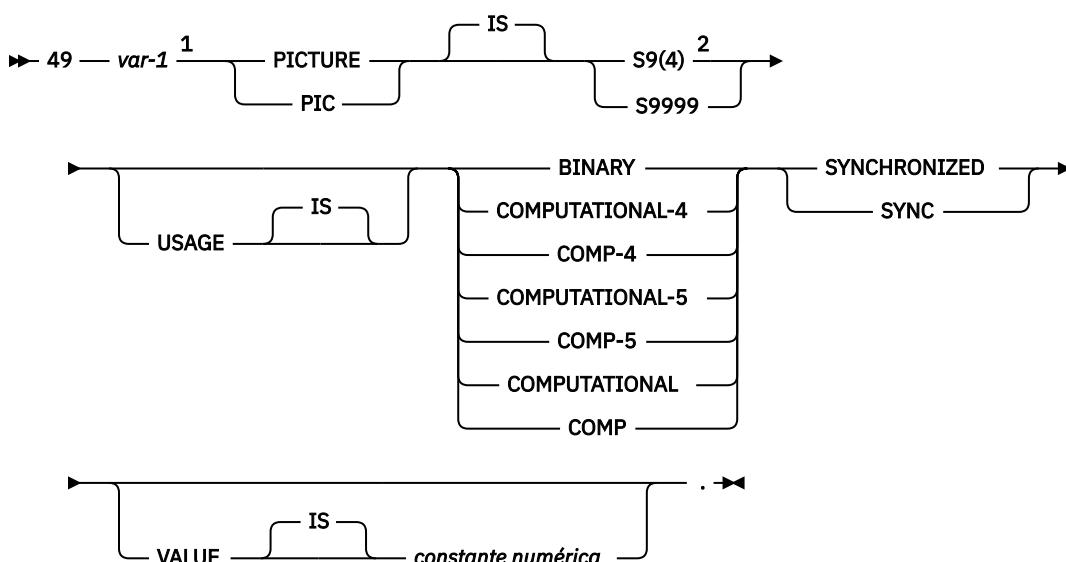
Los siguientes diagramas muestran la sintaxis para declarar matrices de cadenas de caracteres de longitud variable.



Notas:

¹ *level-1* indica un nivel COBOL en el rango 2-48.

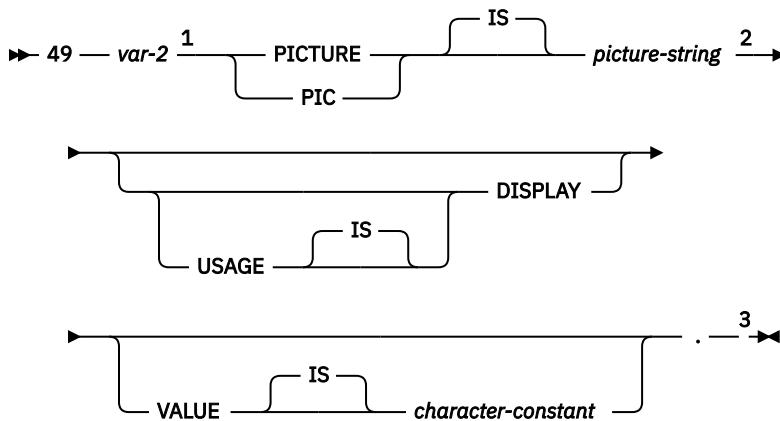
² La dimensión debe ser una constante entera en el rango de 1 a 32767.



Notas:

¹ No puede hacer referencia directa a *var-1* como una matriz de variables de host.

² Db2 utiliza la longitud completa de la variable binaria (S9(4)) aunque COBOL con TRUNC(STD) reconoce valores de hasta solo 9999. Este comportamiento puede causar errores de truncamiento de datos cuando se ejecutan las sentencias COBOL y podría limitar efectivamente la longitud máxima de las cadenas de caracteres de longitud variable a 9999. Considera la posibilidad de utilizar la opción del compilador TRUNC(BIN) o USAGE COMP-5 para evitar el truncamiento de datos.



Notas:

¹ No puede hacer referencia directa a *var-2* como una matriz de variables de host.

² La cadena de caracteres debe tener la forma *X(m)* (o *XX...X*, con *m* instancias de *X*), donde $1 \leq m \leq 32767$ para cadenas de longitud fija; para otras cadenas, *m* no puede ser mayor que el tamaño máximo de una cadena de caracteres de longitud variable.

³ No puedes usar una REDEFINIR intermedia en el nivel 49.

El siguiente ejemplo muestra declaraciones de una matriz de caracteres de longitud fija y una matriz de caracteres de longitud variable.

```

01 OUTPUT-VARS.
  05 NAME OCCURS 10 TIMES.
    49 NAME-LEN  PIC S9(4) COMP-4 SYNC.
    49 NAME-DATA PIC X(40).
  05 SERIAL-NUMBER PIC S9(9) COMP-4 OCCURS 10 TIMES.
  
```

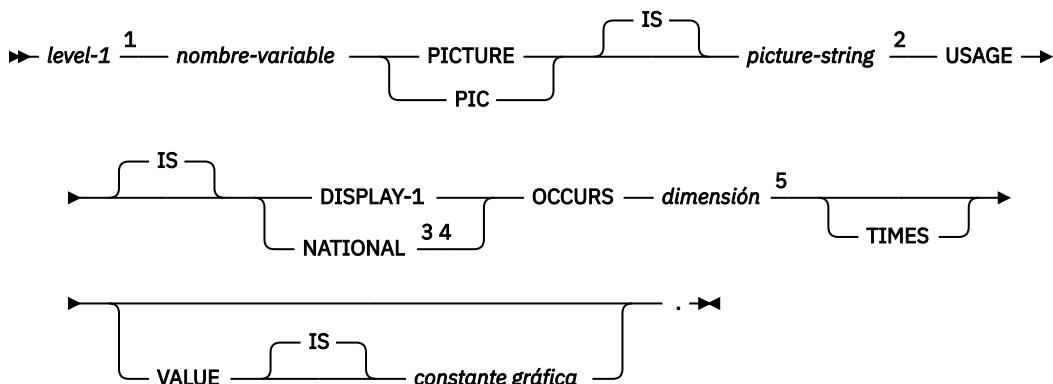
Matrices de variables de host de caracteres gráficos

Puede especificar las siguientes formas de matrices de variables de host gráficas:

- Series de longitud fija
- Series de longitud variable
- DBCLOB

Los siguientes diagramas muestran la sintaxis para formularios distintos de DBCLOB.

El siguiente diagrama muestra la sintaxis para declarar matrices de cadenas gráficas de longitud fija.



Notas:

¹ level-1 indica un nivel COBOL en el rango 2-48.

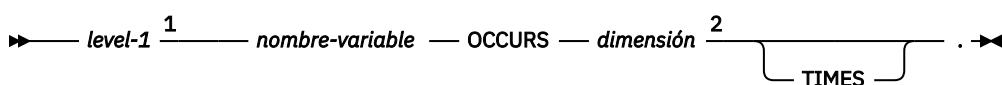
² Para cadenas de longitud fija, el formato para la cadena de imagen es $G(m)$ o $N(m)$ (o, m instancias de $GG\dots G$ o $NN\dots N$), donde $1 \leq m \leq 127$; para otras cadenas, m no puede ser mayor que el tamaño máximo de una cadena gráfica de longitud variable.

³ Utilice USAGE NATIONAL solo para datos de Unicode UTF-16. En la cadena de imágenes para USAGE NATIONAL, debe usar N en lugar de G.

⁴ Puede utilizar USAGE NATIONAL solo si está utilizando el coprocesador de la arquitectura de procesador de señal mixta (Db2).

⁵ La dimensión debe ser una constante entera en el rango de 1 a 32767.

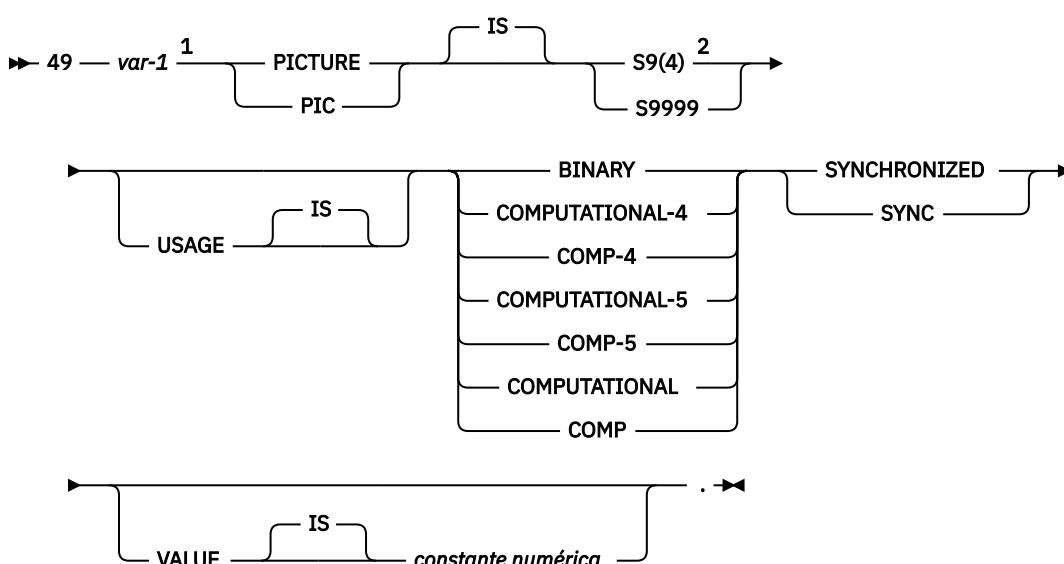
Los siguientes diagramas muestran la sintaxis para declarar matrices de cadenas gráficas de longitud variable.



Notas:

¹ *level-1* indica un nivel COBOL en el rango 2-48.

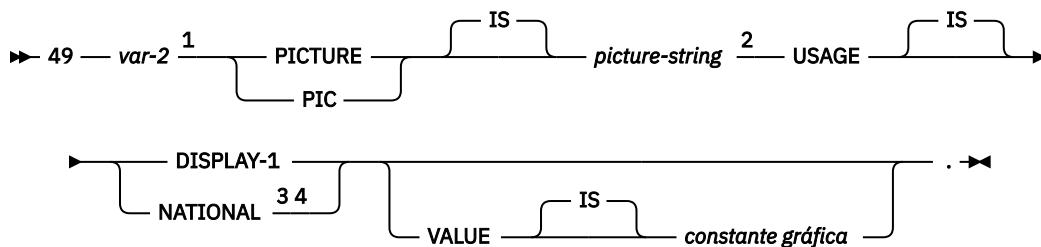
² la dimensión debe ser una constante entera en el rango de 1 a 32767.



Notas:

¹ No puede hacer referencia directa a *var-1* como una matriz de variables de host.

² Db2 utiliza la longitud completa de la variable binaria (S9(4)) aunque COBOL con TRUNC(STD) reconoce valores de hasta solo 9999. Este comportamiento puede causar errores de truncamiento de datos cuando se ejecutan las sentencias COBOL y podría limitar efectivamente la longitud máxima de las cadenas de caracteres de longitud variable a 9999. Considere la posibilidad de utilizar la opción del compilador TRUNC(BIN) o USAGE COMP-5 para evitar el truncamiento de datos.



Notas:

¹ No puede hacer referencia directa a *var-2* como una matriz de variables de host.

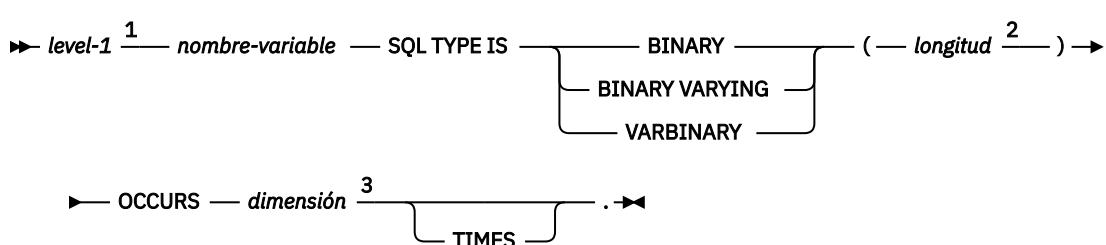
² Para cadenas de longitud fija, el formato para *la cadena de imagen* es *G(m)* o *N(m)* (o, *m* instancias de *GG...G* o *NN...N*), donde $1 \leq m \leq 127$; para otras cadenas, *m* no puede ser mayor que el tamaño máximo de una cadena gráfica de longitud variable.

³ Utilice USAGE NATIONAL solo para datos de Unicode UTF-16. En *la cadena de imágenes* para USAGE NATIONAL, debe usar N en lugar de G.

⁴ Puede utilizar USAGE NATIONAL solo si está utilizando el coprocesador de la arquitectura de procesador de señal mixta (Db2).

Matrices binarias de variables de host

El siguiente diagrama muestra la sintaxis para declarar matrices binarias de variables de host.



Notas:

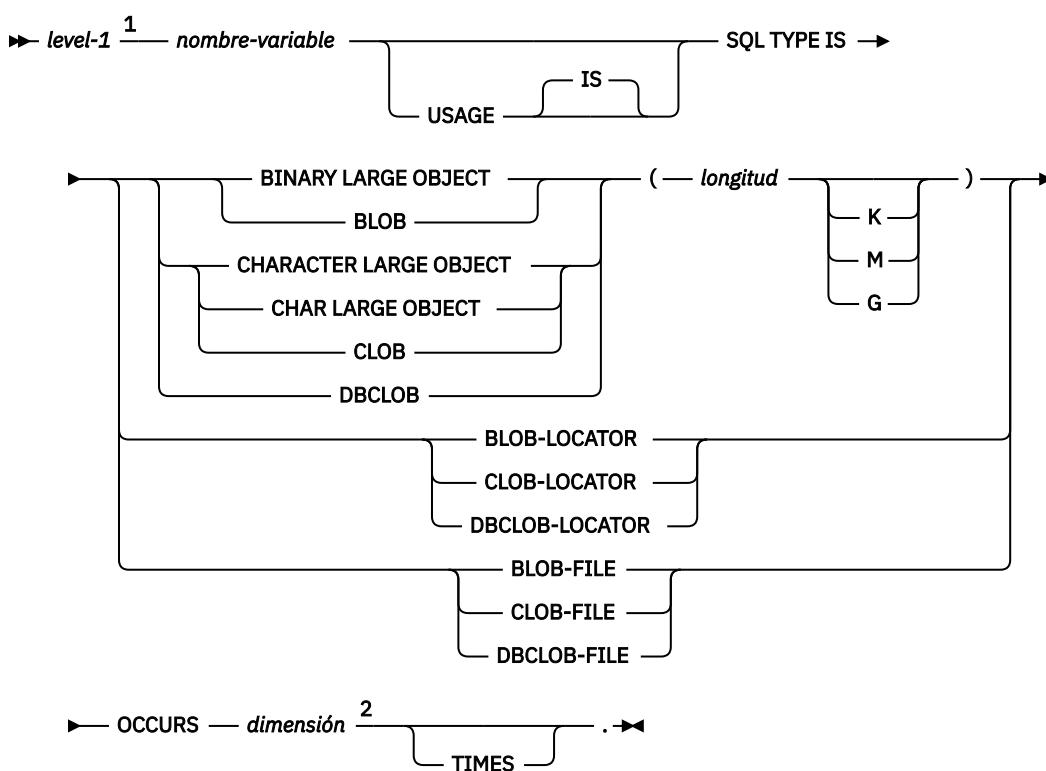
¹ *level-1* indica un nivel COBOL en el rango 2-48.

² Para las variables de host BINARIAS, *la longitud* debe estar en el rango de 1 a 255. Para las variables de host VARBINARY, *la longitud* debe estar en el rango de 1 a 32704.

³ *la dimensión* debe ser una constante entera en el rango de 1 a 32767.

LOB, localizador y matrices de variables de referencia de archivos

El siguiente diagrama muestra la sintaxis para declarar matrices de referencia de archivo, localizador y variable de host BLOB, CLOB y DBCLOB.



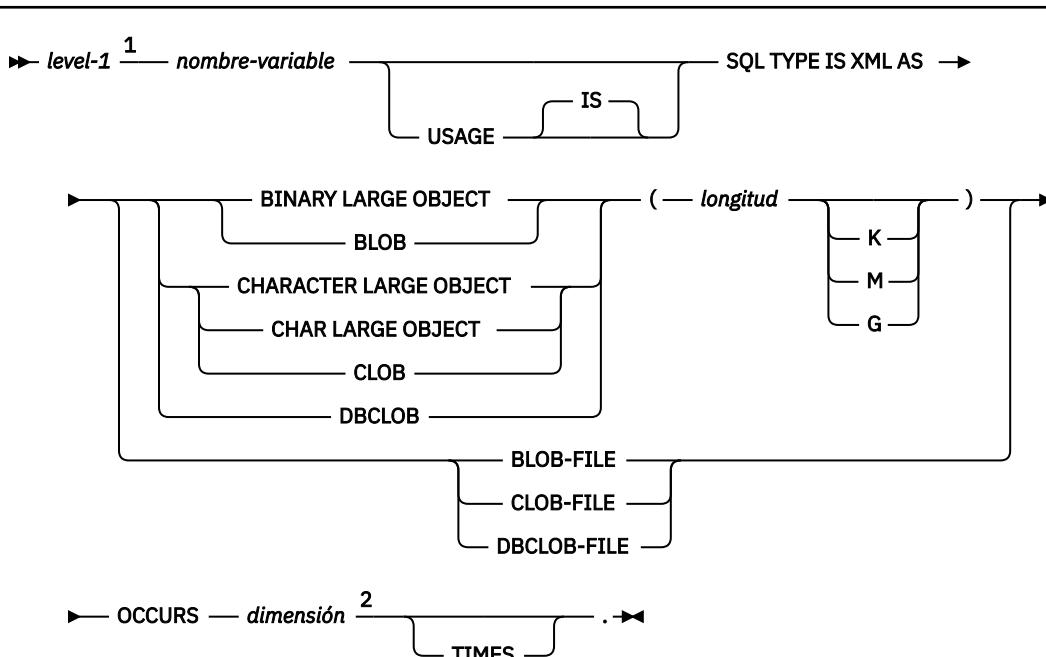
Notas:

¹ level-1 indica un nivel COBOL en el rango 2-48.

² la dimensión debe ser una constante entera en el rango de 1 a 32767.

Variables de matriz de referencia de archivos y host XML

El siguiente diagrama muestra la sintaxis para declarar variables de host BLOB, CLOB y DBCLOB y matrices de referencia de archivos para tipos de datos XML.



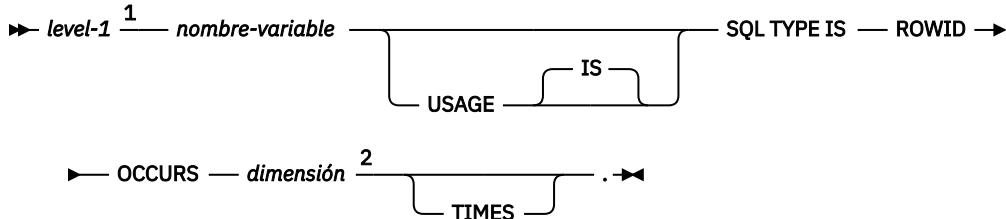
Notas:

¹ *level-1* indica un nivel COBOL en el rango 2-48.

² *la dimensión* debe ser una constante entera en el rango de 1 a 32767.

Matrices de variables ROWID

El siguiente diagrama muestra la sintaxis para declarar matrices de variables ROWID.



Notas:

¹ *level-1* indica un nivel COBOL en el rango 2-48.

² *la dimensión* debe ser una constante entera en el rango de 1 a 32767.

Conceptos relacionados

[Utilización de matrices de variables de host en sentencias SQL](#)

Utilice matrices de variables de lenguaje de host en sentencias de SQL incorporado para representar valores que el programa no conoce hasta que se ejecuta la consulta. Las matrices de variables de host son útiles para almacenar un conjunto de valores recuperados o para pasar un conjunto de valores que se van a insertar en una tabla.

Matrices de variables de host

Puede utilizar matrices de variables de host para pasar una matriz de datos entre Db2 y su aplicación.

Una matriz de variables de host es una matriz de datos que se declara en el lenguaje de host para ser utilizada dentro de una instrucción SQL.

[Matrices de variables de lenguaje principal en PL/I, C, C++ y COBOL \(Db2 SQL\)](#)

Tareas relacionadas

[Inserción de varias filas de datos desde matrices de variables de host](#)

Utilice las variables de host en la sentencia INSERT cuando no conozca al menos algunos de los valores que se han de insertar hasta que el programa se ejecuta.

Almacenamiento de datos LOB en tablas de Db2

Db2 maneja los datos LOB de manera diferente a otros tipos de datos. Como resultado, a veces es necesario realizar acciones adicionales al definir columnas LOB e insertar los datos LOB.

[Recuperación de varias filas de datos en matrices de variables de host](#)

Si sabe que la consulta devuelve varias filas, puede especificar matrices de variables de host para almacenar los valores de columna recuperados.

Estructuras host en COBOL

Una estructura host en COBOL es un conjunto con nombre de variables host que se definen en la WORKING-STORAGE SECTION o LINKAGE SECTION del programa.

Requisitos: Las declaraciones de estructura de host en COBOL deben cumplir los siguientes requisitos:

- Las estructuras de host COBOL pueden tener un máximo de dos niveles, aunque la estructura de host puede darse dentro de una estructura con varios niveles. Sin embargo, puede declarar una cadena de caracteres de longitud variable, que debe ser de nivel 49.
- El nombre de una estructura anfitriona puede ser un nombre de grupo cuyos niveles subordinados nombren elementos de datos elementales.

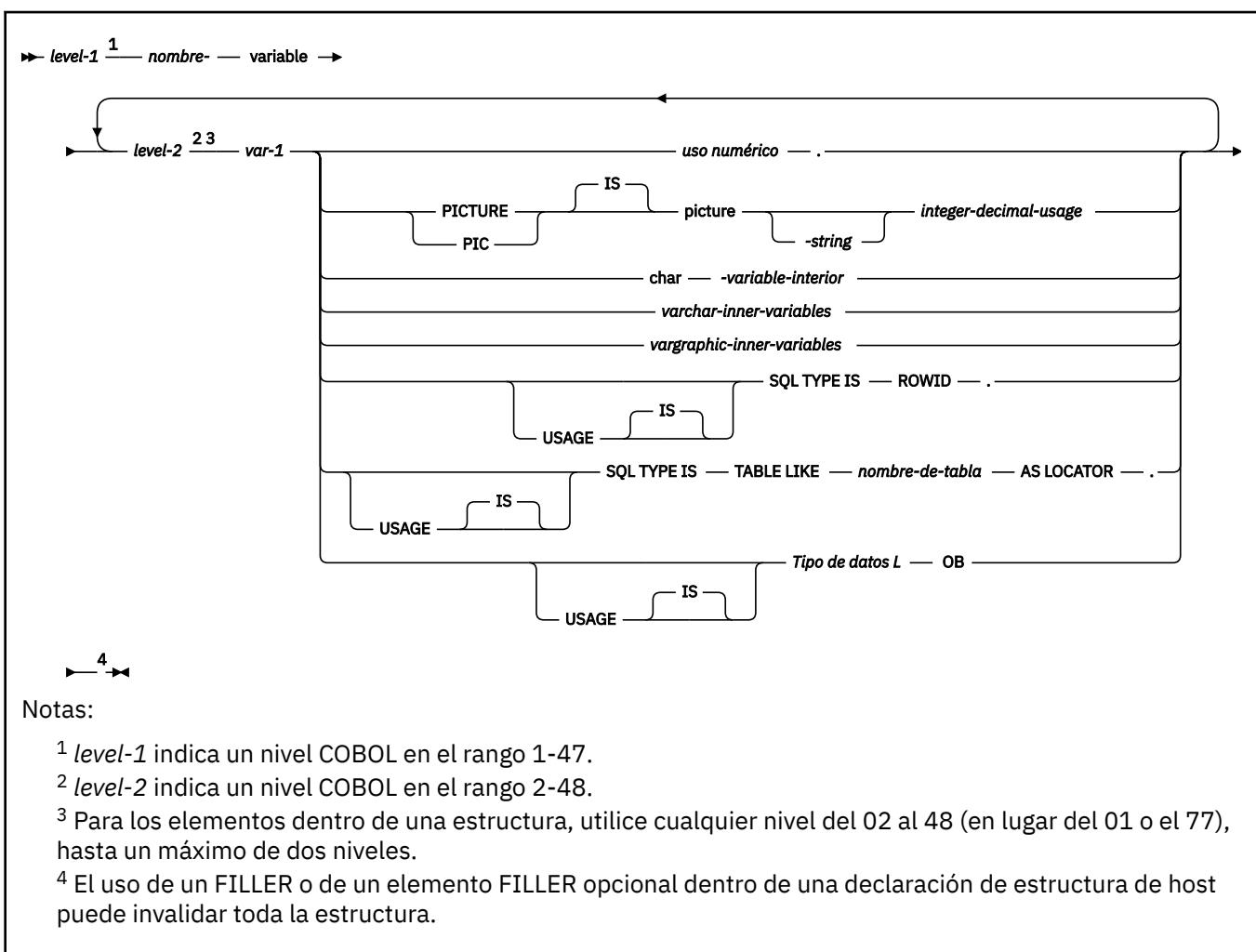
- Si utiliza el precompilador Db2 , no declare variables de host ni estructuras de host en ningún nivel subordinado después de uno de los siguientes elementos:
 - Un elemento COBOL que comienza en el área A
 - Cualquier instrucción SQL (excepto SQL INCLUDE)
 - Cualquier instrucción SQL dentro de un miembro incluido

Cuando el precompilador de C++ (Db2) encuentra uno de los elementos anteriores en una estructura de host, considera que la estructura está completa.

Cuando escriba una instrucción SQL que contenga un nombre de variable de host cualificado (quizás para identificar un campo dentro de una estructura), utilice el nombre de la estructura seguido de un punto y el nombre del campo. Por ejemplo, para la estructura B que contiene el campo C1, especifique B . C1 en lugar de C1 OF B o C1 IN B.

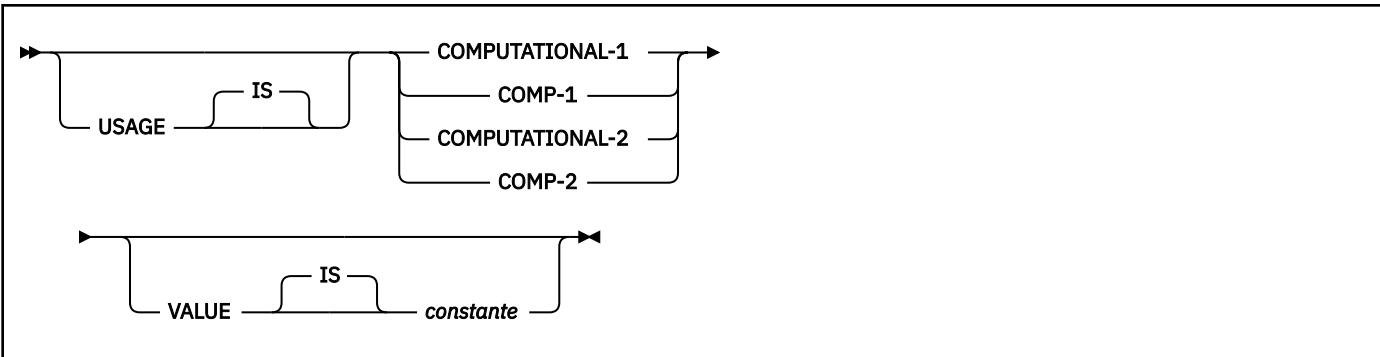
Estructuras host

El siguiente diagrama muestra la sintaxis para declarar estructuras de host.



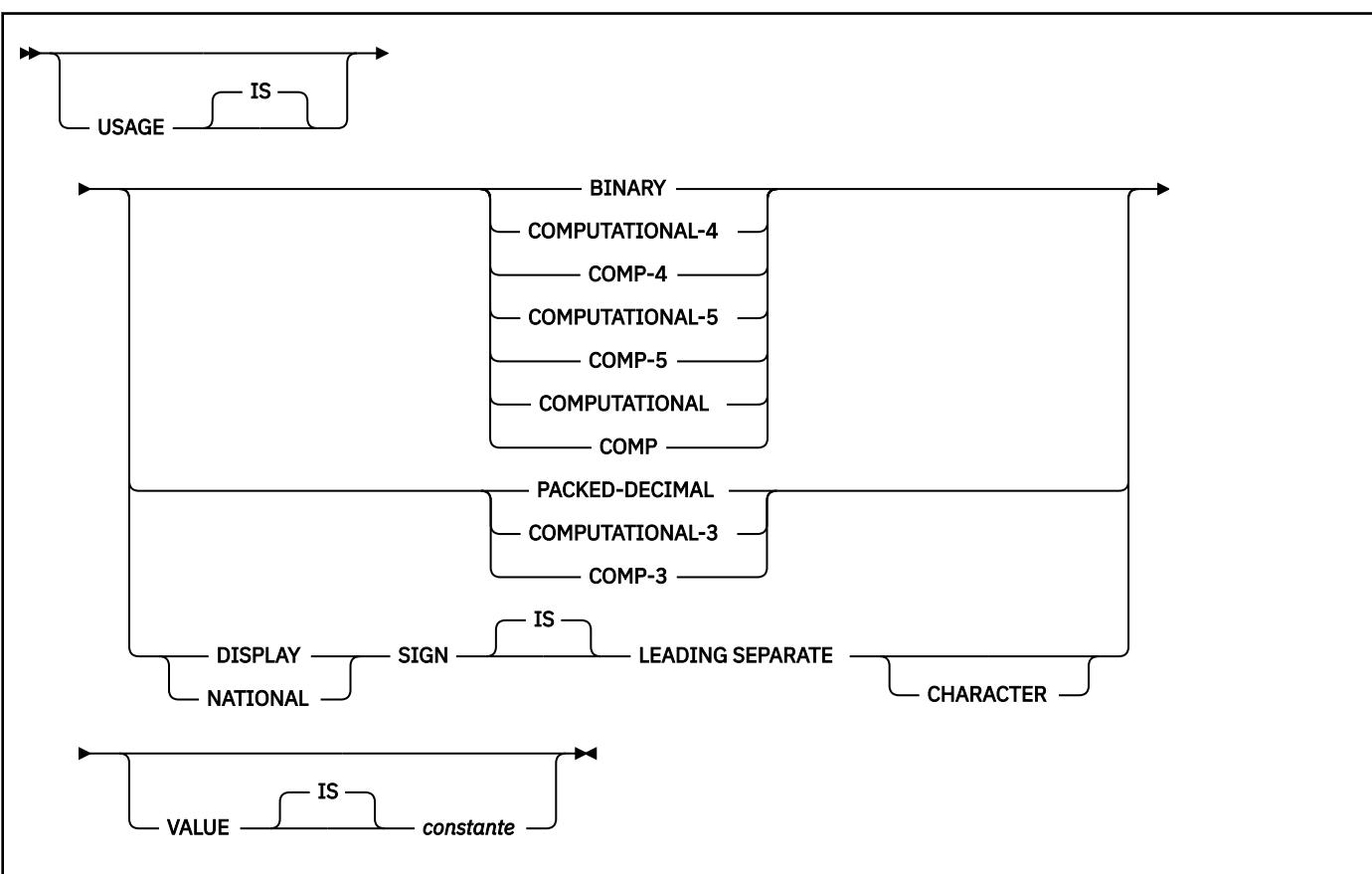
Elementos de uso numérico

El siguiente diagrama muestra la sintaxis de los elementos de uso numérico que se utilizan en las declaraciones de estructuras de host.



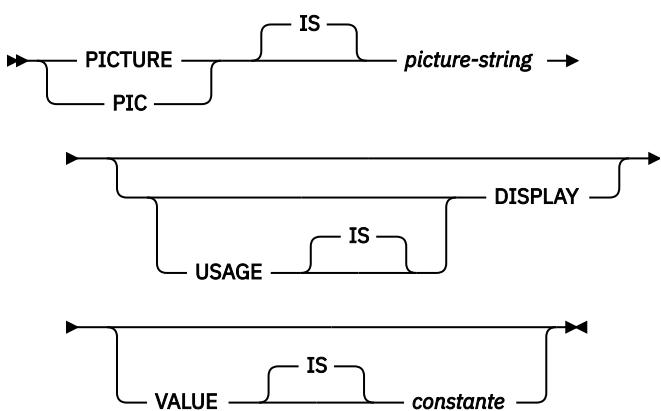
Elementos de uso de números enteros y decimales

El siguiente diagrama muestra la sintaxis para los elementos de uso entero y decimal que se utilizan dentro de las declaraciones de estructuras de host.



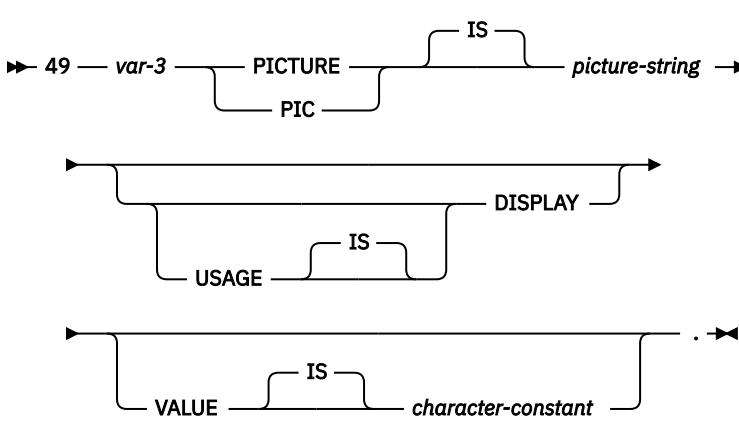
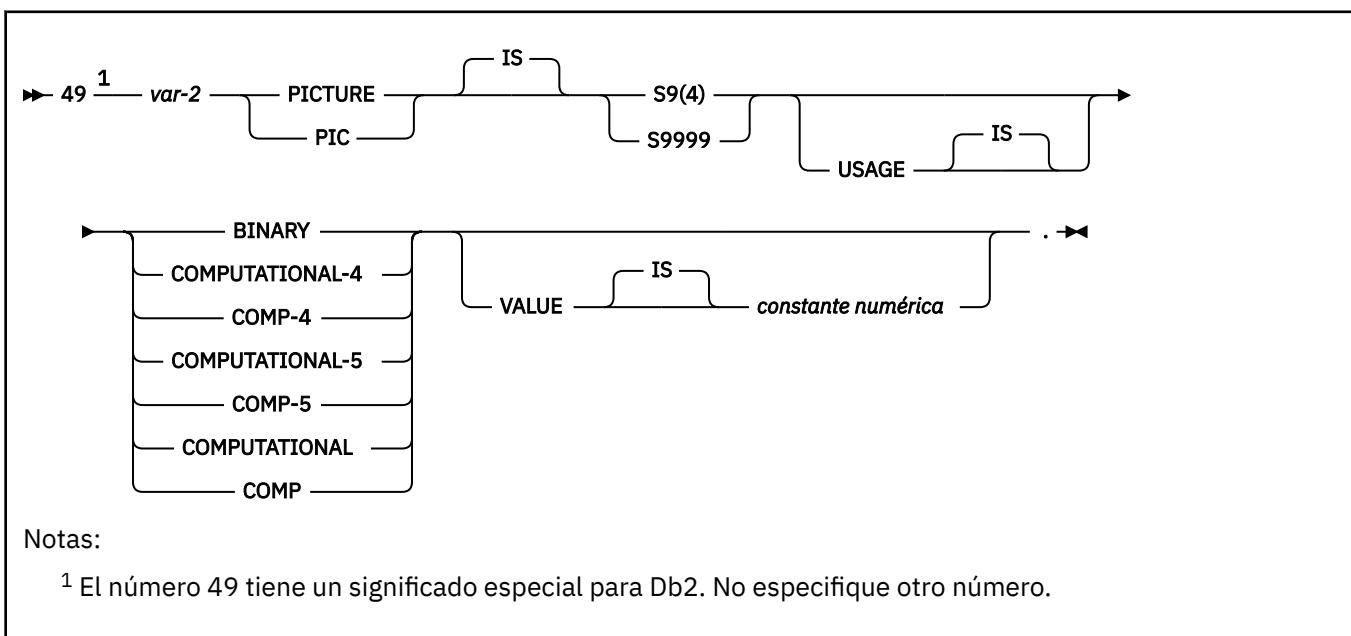
Variables internas CHAR

El siguiente diagrama muestra la sintaxis de las variables internas CHAR que se utilizan en las declaraciones de estructuras de host.



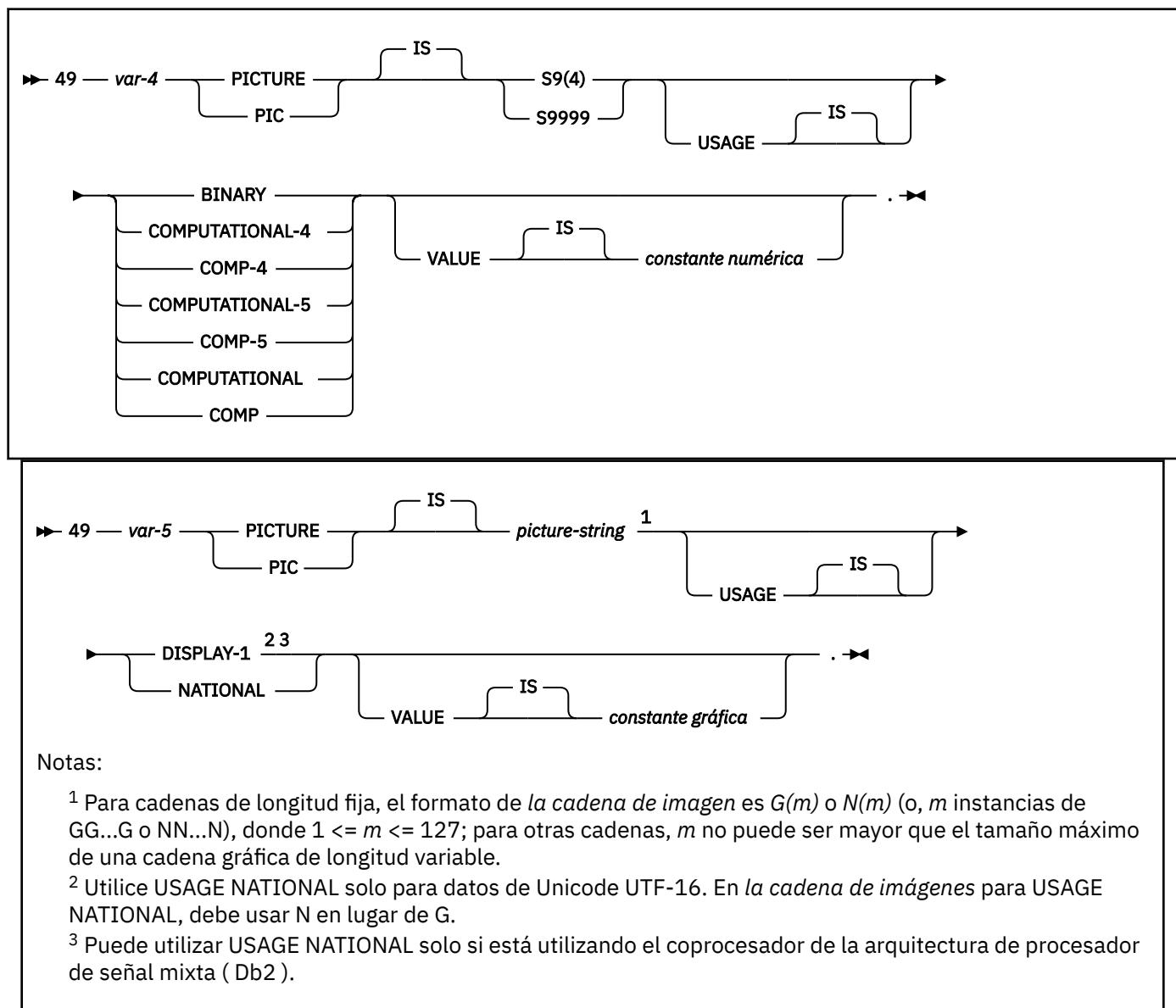
Variables internas VARCHAR

Los siguientes diagramas muestran la sintaxis de las variables internas VARCHAR que se utilizan en las declaraciones de estructuras de host.



Variables internas VARGRAPHIC

Los siguientes diagramas muestran la sintaxis de las variables internas VARGRAPHIC que se utilizan en las declaraciones de estructuras de host.



Notas:

¹ Para cadenas de longitud fija, el formato de *la cadena de imagen* es G(*m*) o N(*m*) (*o, m* instancias de GG...G o NN...N), donde $1 \leq m \leq 127$; para otras cadenas, *m* no puede ser mayor que el tamaño máximo de una cadena gráfica de longitud variable.

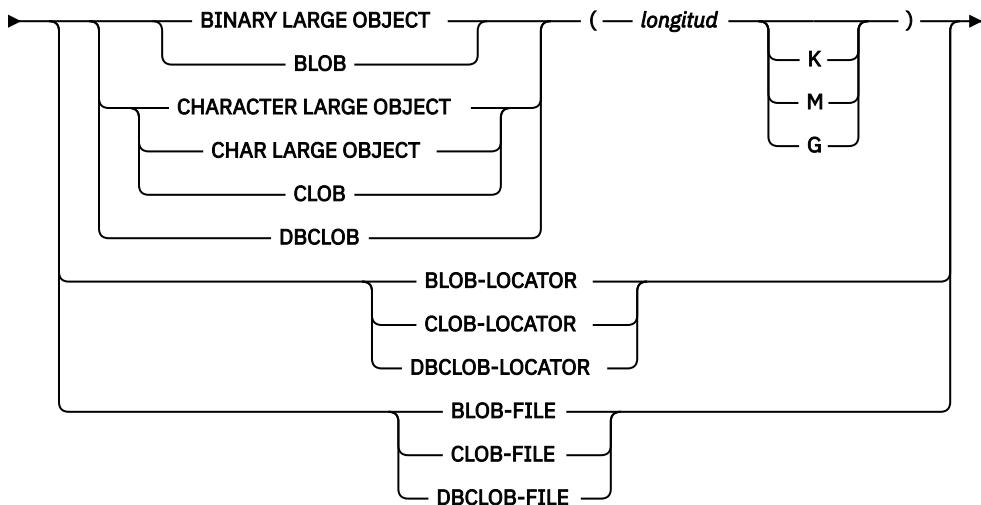
² Utilice USAGE NATIONAL solo para datos de Unicode UTF-16. En *la cadena de imágenes* para USAGE NATIONAL, debe usar N en lugar de G.

³ Puede utilizar USAGE NATIONAL solo si está utilizando el coprocesador de la arquitectura de procesador de señal mixta (Db2).

Variables LOB, localizadores y variables de referencia de archivos

El siguiente diagrama muestra la sintaxis de las variables LOB, localizadores y variables de referencia de archivos que se utilizan en las declaraciones de estructuras de host.

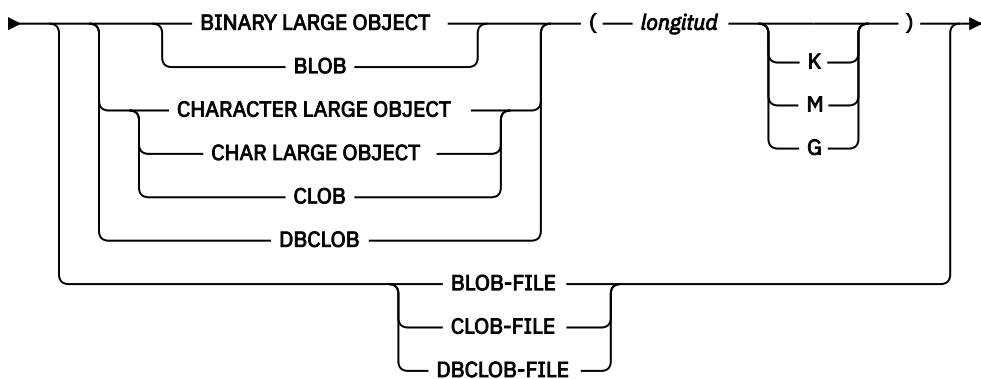
► SQL TYPE IS →



Variables LOB y variables de referencia de archivo para datos XML

El siguiente diagrama muestra la sintaxis de las variables LOB y las variables de referencia de archivos que se utilizan en las declaraciones de estructuras de host para XML.

► SQL TYPE IS XML AS →



Ejemplo

En el siguiente ejemplo, B es el nombre de una estructura de host que contiene los elementos elementales C1 y C2.

```
01 A
 02 B
    03 C1 PICTURE ...
    03 C2 PICTURE ...
```

Para hacer referencia al campo " C1 " en una instrucción SQL, especifique " B.C1".

Conceptos relacionados

[Estructuras host](#)

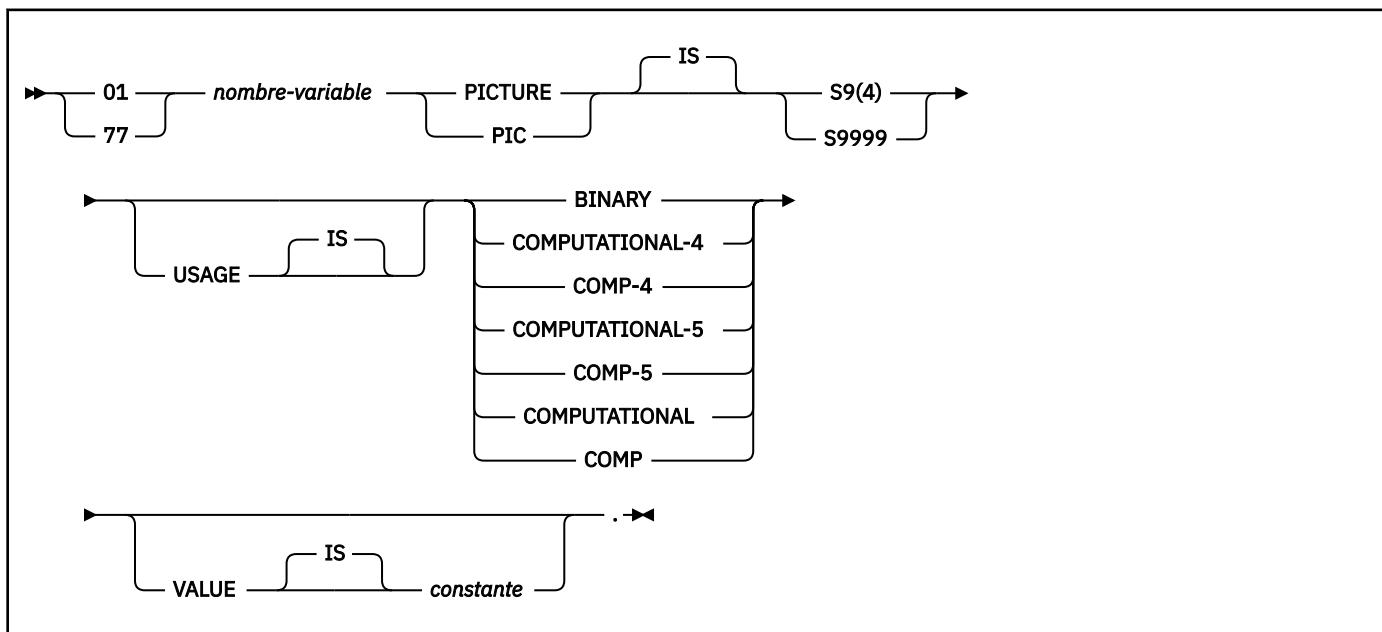
Utilice las estructuras host para pasar un grupo de variables host entre Db2 y su aplicación.

Variables indicadoras, matrices de indicador y matrices de indicador de estructura de host en COBOL

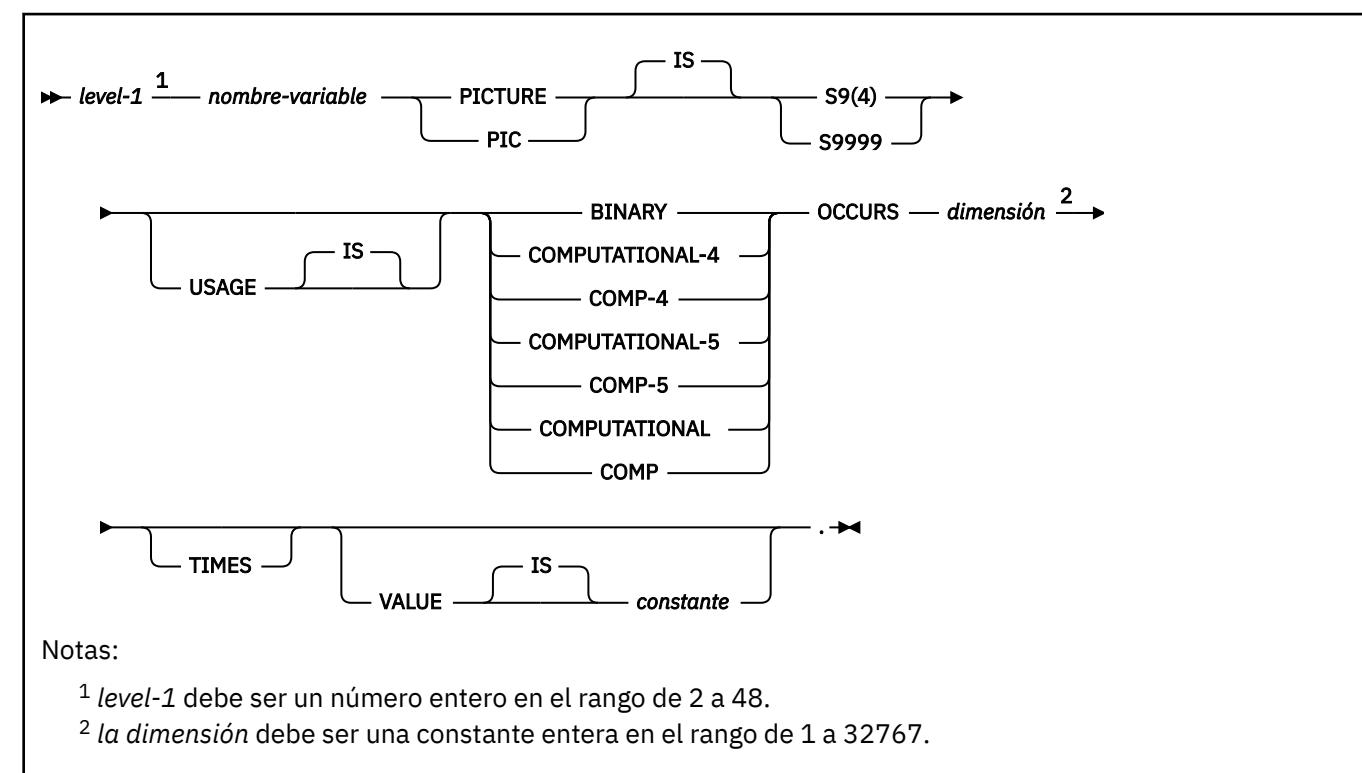
Una variable de indicador COBOL es un entero binario de 2 bytes. Una matriz de variables de indicador COBOL es una matriz en la que cada elemento se declara como un entero binario de 2 bytes. Puede utilizar matrices de variables de indicador para dar soporte a estructuras de host COBOL.

Las variables de indicador se declaran de la misma manera que las variables de host.

El siguiente diagrama muestra la sintaxis para declarar una variable indicadora en COBOL.



El siguiente diagrama muestra la sintaxis para declarar una matriz de indicadores en COBOL.



Notas:

¹ level-1 debe ser un número entero en el rango de 2 a 48.

² la dimensión debe ser una constante entera en el rango de 1 a 32767.

ejemplos

Ejemplo 1:

El siguiente ejemplo muestra las declaraciones de tres variables, con una declaración de variable indicadora para cada variable anfitriona.

```
77 DAYVAR      PIC S9(4) BINARY.  
77 BGNVAR      PIC X(8).  
77 ENDVAR      PIC X(8).  
77 DAYVAR-IND PIC S9(4) BINARY.  
77 BGNVAR-IND PIC S9(4) BINARY.  
77 ENDVAR-IND PIC S9(4) BINARY.
```

La siguiente sentencia FETCH recupera valores de una tabla en las tres variables de host. Si el valor de una columna de la tabla es nulo, Db2 establece la variable indicadora correspondiente en -1.

```
EXEC SQL FETCH CLS_CURSOR INTO  
      :DAYVAR:DAYVAR-IND,  
      :BGNVAR:BGNVAR-IND,  
      :ENDVAR:ENDVAR-IND  
END-EXEC .
```

Ejemplo 2:

El siguiente ejemplo muestra una declaración de una estructura anfitriona y una estructura correspondiente que contiene una matriz de indicadores con el mismo número de elementos que el número de variables en la estructura anfitriona.

```
01 CLS.  
  10 DAYVAR      PIC S9(4) BINARY.  
  10 BGNVAR      PIC X(8).  
  10 ENDVAR      PIC X(8).  
01 CLS-IND-STRUCT.  
  02 CLS-IND    PIC S9(4) BINARY OCCURS 3 TIMES.
```

La siguiente sentencia FETCH recupera valores de una tabla en la estructura host. Si un valor de tabla es nulo, Db2 establece el elemento de matriz de indicadores correspondiente en -1.

```
EXEC SQL FETCH CLS_CURSOR INTO  
      :CLS:CLS-IND  
END-EXEC .
```

Conceptos relacionados

Variables de indicación, matrices y estructuras

Una variable de indicación se asocia con una variable host concreta. Cada variable de indicación contiene un valor entero pequeño que indica alguna información sobre la variable host asociada. Las estructuras y matrices de indicador tienen el mismo propósito para las estructuras y matrices de variables de host.

Tareas relacionadas

Inserción de valores nulos en columnas utilizando las matrices o variables indicadoras

Si necesita insertar valores nulos en una columna, utilizar una matriz o variable indicadora es una forma fácil de hacerlo. Una variable o matriz de indicador está asociada a una variable de host o matriz de variables de lenguaje de host determinada.

Control del CCSID para variables de lenguaje principal de COBOL

El establecimiento del CCSID para variables de lenguaje principal de COBOL es ligeramente distinto del proceso para otros lenguajes de sistema principal. En COBOL, existen diversos valores que afectan al CCSID.

Antes de empezar

Esta tarea se aplica a los programas que utilizan IBM Enterprise COBOL para z/OS y el coprocesador Db2 .

Procedimiento

Para controlar el CCSID para variables de host COBOL, utilice uno o más de los siguientes elementos:

El tipo de datos NACIONAL

Utilice este tipo de datos para declarar valores Unicode en el formato CCSID (UTF-16, Identificador de caracteres de sistema común).

Si declara una variable de host HV1 como USAGE NATIONAL, Db2 siempre maneja HV1 como si hubiera utilizado la siguiente instrucción DECLARE VARIABLE:

```
DECLARE :HV1 VARIABLE CCSID 1200
```

La opción del compilador COBOL CODEPAGE

Utilice esta opción para especificar el CCSID EBCDIC predeterminado de los elementos de datos de caracteres.

La opción del compilador SQLCCSID

Utilice esta opción para controlar si la opción del compilador CODEPAGE influye en el procesamiento de variables de host SQL en sus programas COBOL (disponible en Enterprise COBOL V3R4, o posterior).

Cuando se especifica la opción del compilador SQLCCSID, el coprocesador COBOL Db2 , utiliza el CCSID que se especifica en la opción del compilador CODEPAGE. Todas las variables host de tipo de datos de caracteres, excepto NATIONAL, se especifican con ese CCSID a menos que se anulen explícitamente mediante una instrucción DECLARE VARIABLE.

Cuando se especifica la opción de compilador NOSQLCCSID, el CCSID que se especifica en la opción de compilador CODEPAGE se utiliza para procesar únicamente sentencias COBOL dentro del programa COBOL. Que CCSID no se utilice para el procesamiento de variables de host en sentencias SQL. Db2 utiliza los CCSID que se especifican a través de mecanismos de Db2 y predetermina como codificaciones de valor de datos de variables de host.

La instrucción DECLARE VARIABLE.

Esta declaración establece explícitamente el CCSID para variables de host individuales.

Ejemplo

Suponga que la opción del compilador COBOL SQLCCSID está especificada y que la opción del compilador COBOL CODEPAGE está especificada como CODEPAGE(1141). El siguiente código muestra cómo puede controlar el CCSID:

```
DATA DIVISION.  
 01 HV1 PIC N(10) USAGE NATIONAL.  
 01 HV2 PIC X(20) USAGE DISPLAY.  
 01 HV3 PIC X(30) USAGE DISPLAY.  
 .  
 EXEC SQL  
   DECLARE :HV3 VARIABLE CCSID 1047  
 END-EXEC.  
 .  
 PROCEDURE DIVISION.  
 .  
 EXEC SQL  
   SELECT C1, C2, C3 INTO :HV1, :HV2, :HV3 FROM T1  
 END-EXEC.
```

Cada una de las variables de host tiene los siguientes CCSID:

HV1

1200

HV2

1141

HV3

1047

Supongamos que se especifica la opción del compilador COBOL NOSQLCCSID, la opción del compilador COBOL CODEPAGE se especifica como CODEPAGE(1141), y el CCSID de un solo byte predeterminado de Db2 se establece en 37. En este caso, cada una de las variables de host de este ejemplo tiene los siguientes CCSID:

HV1
1200

HV2
37

HV3
1047

Referencia relacionada

Variables de host en COBOL

En programas COBOL, puede especificar variables host numéricas, de caracteres, gráficas, binarias, LOB, XML y ROWID. También puede especificar conjuntos de resultados y localizadores de tabla y LOB, y variables de referencia de archivos LOB y XML.

[Opciones de compilador \(COBOL\) \(Enterprise COBOL for z/OS Programming Guide\)](#)

SQL equivalente y tipos de datos COBOL

Cuando declara variables host en los programas COBOL, el precompilador utiliza tipos de datos SQL equivalentes. Cuando recupere datos de un tipo de datos SQL concreto en una variable host, tendrá que asegurarse de que la variable host sea de un tipo de datos equivalente.

La siguiente tabla describe el tipo de datos SQL y los valores base SQLTYPE y SQLLEN que el precompilador utiliza para las variables de host en las sentencias SQL.

Tabla 108. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas COBOL

Tipo de datos de variable de host COBOL	SQLTYPE de la variable de host1	SQLLEN de la variable de host	Tipos de datos SQL
COMP-1	480	4	REAL o FLOAT(n) 1≤n≤21
COMP-2	480	8	DOBLE PRECISIÓN, o FLOTANTE (n) 22≤n≤53
S9(i)V9(d) COMP-3 o S9(i)V9(d) DECIMAL EMPAQUETADO	484	$i + d$ en el byte 1, d en el byte 2	DECIMAL($i + d, d$) o NUMÉRICO($i + d, d$)
S9(i)V9(d) MOSTRAR LETRERO PRINCIPAL SEPARADO	504	$i + d$ en el byte 1, d en el byte 2	No existe un equivalente exacto. Utilice DECIMAL($i + d, d$) o NUMÉRICO($i + d, d$)
S9(i)V9(d) SEÑAL NACIONAL PRINCIPAL SEPARADA	504	$i + d$ en el byte 1, d en el byte 2	No existe un equivalente exacto. Utilice DECIMAL($i + d, d$) o NUMÉRICO($i + d, d$)
S9(4) COMP-4, S9(4) COMP-5, S9(4) COMP, o S9(4) BINARIO	500	2	SMALLINT
S9(9) COMP-4, S9(9) COMP-5, S9(9) COMP, o S9(9) BINARIO	496	4	ENTERO
S9(18) COMP-4, S9(18) COMP-5, S9(18) COMP, o S9(18) BINARIO	492	8	BIGINT

Tabla 108. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas COBOL (continuación)

Tipo de datos de variable de host COBOL	SQLTYPE de la variable de host1	SQLLEN de la variable de host	Tipos de datos SQL
Datos de caracteres de longitud fija	452	<i>n</i>	CHAR (<i>n</i>)
Datos de caracteres de longitud variable $1 \leq n \leq 255$	448	<i>n</i>	VARCHAR(<i>n</i>)
Datos de caracteres de longitud variable $m > 255$	456	<i>M</i>	VAR CHAR(<i>m</i>)
Datos gráficos de longitud fija	468	<i>M</i>	GRAPHIC(<i>m</i>)
Datos gráficos de longitud variable $1 \leq m \leq 127$	464	<i>M</i>	VARGRAPHIC(<i>m</i>)
Datos gráficos de longitud variable $m > 127$	472	<i>M</i>	VARGRAPHIC(<i>m</i>)
SQL TYPE es BINARY(<i>n</i>), $1 \leq n \leq 255$	912	<i>n</i>	BINARY(<i>n</i>)
SQL TYPE es VARBINARY(<i>n</i>), $1 \leq n \leq 32704$	908	<i>n</i>	VARBINARY(<i>n</i>)
TIPO SQL ES RESULT-SET-LOCATOR	972	4	Localizador de conjunto de resultados2
SQL TYPE IS TABLE LIKE <i>nombre-tabla</i> AS LOCATOR	976	4	Localizador de mesas2
TIPO SQL ES BLOB-LOCATOR	960	4	Localizador BLOB2
TIPO SQL ES CLOB-LOCATOR	964	4	Localizador CLOB2
TIPO SQL ES DBCLOB-LOCATOR	968	4	Localizador DBCLOB2
USO ES SQL TIPO ES BLOB(<i>i</i>) $1 \leq i \leq 2147483647$	404	<i>I</i>	BL OB(<i>i</i>)
USO ES SQL TIPO ES CLOB(<i>i</i>) $1 \leq i \leq 2147483647$	408	<i>I</i>	CLOB(<i>i</i>)
EL USO ES SQL TIPO ES DBCLOB(<i>m</i>) $1 \leq m \leq 1073741823$ ³	412	<i>I</i>	DBCLOB(<i>m</i>) ³
SQL TYPE IS XML AS BLOB(<i>i</i>)	404	0	XML
SQL TYPE IS XML AS CLOB(<i>i</i>)	408	0	XML
SQL TYPE IS XML AS DBCLOB(<i>i</i>)	412	0	XML
TIPO SQL ES BLOB-FILE	916/917	267	Referencia de archivo BLOB2
TIPO SQL ES CLOB-FILE	920/921	267	Referencia de archivo CLOB2
TIPO SQL ES DBCLOB-FILE	924/925	267	Referencia de archivo DBCLOB2
SQL TYPE IS XML AS BLOB-FILE	916/917	267	Referencia de archivo XML BLOB2

Tabla 108. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas COBOL (continuación)

Tipo de datos de variable de host COBOL	SQLTYPE de la variable de host ¹	SQLLEN de la variable de host	Tipos de datos SQL
SQL TYPE IS XML AS CLOB-FILE	920/921	267	Referencia de archivo XML CLOB2
SQL TYPE IS XML AS DBCLOB-FILE	924/925	267	Referencia de archivo XML DBCLOB2
EL TIPO SQL ES ROWID	904	40	ROWID

Notas:

1. Si una variable host incluye una variable indicadora, el valor SQLTYPE es el valor SQLTYPE base más 1.
2. No utilice este tipo de datos como tipo de columna.
3. *m* es el número de caracteres de doble byte.

La siguiente tabla muestra las variables de host COBOL equivalentes para cada tipo de datos SQL. Utilice esta tabla para determinar el tipo de datos COBOL para las variables de host que defina para recibir la salida de la base de datos. Por ejemplo, si recupera datos TIMESTAMP, puede definir una variable de cadena de caracteres de longitud fija *de longitud n*

Esta tabla muestra las conversiones directas entre los tipos de datos SQL y los tipos de datos COBOL. Sin embargo, varios tipos de datos SQL son compatibles. Cuando realiza asignaciones o comparaciones de datos que tienen tipos de datos compatibles, Db2 convierte esos tipos de datos compatibles.

Tabla 109. Equivalentes de variables de host COBOL que puede utilizar al recuperar datos de un tipo de datos SQL concreto

Tipos de datos SQL	Variable de host COBOL equivalente	Notas
SMALLINT	S9(4) COMP-4, S9(4) COMP-5, S9(4) COMP, or S9(4) BINARY	
ENTERO	S9(9) COMP-4, S9(9) COMP-5, S9(9) COMP, or S9(9) BINARY	
DECIMAL(<i>p,s</i>) o NUMERIC(<i>p,s</i>)	S9(<i>p-s</i>)V9(<i>s</i>) COMP-3 or S9(<i>p-s</i>)V9(<i>s</i>) PACKED-DECIMAL DISPLAY SIGN LEADING SEPARATE NATIONAL SIGN LEADING SEPARATE	<i>p</i> es precisión; <i>s</i> es escala. $0 \leq s \leq p \leq 31$. Si <i>s=0</i> , utilice S9(<i>p</i>)V o S9(<i>p</i>). Si <i>s=p</i> , utilice SV9(<i>s</i>). Si el compilador COBOL no admite números decimales e 31-digit es, no existe un equivalente exacto. Utiliza COMP-2.
REAL o FLOAT (<i>n</i>)	COMP-1	$1 \leq n \leq 21$
DOBLE PRECISIÓN, DOBLE o FLOAT (<i>n</i>)	COMP-2	$22 \leq n \leq 53$
BIGINT	S9(18) COMP-4, S9(18) COMP-5, S9(18) COMP, o S9(18) BINARIO	

Tabla 109. Equivalentes de variables de host COBOL que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host COBOL equivalente	Notas
CHAR (<i>n</i>)	Cadena de caracteres de longitud fija. Por ejemplo, 01 VAR-NAME PIC X(<i>n</i>).	1≤ <i>n</i> ≤25 5
VARCHAR(<i>n</i>)	Cadena de caracteres de longitud variable. Por ejemplo, 01 VAR-NAME. 49 VAR-LEN PIC S9(4) USAGE BINARY. 49 VAR-TEXT PIC X(<i>n</i>).	Las variables internas deben tener un nivel de 49.
GRAPHIC (<i>n</i>)	Cadena gráfica de longitud fija. Por ejemplo, 01 VAR-NAME PIC G(<i>n</i>) USAGE IS DISPLAY-1.	<i>n</i> se refiere al número de caracteres de doble byte, no al número de bytes. 1≤ <i>n</i> ≤ 127
VARGRAPHIC (<i>n</i>)	Cadena gráfica de longitud variable. Por ejemplo, 01 VAR-NAME. 49 VAR-LEN PIC S9(4) USAGE BINARY. 49 VAR-TEXT PIC G(<i>n</i>) USAGE IS DISPLAY-1.	<i>n</i> se refiere al número de caracteres de doble byte, no al número de bytes. Las variables internas deben tener un nivel de 49.
BINARY(<i>n</i>)	TIPO SQL ES BINARIO(<i>n</i>)	1≤ <i>n</i> ≤25 5
VARBINARY(<i>n</i>)	TIPO SQL ES VARBINARIO(<i>n</i>)	1≤ <i>n</i> ≤32704
FECHA	Cadena de caracteres de longitud fija de longitud <i>n</i> . Por ejemplo, 01 VAR-NAME PIC X(<i>n</i>).	Si está utilizando una rutina de salida de fecha, <i>esta</i> la determina dicha rutina. De lo contrario, <i>debo</i> tener al menos 10 años.
HORA	Cadena de caracteres de longitud fija de longitud <i>n</i> . Por ejemplo, 01 VAR-NAME PIC X(<i>n</i>).	Si está utilizando una rutina de salida de tiempo, <i>n</i> está determinado por esa rutina. De lo contrario, <i>n</i> debe ser al menos 6; para incluir segundos, <i>n</i> debe ser al menos 8.
TIMESTAMP	Cadena de caracteres de longitud fija de longitud <i>n</i> . Por ejemplo, 01 VAR-NAME PIC X(<i>n</i>).	<i>n</i> debe tener al menos 19 años. Para incluir microsegundos, <i>n</i> debe ser 26; si <i>n</i> es menor que 26, se produce un truncamiento en la parte de los microsegundos.
TIMESTAMP(0)	Cadena de caracteres de longitud fija de longitud <i>n</i> . Por ejemplo, 01 VAR-NAME PIC X(<i>n</i>).	<i>n</i> debe tener al menos 19 años.

Tabla 109. Equivalentes de variables de host COBOL que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host COBOL equivalente	Notas
MARCA DE TIEMPO (p) p > 0	Cadena de caracteres de longitud fija de longitud <i>n</i> . Por ejemplo, 01 VAR-NAME PIC X(<i>n</i>).).	<i>n</i> debe tener al menos 19 años. Para incluir fracciones de segundo, <i>n</i> debe ser 20+x, donde <i>x</i> es el número de fracciones de segundo que se van a incluir; si <i>x</i> es menor que <i>p</i> , se produce un truncamiento en la parte de las fracciones de segundo.
TIMESTAMP (0) WITH TIME ZONE	Cadena de caracteres de longitud variable. Por ejemplo, 01 VAR-NAME. 49 VAR-LEN PIC S9(4) USAGE BINARY. 49 VAR-TEXT PIC X(<i>n</i>).	Las variables internas deben tener un nivel de 49. <i>n</i> debe ser al menos 25.
FECHA Y HORA (<i>p</i>) CON ZONA HORARIA	Cadena de caracteres de longitud variable. Por ejemplo, 01 VAR-NAME. 49 VAR-LEN PIC S9(4) USAGE BINARY. 49 VAR-TEXT PIC X(<i>n</i>).	Las variables internas deben tener un nivel de 49. <i>n</i> debe ser al menos 26+ <i>p</i> .
Localizador de conjunto de resultados	SQL TYPE IS RESULT-SET-LOCATOR	Utilice este tipo de datos solo para recibir conjuntos de resultados. No utilice este tipo de datos como tipo de columna.
Localizador de tablas	SQL TYPE IS TABLE LIKE <i>table-name</i> AS LOCATOR	Utilice este tipo de datos solo en una función definida por el usuario o en un procedimiento almacenado para recibir filas de una tabla de transición. No utilice este tipo de datos como tipo de columna.
localizador de BLOB	USAGE IS SQL TYPE IS BLOB-LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas BLOB. No utilice este tipo de datos como tipo de columna.
localizador de CLOB	USAGE IS SQL TYPE IS CLOB-LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas CLOB. No utilice este tipo de datos como tipo de columna.
localizador de DBCLOB	USAGE IS SQL TYPE IS DBCLOB-LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas DBCLOB. No utilice este tipo de datos como tipo de columna.
BL OB(<i>i</i>)	USAGE IS SQL TYPE IS BLOB(<i>i</i>)	1≤ <i>n</i> ≤2147483647
CLOB(<i>i</i>)	USAGE IS SQL TYPE IS CLOB(<i>i</i>)	1≤ <i>n</i> ≤2147483647

Tabla 109. Equivalentes de variables de host COBOL que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host COBOL equivalente	Notas
DBC LOB (i)	USAGE IS SQL TYPE IS DBCLOB(<i>i</i>)	<i>i</i> es el número de caracteres de doble byte. $1 \leq n \leq 1073741823$
XML	SQL TYPE IS XML AS BLOB(<i>i</i>)	$1 \leq n \leq 2147483647$
XML	SQL TYPE IS XML AS CLOB(<i>i</i>)	$1 \leq n \leq 2147483647$
XML	SQL TYPE IS XML AS DBCLOB(<i>i</i>)	<i>i</i> es el número de caracteres de doble byte. $1 \leq n \leq 1073741823$
Referencia de archivo BLOB	USAGE IS SQL TYPE IS BLOB-FILE	Utilice este tipo de datos solo para manipular datos en columnas BLOB. No utilice este tipo de datos como tipo de columna.
Referencia de archivo CLOB	USAGE IS SQL TYPE IS CLOB-FILE	Utilice este tipo de datos solo para manipular datos en columnas CLOB. No utilice este tipo de datos como tipo de columna.
Referencia del archivo DBCLOB	USAGE IS SQL TYPE IS DBCLOB-FILE	Utilice este tipo de datos solo para manipular datos en columnas DBCLOB. No utilice este tipo de datos como tipo de columna.
Referencia de archivo XML BLOB	SQL TYPE IS XML AS BLOB-FILE	Utilice este tipo de datos solo para manipular datos XML como archivos BLOB. No utilice este tipo de datos como tipo de columna.
Referencia de archivo XML CLOB	SQL TYPE IS XML AS CLOB-FILE	Utilice este tipo de datos solo para manipular datos XML como archivos CLOB. No utilice este tipo de datos como tipo de columna.
Referencia de archivo XML DBCLOB	SQL TYPE IS XML AS DBCLOB-FILE	Utilice este tipo de datos solo para manipular datos XML como archivos DBCLOB. No utilice este tipo de datos como tipo de columna.
ROWID	SQL TYPE IS ROWID	

La siguiente tabla muestra las definiciones del lenguaje COBOL que se deben utilizar en los procedimientos almacenados COBOL y en las funciones definidas por el usuario, cuando los tipos de datos de los parámetros en las definiciones de rutina son LOB, ROWID o localizadores. Para otros tipos de datos de parámetros, las definiciones del lenguaje COBOL son las mismas que las de la sección anterior (Tabla 109 en la página 716).

Tabla 110. Declaraciones COBOL equivalentes para LOB, ROWID y localizadores en definiciones de rutinas definidas por el usuario

Tipo de datos SQL en la definición	Declaración COBOL
localizador de tablas	01 var PIC S9(9) COMP-5
localizador de BLOB	
localizador de CLOB	
localizador de DBCLOB	
BL OB (n)	01 var. 49 var-LENGTH PIC S9(9) COMP-5. 49 var-DATA PIC X(n).
CLOB(n)	01 var. 49 var-LENGTH PIC S9(9) COMP-5. 49 var-DATA PIC X(n).
DBCLOB (n)	01 var. 49 var-LENGTH PIC S9(9) COMP-5. 49 var-DATA PIC G(n) DISPLAY-1.
ROWID	01 var. 49 var-LEN PIC S9(4) COMP-5. 49 var-TEXT PIC X(40).

Conceptos relacionados

[Compatibilidad de SQL y tipos de datos de lenguaje](#)

Los tipos de datos de variable host que se utilizan en sentencias SQL deben ser compatibles con los tipos de datos de las columnas con las que tiene intención de utilizarlos.

[Declaraciones de variable de referencia de archivo LOB, variable host LOB y localizador LOB](#)

Cuando escribe aplicaciones para manipular datos LOB, necesita declarar variables hosto para contener los datos LOB o el localizador LOB. De lo contrario, debe declarar variables de referencia de archivo LOB para apuntar a datos LOB.

[Tipos de datos de variable host para datos XML en aplicaciones de SQL incorporado \(Db2 Programming for XML\)](#)

Extensiones orientadas a objetos en COBOL

Cuando utiliza extensiones orientadas a objetos en una aplicación COBOL, debe considerar dónde colocar sentencias SQL, SQLCA, SQLDA y declaraciones de variables host. También debe considerar las reglas para las variables host.

Dónde colocar las sentencias SQL en su aplicación : Un conjunto de datos fuente o miembro COBOL puede contener los siguientes elementos:

- Múltiples programas
- Múltiples definiciones de clase, cada una de las cuales contiene múltiples métodos

Puede poner instrucciones SQL solo en el primer programa o clase del conjunto de datos fuente o miembro. Sin embargo, puede poner instrucciones SQL en varios métodos dentro de una clase. Si una aplicación consta de varios conjuntos de datos o miembros, cada uno de los conjuntos de datos o miembros puede contener sentencias SQL.

Dónde colocar las declaraciones de variables SQLCA, SQLDA y host : Puede colocar las declaraciones de variables SQLCA, SQLDA y SQL host en la SECCIÓN WORKING-STORAGE de un programa, clase o método. Un SQLCA o SQLDA en una clase WORKING-STORAGE SECTION es global para todos los métodos de la clase. Un SQLCA o SQLDA en un método WORKING-STORAGE SECTION es local solo para ese método.

Si una clase y un método dentro de la clase contienen un SQLCA o un SQLDA, el método utiliza el SQLCA o el SQLDA que sea local.

Reglas para variables de host : Puede declarar variables COBOL que se utilizan como variables de host en la WORKING-STORAGE SECTION o LINKAGE-SECTION de un programa, clase o método. También puede declarar variables de host en la SECCIÓN LOCAL-STORAGE de un método. El alcance de una variable anfitriona es el método, la clase o el programa dentro del cual se define.

Gestión de códigos de error de SQL en aplicaciones Cobol

Las aplicaciones Cobol pueden solicitar más información sobre los códigos de error de SQL utilizando la subrutina DSNTIAR o emitiendo una sentencia GET DIAGNOSTICS.

Procedimiento

Para solicitar más información sobre errores SQL de programas Cobol, utilice los siguientes métodos:

- Puede utilizar el campo de elemento de condición MESSAGE_TEXT de la instrucción GET DIAGNOSTICS para convertir un código de retorno SQL en un mensaje de texto. Los programas que requieren un soporte de mensajes token largos deben codificar la instrucción GET DIAGNOSTICS en lugar de DSNTIAR.
- Puede utilizar la subrutina DSNTIAR para convertir un código de retorno SQL en un mensaje de texto. DSNTIAR toma datos de SQLCA, los formatea en un mensaje y coloca el resultado en un área de salida de mensajes que usted proporciona en su programa de aplicación.

Sintaxis DSNTIAR

DSNTIAR tiene la siguiente sintaxis:

```
CALL 'DSNTIAR' USING sqlca message lrecl.
```

Parámetros DSNTIAR

Los parámetros DSNTIAR tienen los siguientes significados:

sqlca

Un área de comunicación SQL.

mensaje

Un área de salida, en formato VARCHAR, en la que DSNTIAR coloca el texto del mensaje. La primera semicifra contiene la longitud del área restante; su valor mínimo es 240.

Las líneas de salida de texto, cada una con la longitud especificada en *lrecl*, se colocan en esta área. Por ejemplo, podría especificar el formato del área de salida como:

```
01  ERROR-MESSAGE.
    02  ERROR-LEN    PIC S9(4)  COMP-5 VALUE +1320.
    02  ERROR-TEXT   PIC X(132) OCCURS 10 TIMES
                                INDEXED BY ERROR-INDEX.
77  ERROR-TEXT-LEN      PIC S9(9)  COMP-5 VALUE +132.
:
CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
```

donde ERROR-MESSAGE es el nombre del área de salida de mensajes que contiene 10 líneas de longitud 132 cada una, y ERROR-TEXT-LEN es la longitud de cada línea.

lrecl

Una palabra completa que contiene la longitud de registro lógica de los mensajes de salida, en el rango de 72 a 240.

Un ejemplo de llamada a DSNTIAR desde una aplicación aparece en el programa ensamblador de ejemplo Db2 DSN8BC3, que se encuentra en la biblioteca DSN8C10.

- Si su CICS solicitud requiere CICS almacenamiento, debe utilizar la subrutina DSNTIAC en lugar de DSNTIAR.
- Si llama a DSNTIAR dinámicamente desde un CICS Programa de aplicación COBOL, asegúrese de hacer lo siguiente:

- Compilar la aplicación COBOL con la opción NODYNAM.
- Defina DSNTIAR en el CSD.

Sintaxis DSNTIAC

Si su CICS solicitud requiere CICS almacenamiento, debe utilizar la subrutina DSNTIAC en lugar de DSNTIAR. DSNTIAC tiene la siguiente sintaxis:

```
CALL 'DSNTIAC' USING eib commarea sqlca msg lrecl.
```

Parámetros DSNTIAC

DSNTIAC tiene parámetros adicionales, que debe utilizar para llamadas a rutinas que utilizan CICS comandos.

eib

Bloque de interfaz EXEC

área de clientes

área de comunicación

Para obtener más información sobre estos parámetros, consulte la guía de programación de aplicaciones correspondiente para CICS. Las descripciones de los parámetros restantes son las mismas que las de DSNTIAR. Tanto DSNTIAC como DSNTIAR formatean el SQLCA de la misma manera.

Debe definir DSNTIA1 en el CSD. Si carga DSNTIAR o DSNTIAC, también debe definirlos en el CSD. Para ver un ejemplo de declaraciones de generación de entradas CSD para su uso con DSNTIAC, consulte el trabajo DSNTEJ5A.

El código fuente ensamblador para DSNTIAC y el trabajo DSNTEJ5A, que ensambla y edita enlaces DSNTIAC, están en el *prefijo* del conjunto de datos.SSDNSAMP.

Tareas relacionadas

[Manejo de códigos de error de SQL](#)

Los programas de aplicación pueden solicitar más información sobre los códigos de error SQL en Db2.

Referencia relacionada

[GET DIAGNOSTICS declaración \(Db2 SQL \)](#)

Aplicaciones Fortran que emiten sentencias SQL

Puede codificar sentencias SQL en un programa Fortran siempre que pueda colocar sentencias ejecutables. Si la sentencia SQL está dentro de una sentencia IF, el precompilador genera cualquier sentencia THEN y END IF necesaria.

Fortran las declaraciones de origen deben ser registros de 80 bytes de longitud fija. El precompilador de Db2 no admite la entrada de código fuente de forma libre.

Cada instrucción SQL en un programa de Fortran debe comenzar con EXEC SQL. Las palabras clave EXEC y SQL deben aparecer en una línea, pero el resto de la instrucción puede aparecer en líneas posteriores.

Podría codificar la instrucción UPDATE en un programa de lenguaje de programación (Fortran) de la siguiente manera:

```
EXEC SQL
C   UPDATE DSN8C10.DEPT
C   SET MGRNO = :MGRNUM
C   WHERE DEPTNO = :INTDEPT
```

No puede seguir una instrucción SQL con otra instrucción SQL o una instrucción Fortran en la misma línea.

Fortran no requiere espacios en blanco para delimitar palabras dentro de una sentencia, pero el lenguaje SQL sí los requiere. Las reglas para SQL incrustado siguen las reglas para la sintaxis SQL, que requieren que utilices uno o más espacios en blanco como delimitador.

Comentarios

Puede incluir líneas de comentario de tipo " Fortran " dentro de las sentencias SQL incrustadas siempre que pueda utilizar un espacio en blanco, excepto entre las palabras clave EXEC y SQL. Puede incluir comentarios SQL en cualquier instrucción SQL incrustada. Para obtener más información, consulte [Comentarios SQL \(Db2 SQL\)](#).

El precompilador de Db2 no admite el signo de exclamación (!) como carácter de reconocimiento de comentarios en los programas de Fortran.

Continuación para instrucciones SQL

Las reglas de continuación de línea para las sentencias SQL son las mismas que para las sentencias Fortran, excepto que debe especificar EXEC SQL en una línea. Los ejemplos SQL de este tema tienen Cs en la sexta columna para indicar que son continuaciones de EXEC SQL.

Delimitadores en Fortran

Delimite una instrucción SQL en su programa de Fortran con la palabra clave inicial EXEC SQL y un final de línea o final de la última línea continuada.

Declaración de tablas y vistas

Su programa de Fortran también debe incluir la instrucción DECLARE TABLE para describir cada tabla y ver los accesos del programa.

SQL dinámico en un programa de Fortran

En general, los programas de Fortran s pueden manejar fácilmente sentencias SQL dinámicas. Las sentencias SELECT pueden manejarse si los tipos de datos y el número de campos devueltos son fijos. Si desea utilizar sentencias SELECT de lista variable, necesita utilizar un SQLDA, como se describe en "[Definición de áreas de descriptor de SQL \(SQLDA\)](#)" en la página 502.

Puede utilizar una variable de carácter e Fortran e en las sentencias PREPARE y EXECUTE IMMEDIATE, incluso si es de longitud fija.

Incluir código

Para incluir sentencias SQL o declaraciones de variables de host de un Fortran e de un miembro de un conjunto de datos particionado, utilice la siguiente sentencia SQL en el código fuente donde desee incluir las sentencias:

```
EXEC SQL INCLUDE member-name
```

No se pueden anidar sentencias SQL INCLUDE. No puede utilizar la directiva del compilador INCLUDE (Fortran) para incluir sentencias SQL o declaraciones de variables de host (Fortran).

Márgenes

Codifique las sentencias SQL en las columnas 7-72, ambas inclusive. Si EXEC SQL comienza antes del margen izquierdo especificado, el precompilador de instrucciones SQL (Db2) no reconoce la instrucción SQL.

Nombres

Puede utilizar cualquier nombre válido de Fortran para una variable de host. No utilice nombres de entrada externos que empiecen por «DSN» ni nombres de variables de host que empiecen por «SQL». Estos nombres están reservados para Db2.

No utilice la palabra DEBUG, excepto cuando defina un paquete DEBUG de e Fortran. No utilice las palabras FUNCIÓN, IMPLÍCITA, PROGRAMA y SUBRUTINA para definir variables.

Números de secuencia

Las sentencias fuente que genera el precompilador de Db2 no incluyen números de secuencia.

Etiquetas de extracto

Puede especificar números de instrucciones para instrucciones SQL en las columnas 1 a 5. Sin embargo, durante la preparación del programa, una instrucción SQL etiquetada genera una instrucción CONTINUE (Fortran) con esa etiqueta antes de generar el código que ejecuta la instrucción SQL. Por lo tanto, una instrucción SQL etiquetada nunca debe ser la última instrucción de un bucle DO. Además, no debe etiquetar las sentencias SQL (como INCLUDE y BEGIN DECLARE

SECTION) que se producen antes de la primera sentencia SQL ejecutable, porque podría producirse un error.

WHENEVER, sentencia

El objetivo de la cláusula GOTO en la instrucción SQL WHENEVER debe ser una etiqueta en el código fuente de la instrucción SQL (Fortran) y debe hacer referencia a una instrucción en el mismo subprograma. La instrucción WHENEVER solo se aplica a las instrucciones SQL del mismo subprograma.

Consideraciones especiales sobre el e Fortran

Las siguientes consideraciones se aplican a los programas escritos en lenguaje de programación (Fortran):

- No puede utilizar la instrucción @PROCESS en su código fuente. En su lugar, especifique las opciones del compilador en el campo PARM.
- No puede utilizar la instrucción SQL INCLUDE para incluir las siguientes instrucciones: PROGRAM, SUBROUTINE, BLOCK, FUNCTION o IMPLICIT.

Db2 es compatible con la versión 3, versión 1 (o posterior) de VS Fortran, con las siguientes restricciones:

- La opción paralela no es compatible. Las aplicaciones que contienen instrucciones SQL no deben utilizar el paralelismo de subprocessos (Fortran).
- No puede utilizar el tipo de datos byte en SQL incrustado, porque byte no es un tipo de datos de host reconocible.

Manejo de códigos de error de SQL

Fortran las aplicaciones pueden solicitar más información sobre los errores SQL en Db2. Para obtener más información, consulte “[Gestión de códigos de error de SQL en aplicaciones C y C++](#)” en la página [652](#).

Tareas relacionadas

[Descripción general de las aplicaciones de programación que acceden a datos de Db2 for z/OS](#)
Las aplicaciones que interactúan con Db2, en primer lugar se deben conectar a Db2. Entonces pueden leer, añadir o modificar datos o manipular objetos de Db2.

[Inclusión de SQL dinámico en el programa](#)

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

[Manejo de códigos de error de SQL](#)

Los programas de aplicación pueden solicitar más información sobre los códigos de error SQL en Db2.

[Establecimiento de límites para el uso de recursos de sistema utilizando el recurso de límite de recursos \(Db2 Performance\)](#)

Definición del área de comunicación SQL, SQLSTATE y SQLCODE en Fortran

Los programas en Fortran que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Acerca de esta tarea

Si especifica la opción de procesamiento SQL STDSQL(YES), no defina un SQLCA. Si lo hace, Db2 ignora su SQLCA y la definición de su SQLCA provoca errores en tiempo de compilación. Si especifica la opción de procesamiento SQL STDSQL(NO), incluya un SQLCA explícitamente.

Si su aplicación contiene instrucciones SQL y no incluye un área de comunicaciones SQL (SQLCA), debe declarar variables de host SQLCODE y SQLSTATE individuales. Su programa puede utilizar estas variables para comprobar si una instrucción SQL se ha ejecutado correctamente.

Procedimiento

Elija una de estas acciones:

Opción	Descripción
Para definir el área de comunicaciones SQL:	<p>a. Codifique el SQLCA directamente en el programa o utilice la siguiente instrucción SQL INCLUDE para solicitar una declaración SQLCA estándar:</p> <pre>EXEC SQL INCLUDE SQLCA</pre> <p>Db2 establece los valores SQLCODE y SQLSTATE en el SQLCA después de que se ejecute cada instrucción SQL. Su aplicación debe comprobar estos valores para determinar si la última instrucción SQL se ha realizado correctamente.</p>
Para declarar las variables de host SQLCODE y SQLSTATE:	<p>a. Declare la variable SQLCODE dentro de una declaración BEGIN DECLARE SECTION y una declaración END DECLARE SECTION en sus declaraciones de programa como INTEGER*4. Esta variable también puede denominarse SQLCOD.</p> <p>b. Declare la variable SQLSTATE dentro de una instrucción BEGIN DECLARE SECTION y una instrucción END DECLARE SECTION en las declaraciones de su programa como CHARACTER*5. Esta variable también puede denominarse SQLCOD.</p> <p>Restricción: No declare una variable SQLSTATE como un elemento de una estructura.</p> <p>Requisito: Despues de declarar las variables SQLCODE y SQLSTATE, asegúrese de que todas las sentencias SQL del programa estén dentro del ámbito de la declaración de estas variables.</p>

Tareas relacionadas

[Comprobación de la ejecución de sentencias SQL](#)

Después de ejecutar una sentencia SQL, el programa debe comprobar si hay errores antes de confirmar los datos y manejar los errores que representan.

[Comprobación de la ejecución de sentencias SQL utilizando SQLCA](#)

Un modo de comprobar si una sentencia SQL se ha ejecutado correctamente es utilizar el área de comunicación SQL (SQLCA). Esta área se distingue de la comunicación con Db2.

[Comprobación de la ejecución de sentencias SQL utilizando SQLCODE y SQLSTATE](#)

Siempre que se ejecuta una sentencia SQL, los campos SQLCODE y SQLSTATE de SQLCA reciben un código de retorno.

[Definición de elementos que su programa puede utilizar para comprobar si una sentencia SQL se ha ejecutado correctamente](#)

Si su programa contiene sentencias SQL, el programa debe definir determinada infraestructura para poder comprobar si las sentencias se han ejecutado correctamente. También puede incluir un área de comunicación SQL (SQLCA), que contenga variables SQLCODE y SQLSTATE, o declarar variables host SQLCODE y SQLSTATE.

Definición de áreas de descriptor de SQL en (SQLDA) Fortran

Si su programa incluye determinadas sentencias SQL, debe definir al menos un área de descriptor SQL (SQLDA). Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Procedimiento

Llamar a una subrutina escrita en C, PL/I o lenguaje ensamblador y que utilice la sentencia INCLUDE SQLDA para definir el SQLDA. La subrutina también puede incluir instrucciones SQL para cualquier función SQL dinámica que necesite.

Restricciones:

- Debe colocar las declaraciones SQLDA antes de la primera instrucción SQL que haga referencia al descriptor de datos, a menos que utilice la opción de procesamiento TWOPASS SQL.
- No puede utilizar la instrucción SQL INCLUDE para SQLDA, porque no es compatible con COBOL.

Tareas relacionadas

[Definición de áreas de descriptor de SQL \(SQLDA\)](#)

Si su programa incluye ciertas sentencias SQL, debe definir al menos *un área de descriptor SQL (SQLDA)*. Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Declaración de variables de host y variables de indicador en Fortran

Puede utilizar variables de host, matrices de variables de host y estructuras de host en instrucciones SQL en su programa para pasar datos entre Db2 y su aplicación.

Procedimiento

Para declarar variables de host, matrices de variables de host y estructuras de host:

1. Declare las variables de acuerdo con las siguientes reglas y directrices:

- Cuando declare una variable de tipo carácter, no utilice una expresión para definir la longitud de la variable de tipo carácter. Puede utilizar una variable de tipo host de caracteres con una longitud indefinida (por ejemplo, CHARACTER *(*)). La longitud de cualquiera de estas variables se determina cuando se ejecuta la instrucción SQL asociada.
- Las variables anfitrionas deben ser variables escalares; no pueden ser elementos de vectores o matrices (variables con índice).
- Tenga cuidado al llamar a subrutinas que puedan cambiar los atributos de una variable de host. Dicha alteración puede causar un error mientras el programa se está ejecutando.
- Si especifica la opción de procesamiento ONEPASS SQL, debe declarar explícitamente cada variable de host y cada matriz de variables de host antes de utilizarlas en una instrucción SQL. Si especifica la opción del precompilador TWOPASS, debe declarar cada variable de host antes de usarla en la instrucción DECLARE CURSOR.
- Si especifica la opción de procesamiento SQL STDSQL(YES), debe preceder las sentencias del lenguaje del host que definen las variables del host y las matrices de variables del host con la sentencia BEGIN DECLARE SECTION y seguir las sentencias del lenguaje del host con la sentencia END DECLARE SECTION. De lo contrario, estas declaraciones son opcionales.
- Asegúrese de que cualquier instrucción SQL que utilice una variable de host o una matriz de variables de host esté dentro del ámbito de la instrucción que declara esa variable o matriz.
- Si utiliza el lenguaje de programación C (Db2 precompilador), asegúrese de que los nombres de las variables de host y de las matrices de variables de host sean únicos dentro del programa, incluso si las variables y las matrices de variables se encuentran en bloques, clases, procedimientos, funciones o subrutinas diferentes. Puede calificar los nombres con un nombre de estructura para hacerlos únicos.

2. Opcional: Definir cualquier variable, matriz y estructura de indicadores asociados.

Tareas relacionadas

[Declaración de variables host y variables de indicación](#)

Puede utilizar variables host y variables de indicación en sentencias SQL del programa para pasar datos entre Db2 y la aplicación.

Variables host en Fortran

En programas Fortran, puede especificar variables host numéricas, de caracteres, LOB y ROWID. También puede especificar conjuntos de resultados y localizadores LOB.

Restricciones:

- Solo algunas de las declaraciones válidas de Fortran son declaraciones válidas de variables de host. Si la declaración de una variable no es válida, cualquier instrucción SQL que haga referencia a la variable podría dar lugar al mensaje UNDECLARED HOST VARIABLE.
- Fortran admite algunos tipos de datos sin equivalente SQL (por ejemplo, " REAL*16 " y "COMPLEX"). En la mayoría de los casos, puede utilizar sentencias de conversión (Fortran) para convertir entre los tipos de datos no admitidos y los tipos de datos que SQL permite.
- No puede utilizar localizadores como tipos de columna.

Los siguientes tipos de datos de localización son tipos de datos de Fortran y tipos de datos de SQL:

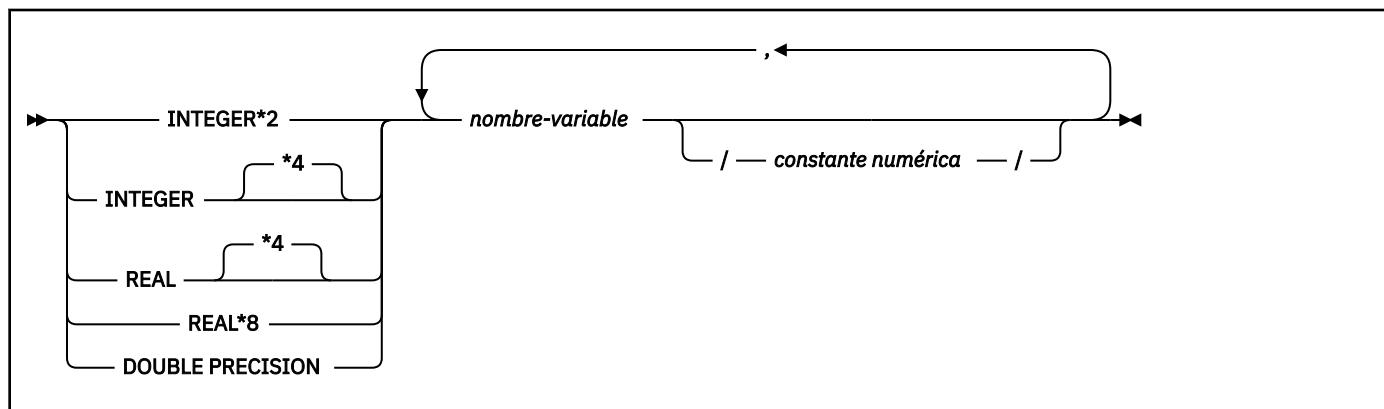
- Localizador de conjunto de resultados
- localizadores de LOB
- Debido a que Fortran no admite tipos de datos gráficos, las aplicaciones de Fortran solo pueden procesar tablas Unicode que utilicen codificación UTF-8.

Recomendaciones:

- Tenga cuidado con el desbordamiento. Por ejemplo, si recupera un valor de columna INTEGER en una variable de host de tipo INTEGER*2, y el valor de columna es mayor que 32767 o -32768, obtendrá una advertencia de desbordamiento o un error, dependiendo de si ha proporcionado una variable indicadora.
- Tenga cuidado con el truncamiento. Por ejemplo, si recupera un valor de columna CHAR de 80 caracteres en una variable de host de tipo " CHARACTER*70 ", los diez caracteres situados más a la derecha de la cadena recuperada se truncarán. Al recuperar un valor de columna decimal o de punto flotante de doble precisión en una variable de host de tipo " INTEGER*4 ", se elimina cualquier valor fraccionario.

Variables numéricas del host

El siguiente diagrama muestra la sintaxis para declarar variables de host numéricas.



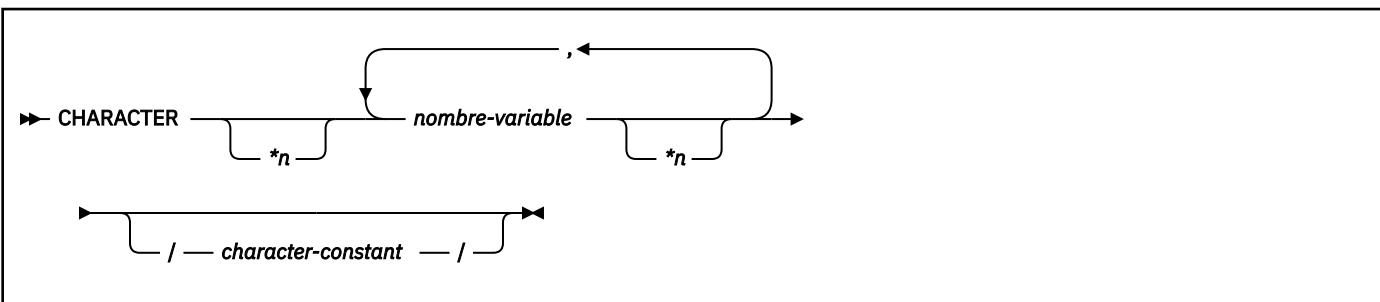
Restricciones:

- Fortran no proporciona un equivalente para el tipo de datos decimal. Para mantener un valor decimal, utilice una de las siguientes variables:

- Una variable entera o de punto flotante, que convierte el valor. Si utiliza una variable entera, perderá la parte fraccionaria del número. Si el número decimal puede superar el valor máximo de un entero o si desea conservar un valor fraccionario, utilice una variable de punto flotante. Los números de coma flotante son aproximaciones de números reales. Por lo tanto, cuando asignas un número decimal a una variable de punto flotante, el resultado puede ser diferente del número original.
- Una variable de host de cadena de caracteres. Utilice la función CHAR para recuperar un valor decimal en ella.
- El tipo de datos SQL DECFLOAT no tiene equivalente en Fortran.

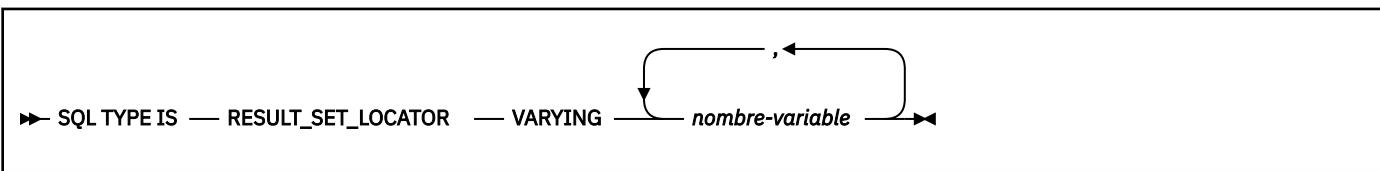
Variables de host de caracteres

El siguiente diagrama muestra la sintaxis para declarar variables de host de caracteres que no sean CLOB.



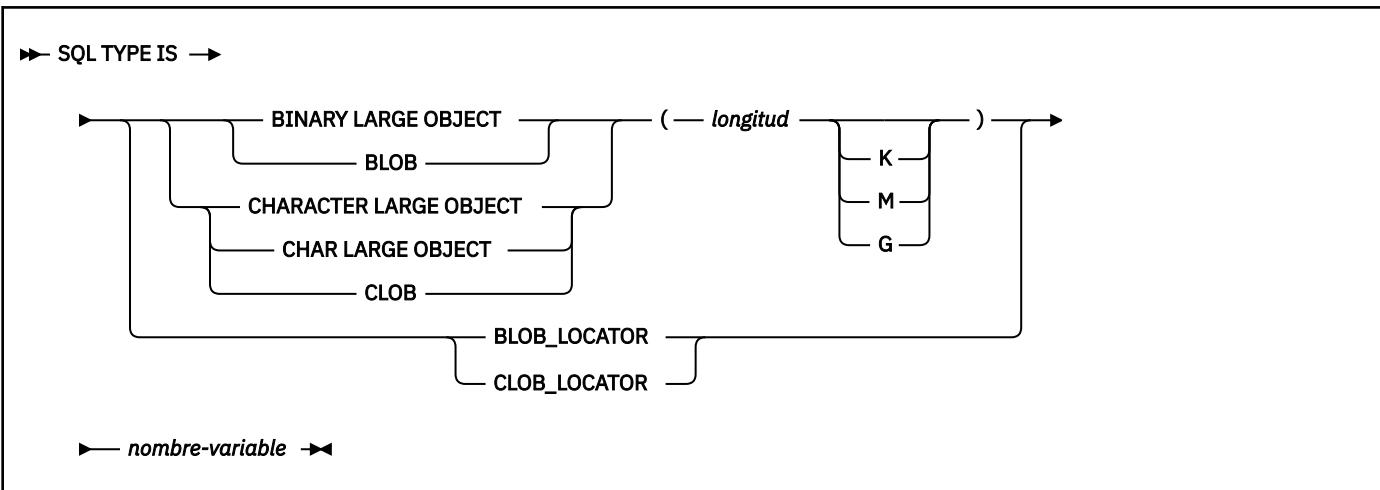
Localizadores de conjuntos de resultados

El siguiente diagrama muestra la sintaxis para declarar localizadores de conjuntos de resultados.



Variables y localizadores LOB

El siguiente diagrama muestra la sintaxis para declarar variables y localizadores de host BLOB y CLOB.



Variables de host ROWID

El siguiente diagrama muestra la sintaxis para las declaraciones de variables ROWID.

► SQL TYPE IS — ROWID — *variable-name* ►

Constantes

La sintaxis de las constantes en los programas de lenguaje de consulta de bases de datos (Fortran) difiere de la sintaxis de las constantes en las sentencias SQL de las siguientes maneras:

- Fortran interpreta una cadena de dígitos con un punto decimal como una constante real. Una instrucción SQL interpreta dicha cadena como una constante decimal. Por lo tanto, utilice la notación exponencial cuando especifique una constante real (es decir, de punto flotante) en una instrucción SQL.
- En Fortran, una constante real (de coma flotante) que tiene una longitud de 8 bytes utiliza una D como indicador de exponente (por ejemplo, 3.14159D+04). Una constante de coma flotante de 8 bytes en una instrucción SQL debe utilizar una E (por ejemplo, 3.14159E+04).

Conceptos relacionados

Variables de host

Utilice variables host para pasar un único elemento de datos entre Db2 y la aplicación.

Uso de variables host en sentencias SQL

Utilice variables de host escalares en sentencias de SQL incorporado para representar un único valor. Las variables host son útiles para almacenar datos recuperados o para pasar valores que van a asignar o utilizar las comparaciones.

Tareas relacionadas

Determinación de si un valor recuperado en una variable host es nulo o está truncado

Antes de que su aplicación manipule los datos recuperados de Db2 en una variable host, determine si el valor es nulo. Determine también si se han truncado al asignarse a la variable. Puede utilizar las variables indicadores para obtener esta información.

Inserción de una sola fila utilizando una variable host

Utilice variables host en la sentencia INSERT cuando no conoce al menos algunos de los valores a insertar hasta que se ejecuta el programa.

Inserción de valores nulos en columnas utilizando las matrices o variables indicadoras

Si necesita insertar valores nulos en una columna, utilizar una matriz o variable indicadora es una forma fácil de hacerlo. Una variable o matriz de indicador está asociada a una variable de host o matriz de variables de lenguaje de host determinada.

Almacenamiento de datos LOB en tablas de Db2

Db2 maneja los datos LOB de manera diferente a otros tipos de datos. Como resultado, a veces es necesario realizar acciones adicionales al definir columnas LOB e insertar los datos LOB.

Recuperación de una única fila de datos en variables host

Si sabe que la consulta devuelve una sola fila, puede especificar una o más variables host que contienen los valores de columna de la fila recuperada.

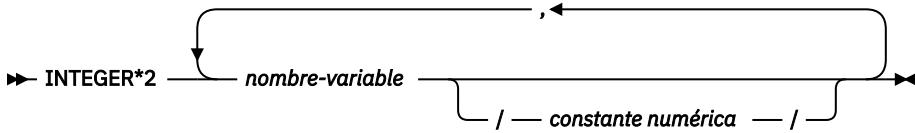
Actualización de datos utilizando variables de host

Cuando quiera actualizar un valor en una tabla Db2, pero no conoce el valor exacto hasta que el programa se ejecute, utilice variables host. Db2 puede cambiar un valor de tabla para que coincida con el valor de la variable de host.

Variables indicadoras en Fortran

Una variable indicadora es un entero de 2 bytes (INTEGER*2). Declare las variables indicadoras de la misma forma que las variables host. Puede mezclar las declaraciones de dos tipos de variables.

El siguiente diagrama muestra la sintaxis para declarar una variable indicadora en Fortran.



Ejemplo

El siguiente ejemplo muestra una sentencia FETCH con las declaraciones de las variables de host que se necesitan para la sentencia FETCH y sus variables indicadoras asociadas.

```

EXEC SQL FETCH CLS_CURSOR INTO :CLSCD,
                           :DAY :DAYIND,
                           :BGN :BGNIND,
                           :END :ENDIND

```

Puede declarar estas variables de la siguiente manera:

```

CHARACTER*7 CLSCD
INTEGER*2 DAY
CHARACTER*8 BGN, END
INTEGER*2 DAYIND, BGNIND, ENDIND

```

Conceptos relacionados

Variables de indicación, matrices y estructuras

Una variable de indicación se asocia con una variable host concreta. Cada variable de indicación contiene un valor entero pequeño que indica alguna información sobre la variable host asociada. Las estructuras y matrices de indicador tienen el mismo propósito para las estructuras y matrices de variables de host.

Tareas relacionadas

Inserción de valores nulos en columnas utilizando las matrices o variables indicadoras

Si necesita insertar valores nulos en una columna, utilizar una matriz o variable indicadora es una forma fácil de hacerlo. Una variable o matriz de indicador está asociada a una variable de host o matriz de variables de lenguaje de host determinada.

Gestión de códigos de error de SQL en aplicaciones Fortran

Las aplicaciones Fortran pueden solicitar más información sobre los códigos de error de SQL utilizando la subrutina DSNTIAR o emitiendo una sentencia GET DIAGNOSTICS.

Procedimiento

Para solicitar más información sobre los errores SQL de los programas de Fortran, utilice los siguientes métodos:

- Puede utilizar la subrutina DSNTIR para convertir un código de retorno SQL en un mensaje de texto. DSNTIR crea una lista de parámetros y llama a DSNTIAR por usted. DSNTIAR toma datos de SQLCA, los formatea en un mensaje y coloca el resultado en un área de salida de mensajes que usted proporciona en su programa de aplicación. Para conocer los conceptos y obtener más información sobre el comportamiento de DSNTIAR, consulte “[Visualización de campos SQLCA invocando DSNTIAR](#)” en la página 558.

Sintaxis DSNTIAR

DSNTIAR tiene la siguiente sintaxis:

```

CALL DSNTIAR ( error-length, message, return-code )

```

Parámetros DSNTIAR

Los parámetros DSNTIAR tienen los siguientes significados:

error-longitud

La longitud total del área de salida del mensaje.

mensaje

Un área de salida, en formato VARCHAR, en la que DSNTIAR coloca el texto del mensaje. La primera semicifra contiene la longitud del área restante; su valor mínimo es 240.

Las líneas de salida de texto se colocan en esta área. Por ejemplo, podría especificar el formato del área de salida como:

```
INTEGER  ERRLEN /1320/
CHARACTER*132 ERRTXT(10)
INTEGER  ICODE
:
CALL DSNTIAR ( ERRLEN, ERRTXT, ICODE )
```

donde ERRLEN es la longitud total del área de salida de mensajes, ERRTXT es el nombre del área de salida de mensajes e ICODE es el código de retorno.

código-retorno

Acepta un código de devolución de DSNTIAR.

Un ejemplo de llamada a DSNTIR (que luego llama a DSNTIAR) desde una aplicación aparece en el programa ensamblador de ejemplo Db2 DSN8BF3, que se encuentra en la biblioteca DSN8C10.SDSNSAMP. Visite “[Aplicaciones de ejemplo suministradas con Db2 for z/OS](#)” en la página [1091](#) para obtener instrucciones sobre cómo acceder e imprimir el código fuente del programa de muestra.

- También puede utilizar el campo de elemento de condición MESSAGE_TEXT de la instrucción GET DIAGNOSTICS para convertir un código de retorno SQL en un mensaje de texto. Los programas que requieren un soporte de mensajes token largos deben codificar la instrucción GET DIAGNOSTICS en lugar de DSNTIAR.

Para obtener más información sobre GET DIAGNOSTICS, consulte “[Comprobación de la ejecución de sentencias SQL utilizando la instrucción GET DIAGNOSTICS](#)” en la página 563.

Tareas relacionadas

[Manejo de códigos de error de SQL](#)

Los programas de aplicación pueden solicitar más información sobre los códigos de error SQL en Db2.

Referencia relacionada

[GET DIAGNOSTICS declaración \(Db2 SQL\)](#)

SQL equivalente y tipos de datos Fortran

Cuando declara variables host en los programas Fortran, el precompilador utiliza tipos de datos SQL equivalentes. Cuando recupera datos de un tipo de datos SQL concreto en una variable host, asegúrese de que la variable host es de un tipo de datos equivalente.

La siguiente tabla describe el tipo de datos SQL y los valores base SQLTYPE y SQLLEN que el precompilador utiliza para las variables de host en las sentencias SQL.

Tabla 111. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas de lenguaje de consulta estructurado (Fortran)

Fortran tipo de datos de variables de host	SQLTYPE de la variable de host ¹	SQLLEN de la variable de host	Tipos de datos SQL
INTEGER*2	500	2	SMALLINT
INTEGER*4	496	4	ENTERO
REAL*4	480	4	FLOAT (precisión simple)
REAL*8	480	8	FLOAT (doble precisión)
CARÁCTER*n	452	n	CHAR (n)

Tabla 111. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas de lenguaje de consulta estructurado (Fortran) (continuación)

Fortran tipo de datos de variables de host	SQLTYPE de la variable de host ¹	SQLLEN de la variable de host	Tipos de datos SQL
TIPO SQL ES RESULT_SET_LOCATOR	972	4	Localizador de conjunto de resultados. No utilice este tipo de datos como tipo de columna.
TIPO SQL ES BLOB_LOCATOR	960	4	Localizador BLOB. No utilice este tipo de datos como tipo de columna.
TIPO SQL ES CLOB_LOCATOR	964	4	Localizador CLOB. No utilice este tipo de datos como tipo de columna.
TIPO SQL ES BLOB(<i>n</i>) $1 \leq n \leq 2147483647$	404	<i>n</i>	BLOB (<i>n</i>)
TIPO SQL ES CLOB(<i>n</i>) $1 \leq n \leq 2147483647$	408	<i>n</i>	CLOB (<i>n</i>)
EL TIPO SQL ES ROWID	904	40	ROWID

Notas:

- Si una variable de host incluye una variable de indicador, el valor SQLTYPE es el valor SQLTYPE base más 1.

Fortran La siguiente tabla muestra las variables de host de SQL equivalentes para cada tipo de datos SQL. Utilice esta tabla para determinar el tipo de datos e Fortran s para las variables de host que defina para recibir la salida de la base de datos. Por ejemplo, si recupera datos TIMESTAMP, puede definir una variable de tipo CHARACTER*n.

Esta tabla muestra las conversiones directas entre los tipos de datos SQL y los tipos de datos de Fortran. Sin embargo, varios tipos de datos SQL son compatibles. Cuando realiza asignaciones o comparaciones de datos que tienen tipos de datos compatibles, Db2 convierte esos tipos de datos compatibles.

Tabla 112. Fortran equivalentes de variables de host que puede utilizar al recuperar datos de un tipo de datos SQL concreto

Tipos de datos SQL	Fortran variable de host equivalente	Notas
SMALLINT	INTEGER*2	
ENTERO	INTEGER*4	
BIGINT	no soportado	
DECIMAL(<i>p,s</i>) o NUMERIC(<i>p,s</i>)	sin equivalente exacto	Uso REAL*8
FLOAT(<i>n</i>) precisión simple	REAL*4	$1 \leq n \leq 21$
FLOAT(<i>n</i>) doble precisión	REAL*8	$2 \leq n \leq 53$
CHAR (<i>n</i>)	CARÁCTER*n	$1 \leq n \leq 255$
VARCHAR(<i>n</i>)	sin equivalente exacto	Utilice una variable de tipo host de caracteres que sea lo suficientemente grande como para contener el mayor valor VARCHAR esperado.

Tabla 112. Fortran equivalentes de variables de host que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Fortran variable de host equivalente	Notas
BINARY	no soportado	
VARBINARY	no soportado	
GRAPHIC (<i>n</i>)	no soportado	
VARGRAPHIC (<i>n</i>)	no soportado	
FECHA	CARÁCTER*n	Si está utilizando una rutina de salida de fecha, <i>n</i> viene determinada por dicha rutina; de lo contrario, <i>n</i> debe ser al menos 10.
HORA	CARÁCTER*n	Si está utilizando una rutina de salida de tiempo, <i>n</i> está determinado por esa rutina. De lo contrario, <i>n</i> debe ser al menos 6; para incluir segundos, <i>n</i> debe ser al menos 8.
TIMESTAMP	CARÁCTER*n	<i>n</i> debe tener al menos 19 años. Para incluir microsegundos, <i>n</i> debe ser 26; si <i>n</i> es menor que 26, se produce un truncamiento en la parte de los microsegundos.
TIMESTAMP(0)	CARÁCTER*n	<i>n</i> debe tener al menos 19 años.
TIMESTAMP(<i>p</i>) <i>p</i> > 0	CARÁCTER*n	<i>n</i> debe tener al menos 19 años. Para incluir fracciones de segundo, <i>n</i> debe ser 20+x, donde <i>x</i> es el número de fracciones de segundo que se van a incluir; si <i>x</i> es menor que <i>p</i> , se trunca la parte de las fracciones de segundo.
FECHA Y HORA (<i>p</i>) CON ZONA HORARIA	sin equivalente exacto	Utilice una variable de host de caracteres que sea lo suficientemente grande como para contener la marca de tiempo más grande esperada con el valor de zona horaria.
Localizador de conjunto de resultados	TIPO SQL ES RESULT_SET_LOCATOR	Utilice este tipo de datos solo para recibir conjuntos de resultados. No utilice este tipo de datos como tipo de columna.
localizador de BLOB	TIPO SQL ES BLOB_LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas BLOB. No utilice este tipo de datos como tipo de columna. ¹
localizador de CLOB	TIPO SQL ES CLOB_LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas CLOB. No utilice este tipo de datos como tipo de columna.
localizador de DBCLOB	no soportado	
BLOB (<i>n</i>)	TIPO SQL ES BLOB(<i>n</i>)	1≤ <i>n</i> ≤214748 36471

Tabla 112. Fortran equivalentes de variables de host que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Fortran variable de host equivalente	Notas
CLOB (n)	TIPO SQL ES CLOB(n)	1≤n≤214748 36471
DBCLOB (n)	no soportado	
ROWID	EL TIPO SQL ES ROWID	
XML	no soportado	

Conceptos relacionados

[Compatibilidad de SQL y tipos de datos de lenguaje](#)

Los tipos de datos de variable host que se utilizan en sentencias SQL deben ser compatibles con los tipos de datos de las columnas con las que tiene intención de utilizarlos.

[Declaraciones de variable de referencia de archivo LOB, variable host LOB y localizador LOB](#)

Cuando escribe aplicaciones para manipular datos LOB, necesita declarar variables hosto para contener los datos LOB o el localizador LOB. De lo contrario, debe declarar variables de referencia de archivo LOB para apuntar a datos LOB.

Aplicaciones PL/I que emiten sentencias SQL

Puede codificar sentencias SQL en un programa PL/I siempre que pueda utilizar sentencias ejecutables.

La primera declaración del programa PL/I debe ser la declaración PROCEDURE con OPTIONS(MAIN), a menos que el programa sea un procedimiento almacenado. Una aplicación de procedimiento almacenado puede ejecutarse como subrutina.

Cada instrucción SQL en un programa PL/I debe comenzar con EXEC SQL y terminar con un punto y coma (;). Las palabras clave EXEC y SQL deben aparecer en una línea, pero el resto de la instrucción puede aparecer en líneas posteriores.

Podría codificar una instrucción UPDATE en un programa PL/I de la siguiente manera:

```
EXEC SQL UPDATE DSN8C10.DEPT  
      SET MGRNO = :MGR_NUM  
      WHERE DEPTNO = :INT_DEPT ;
```

Comentarios

Puede incluir comentarios PL/I en sentencias SQL incrustadas siempre que pueda utilizar un espacio en blanco, excepto entre las palabras clave EXEC y SQL. También puede incluir comentarios SQL en cualquier instrucción SQL. Para obtener más información, consulte [Comentarios SQL \(Db2 SQL\)](#).

Para incluir caracteres DBCS en comentarios, debe delimitar los caracteres con un carácter de control shift-out y shift-in; el primer carácter shift-in en la cadena DBCS señala el final de la cadena DBCS.

Continuación para instrucciones SQL

Las reglas de continuación de línea para las sentencias SQL son las mismas que para otras sentencias PL/I, excepto que debe especificar EXEC SQL en una línea.

Delimitadores para instrucciones SQL

Delimita una instrucción SQL en su programa PL/I con la palabra clave inicial " EXEC SQL " y un punto y coma (;).

Declaración de tablas y vistas

Su programa PL/I debe incluir una instrucción DECLARE TABLE para describir cada tabla y ver los accesos del programa. Puede utilizar el generador de declaraciones de Db2 (DCLGEN) para generar las declaraciones DECLARE TABLE.

Incluir código

Puede utilizar sentencias SQL o declaraciones de variables de host PL/I de un miembro de un conjunto de datos particionado utilizando la siguiente sentencia SQL en el código fuente donde desee incluir las sentencias:

```
EXEC SQL INCLUDE member-name;
```

No se pueden anidar sentencias SQL INCLUDE. No utilice la instrucción PL/I %INCLUDE para incluir instrucciones SQL o instrucciones DCL de variables de host. Debe utilizar el preprocesador PL/I para resolver cualquier instrucción %INCLUDE antes de utilizar el precompilador Db2 . No utilice directivas de preprocesador PL/I dentro de sentencias SQL.

Márgenes

Codifique las sentencias SQL en las columnas 2-72, a menos que haya especificado otros márgenes al precompilador Db2 . Si EXEC SQL comienza antes del margen izquierdo especificado, el precompilador de instrucciones SQL (Db2) no reconoce la instrucción SQL.

Nombres

Puede utilizar cualquier nombre PL/I válido para una variable de host. No utilice nombres de entrada externos o nombres de planes de acceso que empiecen por «DSN», ni nombres de variables de host que empiecen por «SQL». Estos nombres están reservados para Db2.

Números de secuencia

Las sentencias fuente que genera el precompilador de Db2 no incluyen números de secuencia. IEL0378I los mensajes del compilador PL/I identifican líneas de código sin números de secuencia. Puede ignorar estos mensajes.

Etiquetas de extracto

Puede especificar una etiqueta de sentencia para las sentencias SQL ejecutables. Sin embargo, las sentencias INCLUDE *nombre-archivo-texto* y END DECLARE SECTION no pueden tener etiquetas de sentencia.

WHENEVER, sentencia

El objetivo de la cláusula GOTO en una instrucción SQL WHENEVER debe ser una etiqueta en el código fuente PL/I y debe estar dentro del alcance de cualquier instrucción SQL a la que WHENEVER afecte.

Uso de caracteres del juego de caracteres de doble byte (DBCS)

Las siguientes consideraciones se aplican al uso de DBCS en programas PL/I con sentencias SQL:

- Si utiliza DBCS en el código fuente PL/I, se aplican las reglas de codificación de caracteres (Db2) para los siguientes elementos del lenguaje:
 - Series gráficas
 - Constantes de series gráficas
 - Identificadores del lenguaje principal
 - Datos mezclados en series de caracteres
 - Opción DATOS MIXTOS
- El preprocesador PL/I transforma el formato de las constantes DBCS. Si no desea esa transformación, ejecute el precompilador Db2 **antes que** el preprocesador.
- Si utiliza constantes de cadena gráfica o datos mixtos en sentencias SQL preparadas dinámicamente, y si su aplicación requiere el compilador PL/I versión 2 (o posterior), las sentencias preparadas dinámicamente deben utilizar el formato de constante mixta PL/I.
 - Si prepara la declaración a partir de una variable host, cambie la asignación de cadena a una cadena mixta PL/I.
 - Si prepara la declaración a partir de una cadena PL/I, cámbiela por una variable de host y, a continuación, cambie la asignación de cadena por una cadena mixta PL/I.

Ejemplo:

```
SQLSTMT = 'SELECT <dbdb> FROM table-name'M;
EXEC SQL PREPARE STMT FROM :SQLSTMT;
```

- Si desea que un identificador DBCS se parezca a una cadena gráfica PL/I, debe utilizar un identificador delimitado.
- Si incluye caracteres DBCS en los comentarios, debe delimitar los caracteres con un carácter de control shift-out y shift-in. El primer carácter de cambio de turno señala el final de la cadena DBCS.
- Puede declarar nombres de variables de host que utilicen caracteres DBCS en programas de aplicación PL/I. Las reglas para usar nombres de variables DBCS en PL/I siguen las reglas existentes para los identificadores ordinarios DBCS SQL, excepto en lo que respecta a la longitud. La longitud máxima de una variable de host es de 128 bytes Unicode en un Db2. . Para obtener información sobre las reglas para los identificadores ordinarios DBCS SQL, consulte la información sobre los identificadores SQL.

Restricciones:

- Los nombres de variables DBCS deben contener únicamente caracteres DBCS. Mezclar caracteres de un solo byte (SBCS) con caracteres DBCS en un nombre de variable DBCS produce resultados impredecibles.
- Un nombre de variable DBCS no puede continuar en la línea siguiente.
- El preprocesador PL/I cambia los caracteres DBCS que no son kanji a caracteres SBCS de código de intercambio decimal codificado binario extendido (EBCDIC). Para evitar este cambio, utilice caracteres Kanji DBCS para los nombres de variables DBCS, o ejecute el compilador PL/I sin el preprocesador PL/I.

Consideraciones especiales de PL/I

Las siguientes consideraciones se aplican a los programas escritos en PL/I:

- Al compilar un programa PL/I que incluya sentencias SQL, debe utilizar la opción del compilador PL/I CHARSET (60 EBCDIC).
- Al compilar un programa PL/I que utilice tipos de datos BIGINT o LOB, especifique las siguientes opciones del compilador: LIMITS(FIXEDBIN(63), FIXEDDEC(31))
- En casos excepcionales, los comentarios generados en PL/I pueden contener un punto y coma. El punto y coma genera el mensaje del compilador IEL0239I, que puede ignorar.
- El código generado en una declaración PL/I puede contener la función ADDR de un campo definido como carácter variable. Esto produce el mensaje IBM1051I o IBM1180I W, los cuales puede ignorar.
- El código generado por el precompilador en el código fuente PL/I puede contener la función NULL(). Esto produce el mensaje IEL0533I, que puede ignorar a menos que también utilice NULL como variable PL/I. Si utiliza NULL como variable PL/I en una aplicación de Db2 , también debe declarar NULL como función incorporada (DCL NULL BUILTIN;) para evitar errores del compilador PL/I.
- El macroprocesador PL/I puede generar sentencias SQL o sentencias DCL de variables de host si ejecuta el macroprocesador antes de ejecutar el precompilador de Db2 .

Si utiliza el macroprocesador PL/I, no utilice la instrucción PL/I *PROCESS en el código fuente para pasar opciones al compilador PL/I. Puede especificar las opciones necesarias en el parámetro COPTION del comando DSNH o en la opción PARM.PLI=options de la instrucción EXEC en el procedimiento DSNHPLI.

- El uso de la función multitarea PL/I, en la que múltiples tareas ejecutan sentencias SQL, provoca resultados impredecibles.
- El tipo de datos de host PL/I WIDECHAR solo es compatible a través del coprocesador Db2 .
- Cuando se utiliza la constante PL/I WX widechar, Db2 solo admite el formato bigendian. Por lo tanto, cuando asigne una constante a la variable de host de tipo widechar en PL/I, asegúrese de que se utiliza el formato bigendian. Por ejemplo:

```
HVWC1 = '003100320033006100620063'WX;
```

Equivalente a:

```
HVWC1 = '123abc';
```

HVWC1 se define como una variable de host de tipo WIDECHAR.

- Opción PL/I SQL Preprocessor, CCSID0 y NOCCSID0, consideración de uso cuando se utiliza con el coprocesador Db2 .

– Cuando se utiliza CCSID0 (predeterminado), se promueve la compatibilidad con programas PL/I más antiguos, que utilizaban el precompilador Db2 . Durante la preparación del programa, no se asocia ningún valor CCSID con la variable host, excepto para la variable host de tipo WIDECHAR. Para la variable host de tipo WIDECHAR, el preprocesador PL/I SQL siempre asigna CCSID 1200.

Durante BIND y el tiempo de ejecución, si no hay ningún CCSID asociado a la variable host, se utiliza la opción BIND, ENCODING, que está pensada para los datos de la aplicación. Si no se especifica la opción ENCODING BIND, se utiliza el valor predeterminado de la opción ENCODING BIND.

- Cuando utiliza NOCCSID0, se asocia un CCSID con la variable de host durante la preparación del programa. El CCSID se deriva de los siguientes elementos durante la preparación del programa:

- DECLARE :hv VARIABLE CCSID xxxx especificado.
- Fuente CCSID, si no DECLARE VARIABLE... Se especifica CCSID xxxx para la variable de host. Durante el tiempo de BIND, tenga en cuenta que el proceso de BIND desconoce el CCSID asignado a la variable host durante la preparación del programa. Para obtener más información sobre la resolución CCSID de tiempo BIND, consulte [Determinación del esquema de codificación y CCSID de una serie \(Introducción a DB2 para z/OS\)](#).

Para la variable host utilizada en SQL estático, asegurarse de que se asigne/derive un CCSID preciso y coincidente a través de DECLARE VARIABLE... CCSID xxxx, CCSID de origen u opción ENCODING BIND o el valor predeterminado de instalación

Para el marcador de parámetros utilizado en SQL dinámico, asegurarse de que se asigne/derive un CCSID preciso para la variable de host correspondiente a través de DECLARE VARIABLE... CCSID xxxx, opción ENCODING BIND o la configuración predeterminada de la instalación. El CCSID de origen no influye en el marcador de parámetros.

Manejo de códigos de error de SQL

Las aplicaciones PL/I pueden solicitar más información sobre errores SQL en Db2. Para obtener más información, consulte “[Gestión de códigos de error de SQL en aplicaciones PL/I](#)” en la página 741.

Conceptos relacionados

[DCLGEN \(generador de declaraciones\)](#)

El programa debe declarar las tablas y vistas a las que accede. El generador de declaraciones de Db2, DCLGEN, crea estas sentencias DECLARE para programas C, COBOL y PL/I programs, y así no necesita codificar las sentencias él mismo. DCLGEN genera también las estructuras de variable de lenguaje principal correspondientes.

[Utilización de matrices de variables de host en sentencias SQL](#)

Utilice matrices de variables de lenguaje de host en sentencias de SQL incorporado para representar valores que el programa no conoce hasta que se ejecuta la consulta. Las matrices de variables de host son útiles para almacenar un conjunto de valores recuperados o para pasar un conjunto de valores que se van a insertar en una tabla.

[Identificadores en SQL \(Db2 SQL\)](#)

Tareas relacionadas

[Descripción general de las aplicaciones de programación que acceden a datos de Db2 for z/OS](#)

Las aplicaciones que interactúan con Db2, en primer lugar se deben conectar a Db2. Entonces pueden leer, añadir o modificar datos o manipular objetos de Db2.

[Inclusión de SQL dinámico en el programa](#)

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

Manejo de códigos de error de SQL

Los programas de aplicación pueden solicitar más información sobre los códigos de error SQL en Db2.

Establecimiento de límites para el uso de recursos de sistema utilizando el recurso de límite de recursos (Db2 Performance)

Ejemplos de programación de PL/I

Puede escribir programas en PL/I que acceden a un subsistema Db2 local o remoto y ejecutan sentencias de SQL estático o dinámico. Esta información contiene varios ejemplos de programación de ese tipo.

Para preparar y ejecutar estas aplicaciones, utilice el JCL de *prefijo.SDSNSAMP* como modelo para su JCL.

Referencia relacionada

Ejemplos de programación de Assembler, C, C++, COBOL, PL/I y REXX (Db2 Programming samples)

Ejemplo de programa PL/I que llama a un procedimiento almacenado

Puede llamar al procedimiento almacenado GETPRML que utiliza la convención de vinculación GENERAL WITH NULLS desde un programa PL/I en un sistema z/OS .

La siguiente figura contiene el programa PL/I de ejemplo que llama al procedimiento almacenado GETPRML.

```

*PROCESS SYSTEM(MVS);
CALPRML:
  PROC OPTIONS(MAIN);

  /*************************************************************************/
  /* Declare the parameters used to call the GETPRML                   */
  /* stored procedure.                                                 */
  /*************************************************************************/
  DECLARE PROCNM CHAR(18),      /* INPUT parm -- PROCEDURE name */
         SCHEMA CHAR(8),       /* INPUT parm -- User's schema */
         OUT_CODE FIXED BIN(31),
                           /* OUTPUT -- SQLCODE from the    */
                           /*          SELECT operation.   */
         PARMLST CHAR(254)     /* OUTPUT -- RUNOPTS for      */
                           /*          VARYING,           */
                           /*          the matching row in the */
                           /*          catalog table SYSROUTINES */
         PARMIND FIXED BIN(15); /* PARMLST indicator variable */
  /*************************************************************************/
  /* Include the SQLCA                                                 */
  /*************************************************************************/
  EXEC SQL INCLUDE SQLCA;
  /*************************************************************************/
  /* Call the GETPRML stored procedure to retrieve the                */
  /* RUNOPTS values for the stored procedure. In this               */
  /* example, we request the RUNOPTS values for the                 */
  /* stored procedure named DSN8EP2.                                 */
  /*************************************************************************/
  PROCNM = 'DSN8EP2';
  /* Input parameter -- PROCEDURE to be found */
  SCHEMA = ' ';
  /* Input parameter -- SCHEMA in SYSROUTINES */
  PARMIND = -1; /* The PARMLST parameter is an output parm. */
  /* Mark PARMLST parameter as null, so the DB2 */
  /* requester does not have to send the entire */
  /* PARMLST variable to the server. This        */
  /* helps reduce network I/O time, because     */
  /* PARMLST is fairly large.                  */

  EXEC SQL
    CALL GETPRML(:PROCNM,
                  :SCHEMA,
                  :OUT_CODE,
                  :PARMLST INDICATOR :PARMIND);

  IF SQLCODE=-0 THEN      /* If SQL CALL failed,          */
    DO;
      PUT SKIP EDIT('SQL CALL failed due to SQLCODE = ',
                    SQLCODE) (A(34),A(14));
      PUT SKIP EDIT('SQLERRM = ',
                    SQLERRM) (A(10),A(70));
    END;
  ELSE                   /* If the CALL worked,          */
    IF OUT_CODE=-0 THEN  /* Did GETPRML hit an error? */
      PUT SKIP EDIT('GETPRML failed due to RC = ',
                    OUT_CODE) (A(33),A(14));
    ELSE                /* Everything worked.          */
      PUT SKIP EDIT('RUNOPTS = ', PARMLST) (A(11),A(200));
  RETURN;
END CALPRML;

```

Figura 42. Llamar a un procedimiento almacenado desde un programa PL/I

Ejemplo de procedimiento almacenado PL/I con una convención de vinculación GENERAL

Puede llamar a un procedimiento almacenado que utilice la convención de vinculación GENERAL desde un programa PL/I.

Este ejemplo de procedimiento almacenado busca en la tabla de catálogo Db2 SYSIBM.SYSROUTINES una fila que coincida con los parámetros de entrada del programa cliente. Los dos parámetros de entrada contienen valores para NAME y SCHEMA.

La convención de vinculación para este procedimiento almacenado es GENERAL.

Los parámetros de salida de este procedimiento almacenado contienen el SQLCODE de la operación SELECT y el valor de la columna RUNOPTS recuperado de la tabla SYSIBM.SYSROUTINES.

La declaración CREATE PROCEDURE para este procedimiento almacenado podría tener este aspecto:

```
CREATE PROCEDURE GETPRML(Procnm CHAR(18) IN, Schema CHAR(8) IN,
    Outcode INTEGER OUT, Parmlst VARCHAR(254) OUT)
LANGUAGE PLI
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME "GETPRML"
COLLID GETPRML
ASUTIME NO LIMIT
PARAMETER STYLE GENERAL
STAY RESIDENT NO
RUN OPTIONS "MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)"
WLM ENVIRONMENT SAMPPROG
PROGRAM TYPE MAIN
SECURITY DB2
RESULT SETS 0
COMMIT ON RETURN NO;
```

El siguiente ejemplo es un procedimiento almacenado PL/I con la convención de vinculación GENERAL.

```
*PROCESS SYSTEM(MVS);

GETPRML:
    PROC(Procnm, Schema, Out_code, Parmlst)
        OPTIÖNS(MAIN NOEXECOPS REENTRANT);

    DECLARE Procnm CHAR(18),      /* INPUT parm -- PROCEDURE name */
           Schema CHAR(8),      /* INPUT parm -- User's SCHEMA */
           Out_code FIXED BIN(31), /* OUTPUT -- SQLCODE from      */
                                /* the SELECT operation.   */
           Parmlst CHAR(254),     /* OUTPUT -- RUNOPTS for      */
                                /* the matching row in      */
                                /* SYSIBM.SYSROUTINES      */
                                VARYING;             /* */

    EXEC SQL INCLUDE SQLCA;

    /* Execute SELECT from SYSIBM.SYSROUTINES in the catalog. */
    EXEC SQL
        SELECT RUNOPTS INTO :Parmlst
        FROM SYSIBM.SYSROUTINES
        WHERE NAME=:Procnm AND
              SCHEMA=:Schema;

    Out_code = SQLCODE;          /* return SQLCODE to caller */
    RETURN;
END GETPRML;
```

Ejemplo de procedimiento almacenado PL/I con una convención de vinculación GENERAL WITH NULLS

Puede llamar a un procedimiento almacenado que utilice la convención de vinculación GENERAL WITH NULLS desde un programa PL/I.

Este ejemplo de procedimiento almacenado busca en la tabla de catálogo Db2 SYSIBM.SYSROUTINES una fila que coincida con los parámetros de entrada del programa cliente. Los dos parámetros de entrada contienen valores para NAME y SCHEMA.

La convención de vinculación para este procedimiento almacenado es GENERAL WITH NULLS.

Los parámetros de salida de este procedimiento almacenado contienen el SQLCODE de la operación SELECT y el valor de la columna RUNOPTS recuperado de la tabla SYSIBM.SYSROUTINES.

La declaración CREATE PROCEDURE para este procedimiento almacenado podría tener este aspecto:

```
CREATE PROCEDURE GETPRML(Procnm CHAR(18) IN, Schema CHAR(8) IN,
    Outcode INTEGER OUT, Parmlst VARCHAR(254) OUT)
LANGUAGE PLI
```

```

DETERMINISTIC
READS SQL DATA
EXTERNAL NAME "GETPRML"
COLLID GETPRML
ASUTIME NO LIMIT
PARAMETER STYLE GENERAL WITH NULLS
STAY RESIDENT NO
RUN OPTIONS "MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)"
WLM ENVIRONMENT SAMPPROG
PROGRAM TYPE MAIN
SECURITY DB2
RESULT SETS 0
COMMIT ON RETURN NO;

```

El siguiente ejemplo es un procedimiento almacenado PL/I con la convención de vinculación GENERAL WITH NULLS.

```

*PROCESS SYSTEM(MVS);

GETPRML:
  PROC(PROCNM, SCHEMA, OUT_CODE, PARMLST, INDICATORS)
    OPTIONS(MAIN NOEXECOPS REENTRANT);

  DECLARE PROCNM CHAR(18),      /* INPUT parm -- PROCEDURE name */
         SCHEMA CHAR(8),      /* INPUT parm -- User's schema */

         OUT_CODE FIXED BIN(31), /* OUTPUT -- SQLCODE from   */
                                /* the SELECT operation. */
         PARMLST CHAR(254)      /* OUTPUT -- PARMLIST for */
                                /* VARYING;           */
                                /* the matching row in */
                                /* SYSIBM.SYSROUTINES */
  DECLARE 1 INDICATORS,        /* Declare null indicators for */
                                /* input and output parameters. */
  3 PROCNM_IND   FIXED BIN(15),
  3 SCHEMA_IND   FIXED BIN(15),
  3 OUT_CODE_IND FIXED BIN(15),
  3 PARMLST_IND  FIXED BIN(15);

  EXEC SQL INCLUDE SQLCA;

  IF PROCNM_IND<0 | SCHEMA_IND<0 THEN
    DO;          /* If any input parm is NULL,      */
      OUT_CODE = 9999;    /* Set output return code.       */
      OUT_CODE_IND = 0;   /* Output return code is not NULL.*/
                           /* Assign NULL value to PARMLST. */
    END;
  ELSE          /* If input parms are not NULL,  */
    DO;          /* */
  /*************************************************************************/
  /* Issue the SQL SELECT against the SYSIBM.SYSROUTINES             */
  /* DB2 catalog table.                                                 */
  /*************************************************************************/
    EXEC SQL
      SELECT RUNOPTS INTO :PARMLST
        FROM SYSIBM.SYSROUTINES
        WHERE NAME=:PROCNM AND
              SCHEMA=:SCHEMA;
    PARMLST_IND = 0;      /* Mark PARMLST as not NULL.     */
    OUT_CODE = SQLCODE;   /* return SQLCODE to caller      */
    OUT_CODE_IND = 0;    /* */
    OUT_CODE_IND = 0;    /* Output return code is not NULL.*/
  END;
  RETURN;

END GETPRML;

```

Gestión de códigos de error de SQL en aplicaciones PL/I

Las aplicaciones PL/I pueden solicitar más información sobre los códigos de error de SQL utilizando la subrutina DSNTIAR o emitiendo una sentencia GET DIAGNOSTICS.

Procedimiento

Para solicitar información sobre errores SQL en programas PL/I, utilice los siguientes métodos:

- Puede utilizar la subrutina DSNTIAR para convertir un código de retorno SQL en un mensaje de texto. DSNTIAR toma datos de SQLCA, los formatea en un mensaje y coloca el resultado en un área de salida de mensajes que usted proporciona en su programa de aplicación. Para conocer los conceptos y obtener más información sobre el comportamiento de DSNTIAR, consulte “[Visualización de campos SQLCA invocando DSNTIAR](#)” en la página 558.

Sintaxis DSNTIAR

```
CALL DSNTIAR ( sqlca, message, lrecl );
```

Parámetros DSNTIAR

Los parámetros DSNTIAR tienen los siguientes significados:

sqlca

Un área de comunicación SQL.

mensaje

Un área de salida, en formato VARCHAR, en la que DSNTIAR coloca el texto del mensaje. La primera semicifra contiene la longitud del área restante; su valor mínimo es 240.

Las líneas de salida de texto, cada una con la longitud especificada en *lrecl*, se colocan en esta área. Por ejemplo, podría especificar el formato del área de salida como:

```
DCL DATA_LEN FIXED BIN(31) INIT(132);
DCL DATA_DIM FIXED BIN(31) INIT(10);
DCL 1 ERROR_MESSAGE AUTOMATIC,
      3 ERROR_LEN   FIXED BIN(15) UNAL INIT((DATA_LEN*DATA_DIM)),
      3 ERROR_TEXT(DATA_DIM) CHAR(DATA_LEN);
:
CALL DSNTIAR ( SQLCA, ERROR_MESSAGE, DATA_LEN );
```

donde ERROR_MESSAGE es el nombre del área de salida del mensaje, DATA_DIM es el número de líneas en el área de salida del mensaje y DATA_LEN es la longitud de cada línea.

lrecl

Una palabra completa que contiene la longitud de registro lógica de los mensajes de salida, en el rango de 72 a 240.

Debido a que DSNTIAR es un programa de lenguaje ensamblador, debe incluir las siguientes directivas en su aplicación PL/I:

```
DCL DSNTIAR ENTRY OPTIONS (ASM,INTER,RETCODE);
```

En el programa ensamblador de ejemplo Db2 DSN8BP3, contenido en la biblioteca DSN8C10.SDSNSAMP, aparece un ejemplo de llamada a DSNTIAR desde una aplicación. Visite “[Aplicaciones de ejemplo suministradas con Db2 for z/OS](#)” en la página 1091 para obtener instrucciones sobre cómo acceder e imprimir el código fuente del programa de muestra.

- Si su CICS solicitud requiere CICS almacenamiento, debe utilizar la subrutina DSNTIAC en lugar de DSNTIAR.

Sintaxis DSNTIAC

DSNTIAC tiene la siguiente sintaxis:

```
CALL DSNTIAC ( eib, commarea, sqlca, msg, lrecl );
```

Parámetros DSNTIAC

DSNTIAC tiene parámetros adicionales, que debe utilizar para llamadas a rutinas que utilizan CICS comandos.

eib

Bloque de interfaz EXEC

área de clientes

área de comunicación

Para obtener más información sobre estos parámetros, consulte la guía de programación de aplicaciones correspondiente para CICS. Las descripciones de los parámetros restantes son las mismas que las de DSNTIAR. Tanto DSNTIAC como DSNTIAR formatean el SQLCA de la misma manera.

Debe definir DSNTIA1 en el CSD. Si carga DSNTIAR o DSNTIAC, también debe definirlos en el CSD. Para ver un ejemplo de declaraciones de generación de entradas CSD para su uso con DSNTIAC, consulte el trabajo DSNTEJ5A.

El código fuente ensamblador para DSNTIAC y el trabajo DSNTEJ5A, que ensambla y edita enlaces DSNTIAC, están en *el prefijo* del conjunto de datos.SDSENSAMP.

- También puede utilizar el campo de elemento de condición MESSAGE_TEXT de la instrucción GET DIAGNOSTICS para convertir un código de retorno SQL en un mensaje de texto. Los programas que requieren un soporte de mensajes token largos deben codificar la instrucción GET DIAGNOSTICS en lugar de DSNTIAR.

Para obtener más información sobre GET DIAGNOSTICS, consulte “[Comprobación de la ejecución de sentencias SQL utilizando la instrucción GET DIAGNOSTICS](#)” en la página 563.

Tareas relacionadas

[Manejo de códigos de error de SQL](#)

Los programas de aplicación pueden solicitar más información sobre los códigos de error SQL en Db2.

Referencia relacionada

[GET DIAGNOSTICS declaración \(Db2 SQL\)](#)

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en PL/I

Los programas en PL/I que contienen sentencias SQL pueden incluir un área de comunicación SQL (SQLCA) para comprobar si una sentencia SQL se ha ejecutado correctamente. También, estos programas pueden declarar variables host SQLCODE y SQLSTATE individuales.

Acerca de esta tarea

Si especifica la opción de procesamiento SQL STDSQL(YES), no defina un SQLCA. Si lo hace, Db2 ignora su SQLCA y la definición de su SQLCA provoca errores en tiempo de compilación. Si especifica la opción de procesamiento SQL STDSQL(NO), incluya un SQLCA explícitamente.

Si su aplicación contiene instrucciones SQL y no incluye un área de comunicaciones SQL (SQLCA), debe declarar variables de host SQLCODE y SQLSTATE individuales. Su programa puede utilizar estas variables para comprobar si una instrucción SQL se ha ejecutado correctamente.

Procedimiento

Elija una de estas acciones:

Opción	Descripción
Para definir el área de comunicaciones SQL:	<p>a. Codifique el SQLCA directamente en el programa o utilice la siguiente instrucción SQL INCLUDE para solicitar una declaración SQLCA estándar:</p> <pre>EXEC SQL INCLUDE SQLCA</pre> <p>Db2 establece los valores SQLCODE y SQLSTATE en el SQLCA después de que se ejecute cada instrucción SQL. Su aplicación debe comprobar estos valores para determinar si la última instrucción SQL se ha realizado correctamente.</p>

Opción	Descripción
Para declarar variables de host SQLCODE y SQLSTATE:	<p>a. Declare la variable SQLCODE dentro de una declaración BEGIN DECLARE SECTION y una declaración END DECLARE SECTION en las declaraciones de su programa como BIN FIXED (31).</p> <p>b. Declare la variable SQLSTATE dentro de una instrucción BEGIN DECLARE SECTION y una instrucción END DECLARE SECTION en las declaraciones de su programa como CHARACTER(5).</p> <p>Restricción: No declare una variable SQLSTATE como un elemento de una estructura.</p> <p>Requisito: Después de declarar las variables SQLCODE y SQLSTATE, asegúrese de que todas las sentencias SQL del programa estén dentro del ámbito de la declaración de estas variables.</p>

Tareas relacionadas

[Comprobación de la ejecución de sentencias SQL](#)

Después de ejecutar una sentencia SQL, el programa debe comprobar si hay errores antes de confirmar los datos y manejar los errores que representan.

[Comprobación de la ejecución de sentencias SQL utilizando SQLCA](#)

Un modo de comprobar si una sentencia SQL se ha ejecutado correctamente es utilizar el área de comunicación SQL (SQLCA). Esta área se distingue de la comunicación con Db2.

[Comprobación de la ejecución de sentencias SQL utilizando SQLCODE y SQLSTATE](#)

Siempre que se ejecuta una sentencia SQL, los campos SQLCODE y SQLSTATE de SQLCA reciben un código de retorno.

[Definición de elementos que su programa puede utilizar para comprobar si una sentencia SQL se ha ejecutado correctamente](#)

Si su programa contiene sentencias SQL, el programa debe definir determinada infraestructura para poder comprobar si las sentencias se han ejecutado correctamente. También puede incluir un área de comunicación SQL (SQLCA), que contenga variables SQLCODE y SQLSTATE, o declarar variables host SQLCODE y SQLSTATE.

Definición de áreas de descriptor de SQL (SQLDA) en PL/I

Si su programa incluye determinadas sentencias SQL, debe definir al menos un área de descriptor SQL (SQLDA). Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Procedimiento

Codifique el SQLDA directamente en el programa, o utilice la siguiente instrucción SQL INCLUDE para solicitar una declaración SQLDA estándar:

```
EXEC SQL INCLUDE SQLDA
```

Restricción: Debe colocar las declaraciones SQLDA antes de la primera instrucción SQL que haga referencia al descriptor de datos, a menos que utilice la opción de procesamiento TWOPASS SQL.

Tareas relacionadas

[Definición de áreas de descriptor de SQL \(SQLDA\)](#)

Si su programa incluye ciertas sentencias SQL, debe definir al menos *un área de descriptor SQL (SQLDA)*. Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Referencia relacionada

[Área de descriptor de SQL \(SQLDA\) \(Db2 SQL\)](#)

Declaración de variables host y variables de indicador en PL/I

Puede utilizar variables de host, matrices de variables de host y estructuras de host en instrucciones SQL en su programa para pasar datos entre Db2 y su aplicación.

Procedimiento

Para declarar variables de host, matrices de variables de host y estructuras de host:

1. Declare las variables de acuerdo con las siguientes reglas y directrices:

- Si especifica la opción de procesamiento ONEPASS SQL, debe declarar explícitamente cada variable de host y cada matriz de variables de host antes de utilizarlas en una instrucción SQL. Si especifica la opción del precompilador TWOPASS, debe declarar cada variable de host antes de usarla en la instrucción DECLARE CURSOR.
- Si especifica la opción de procesamiento SQL STDSQL(YES), debe preceder las sentencias del lenguaje del host que definen las variables del host y las matrices de variables del host con la sentencia BEGIN DECLARE SECTION y seguir las sentencias del lenguaje del host con la sentencia END DECLARE SECTION. De lo contrario, estas declaraciones son opcionales.
- Asegúrese de que cualquier instrucción SQL que utilice una variable de host o una matriz de variables de host esté dentro del ámbito de la instrucción que declara esa variable o matriz.
- Si utiliza el lenguaje de programación C (Db2 precompiler), asegúrese de que los nombres de las variables de host y de las matrices de variables de host sean únicos dentro del programa, incluso si las variables y las matrices de variables se encuentran en bloques, clases, procedimientos, funciones o subrutinas diferentes. Puede calificar los nombres con un nombre de estructura para hacerlos únicos.

2. Opcional: Definir cualquier variable, matriz y estructura de indicadores asociados.

Tareas relacionadas

[Declaración de variables host y variables de indicación](#)

Puede utilizar variables host y variables de indicación en sentencias SQL del programa para pasar datos entre Db2 y la aplicación.

Variables host en PL/I

En programas PL/I, puede especificar variables host numéricas, de caracteres, gráficas, binarias, LOB, XML y ROWID. También puede especificar conjuntos de resultados, tabla y localizadores LOB y variables de referencia de archivos LOB y XML.

Restricciones:

- Solo algunas de las declaraciones PL/I válidas son declaraciones de variables de host válidas. El precompilador utiliza los valores predeterminados de los atributos de datos que se especifican en la instrucción PL/I DEFAULT. Si la declaración de una variable de host no es válida, cualquier instrucción SQL que haga referencia a la variable podría dar lugar al mensaje UNDECLARED HOST VARIABLE.
- Los atributos de alineación, alcance y almacenamiento de las variables de host tienen las siguientes restricciones:
 - Una declaración con el atributo de ámbito EXTERNO y el atributo de almacenamiento ESTÁTICO también debe tener el atributo de almacenamiento INICIAL.
 - Si utiliza el atributo de almacenamiento BASED, debe seguirlo con una expresión de localización de elementos PL/I.
 - Las variables de host pueden ser de clase de almacenamiento ESTÁTICA, CONTROLADA, BASADA o AUTOMÁTICA, o bien opciones. Sin embargo, CICS requiere que los programas sean reentrantes.

Aunque el precompilador utiliza únicamente los nombres y atributos de datos de las variables e ignora los atributos de alineación, alcance y almacenamiento, usted no debe ignorar estas restricciones. Si las ignora, podría tener problemas para compilar el código fuente PL/I que genera el precompilador.

- PL/I admite algunos tipos de datos sin equivalente SQL (variables COMPLEX y BIT, por ejemplo). En la mayoría de los casos, puede utilizar sentencias PL/I para convertir entre los tipos de datos PL/I no compatibles y los tipos de datos compatibles con SQL.
- No puede utilizar localizadores como tipos de columna.

Los siguientes tipos de datos de localización son tipos de datos PL/I, así como tipos de datos SQL:

- Localizador de conjunto de resultados
- Localizador de tablas
- localizadores de LOB
- El precompilador no admite reglas de alcance PL/I.

Recomendaciones:

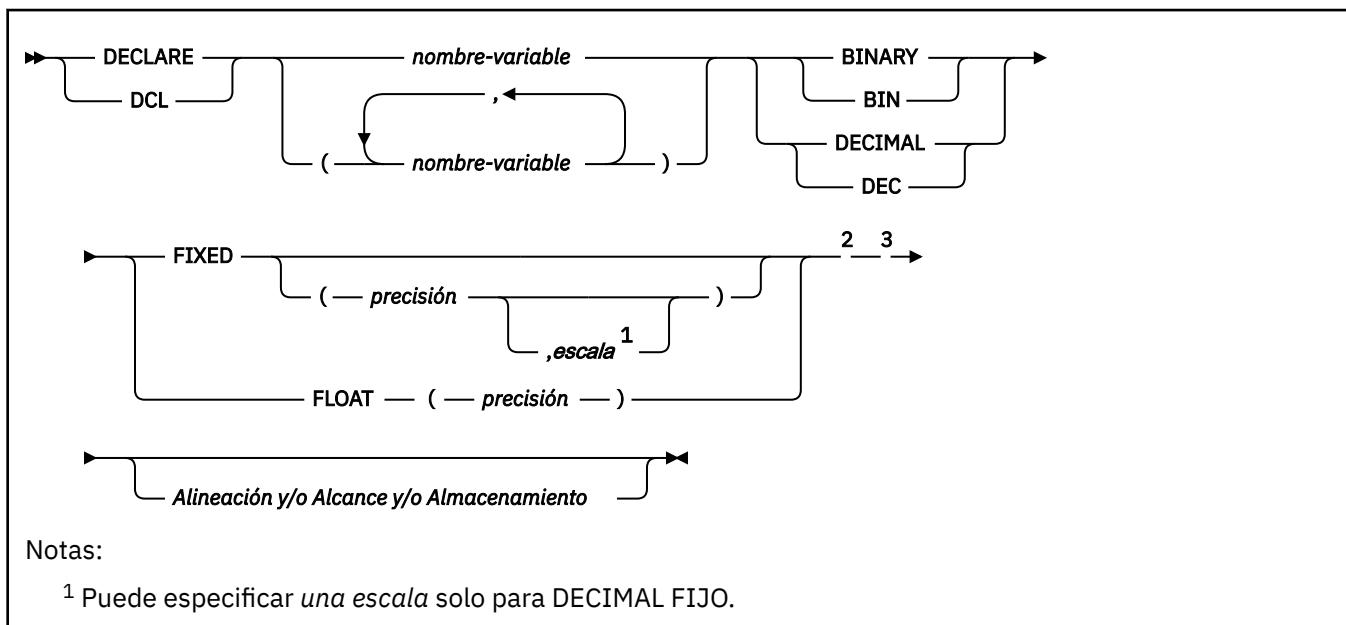
- Tenga cuidado con el desbordamiento. Por ejemplo, si recupera un valor de columna INTEGER en una variable de host BIN FIXED(15) y el valor de la columna es mayor que 32767 o menor que -32768, obtendrá una advertencia de desbordamiento o un error, dependiendo de si ha proporcionado una variable indicadora.
- Tenga cuidado con el truncamiento. Por ejemplo, si recupera un valor de columna CHAR de 80 caracteres en una variable de host CHAR(70), los diez caracteres situados más a la derecha de la cadena recuperada se truncan. Recuperar un valor de columna decimal o de punto flotante de doble precisión en una variable de host BIN FIXED(31) elimina cualquier parte fraccionaria del valor. Del mismo modo, recuperar un valor de columna con un tipo de datos DECIMAL en una variable decimal PL/I con una precisión inferior podría truncar el valor.

Variables numéricas del host

Puede especificar las siguientes formas de variables de host numéricas:

- Números de punto flotante (hexadecimal y decimal)
- Números enteros y números enteros pequeños
- Números decimales

El siguiente diagrama muestra la sintaxis para declarar variables de host numéricas.



² Puede especificar atributos de variables de host en cualquier orden que sea aceptable para PL/I. Por ejemplo, BIN FIXED(31), BINARY FIXED(31), BIN(31) FIXED y FIXED BIN(31) son todos aceptables.

³ El atributo UNSIGNED no debe especificarse con BINARY y FIXED para una declaración de variable de host numérica.

Para los tipos de datos de punto flotante binario o punto flotante hexadecimal, utilice la opción de procesamiento FLOAT SQL para especificar si la variable del host está en formato de punto flotante binario IEEE o punto flotante hexadecimal z/Architecture. Db2 no comprueba si el formato del contenido de la variable host coincide con el formato que especificó con la opción de procesamiento FLOAT SQL. Por lo tanto, debe asegurarse de que el contenido de la variable de host de punto flotante coincida con el formato que especificó con la opción de procesamiento FLOAT SQL. Db2 convierte todos los datos de entrada de punto flotante al formato de punto flotante hexadecimal de z/Architecture antes de almacenarlos.

Si el compilador PL/I que está utilizando no admite un tipo de datos decimales con una precisión superior a 15, utilice uno de los siguientes tipos de variables para datos decimales:

- Variables decimales con una precisión menor o igual a 15, si los valores de datos reales encajan. Si recupera un valor decimal en una variable decimal con una escala inferior a la columna de origen en la base de datos, la parte fraccionaria del valor podría truncarse.
- Un entero o una variable de punto flotante, que convierte el valor. Si utiliza una variable entera, perderá la parte fraccionaria del número. Si el número decimal puede superar el valor máximo de un entero o si desea conservar un valor fraccionario, utilice una variable de punto flotante. Los números de coma flotante son aproximaciones de números reales. Por lo tanto, cuando asignas un número decimal a una variable de punto flotante, el resultado puede ser diferente del número original.
- Una variable de host de cadena de caracteres. Utilice la función CHAR para recuperar un valor decimal en ella.

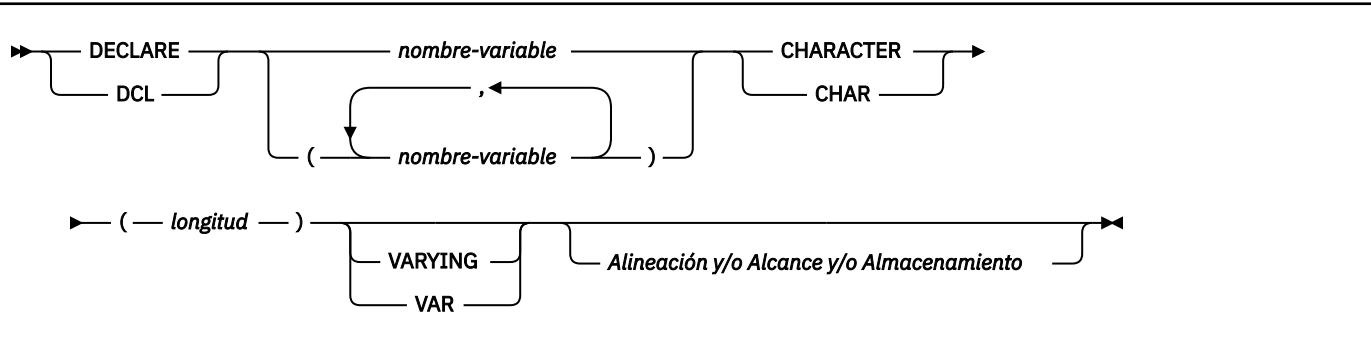
Para utilizar los tipos de datos de host de punto flotante decimal PL/I, debe utilizar las opciones de compilador FLOAT(DFP) y ARCH(7) y el coprocesador Db2 . La precisión máxima para el FLOAT DECIMAL extendido será 34 (no 33 como para el float hexadecimal). La precisión máxima para el tipo de coma flotante DECIMAL FLOAT corto será 7 (no 6 como para el tipo de coma flotante hexadecimal).

Variables de host de caracteres

Puede especificar las siguientes formas de variables de host de caracteres:

- Series de longitud fija
- Series de longitud variable
- CLOB

El siguiente diagrama muestra la sintaxis para declarar variables de host de caracteres, distintas de los CLOB.

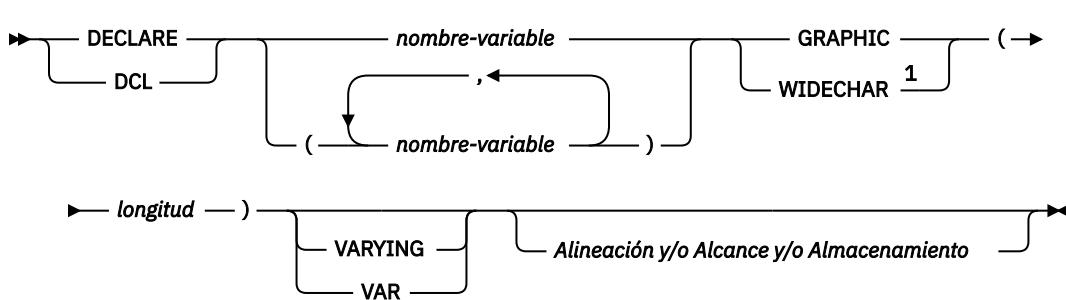


Variables gráficas de host

Puede especificar las siguientes formas de variables de host de caracteres:

- Series de longitud fija
- Series de longitud variable
- DBCLOB

El siguiente diagrama muestra la sintaxis para declarar variables de host gráficas distintas de DBCLOB.



Notas:

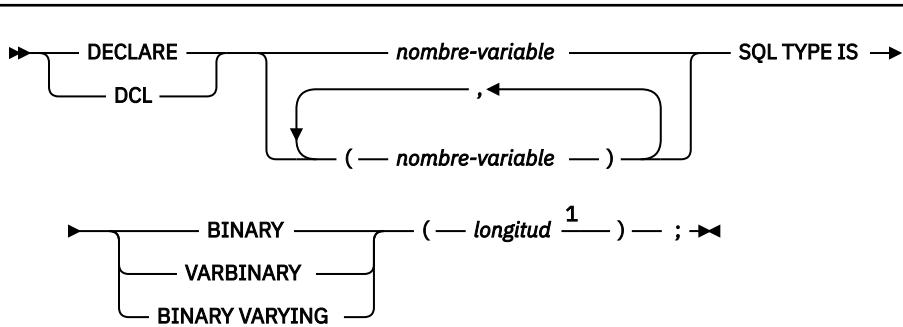
- ¹ Utilice WIDECHAR solo para datos de e UTF-16 encia UNICODE. WIDECHAR solo es compatible con el coprocesador de la arquitectura Db2 .

Variables binarias de host

Puede especificar las siguientes formas de variables de host binarias:

- Series de longitud fija
- Series de longitud variable
- BLOB

El siguiente diagrama muestra la sintaxis para declarar variables de host BINARIAS.



Notas:

- ¹ Para las variables de host BINARIAS, la longitud debe estar en el rango de 1 a 255. Para las variables de host VARBINARY, la longitud debe estar en el rango de 1 a 32704.

PL/I no tiene variables que se correspondan con los tipos de datos binarios SQL BINARY y VARBINARY. Para crear variables de host que puedan utilizarse con estos tipos de datos, utilice la cláusula SQL TYPE IS.

Cuando se hace referencia a una variable de host BINARIA o VARBINARIA en una instrucción SQL, se debe utilizar la variable que se especifica en la declaración SQL TYPE. Cuando haga referencia a la variable de host en una instrucción de lenguaje de host, debe utilizar la variable que genera Db2 .

Ejemplos de declaraciones de variables binarias

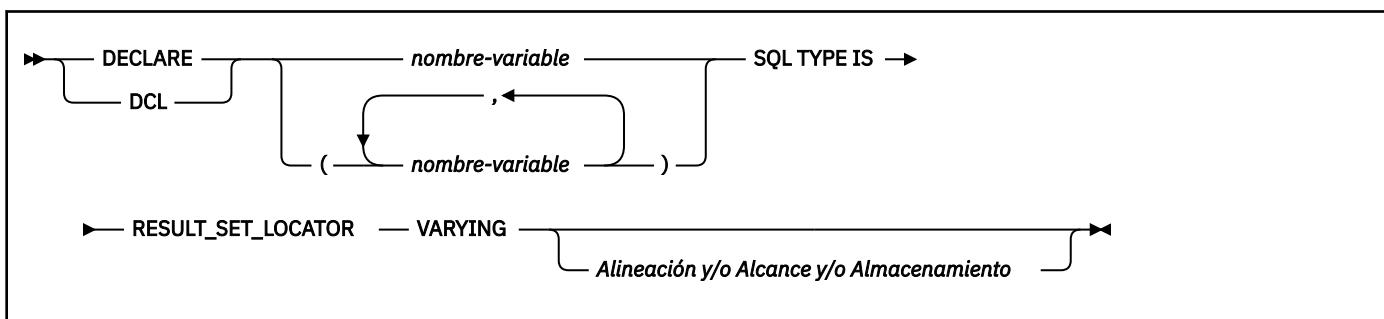
La siguiente tabla muestra ejemplos de variables que genera Db2 cuando se declaran variables de host binarias.

Tabla 113. Ejemplos de declaraciones de variables BINARIAS y VARBINARIAS para PL/I

Declaración variable que se incluye en el programa PL/I	Variable correspondiente que genera Db2 en el miembro de origen de salida
DCL BIN_VAR SQL TYPE IS BINARY(10);	DCL BIN_VAR CHAR(10);
DCL VBIN_VAR SQL TYPE IS VARBINARY(10);	DCL VBIN_VAR CHAR(10) VAR;

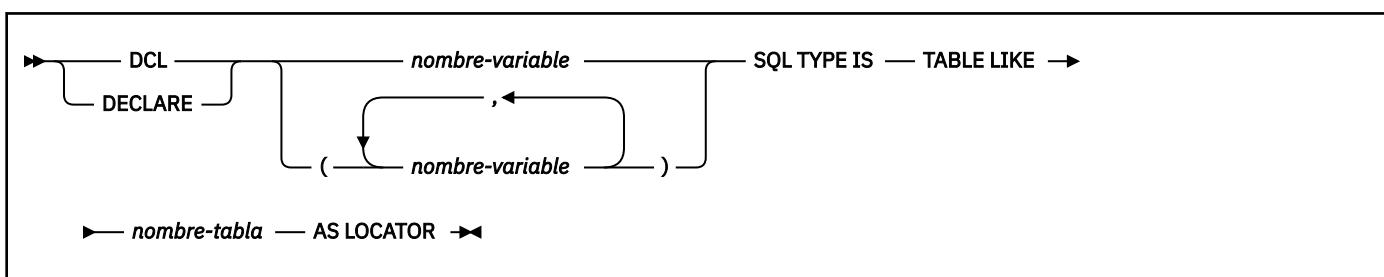
Localizadores de conjuntos de resultados

El siguiente diagrama muestra la sintaxis para declarar localizadores de conjuntos de resultados.



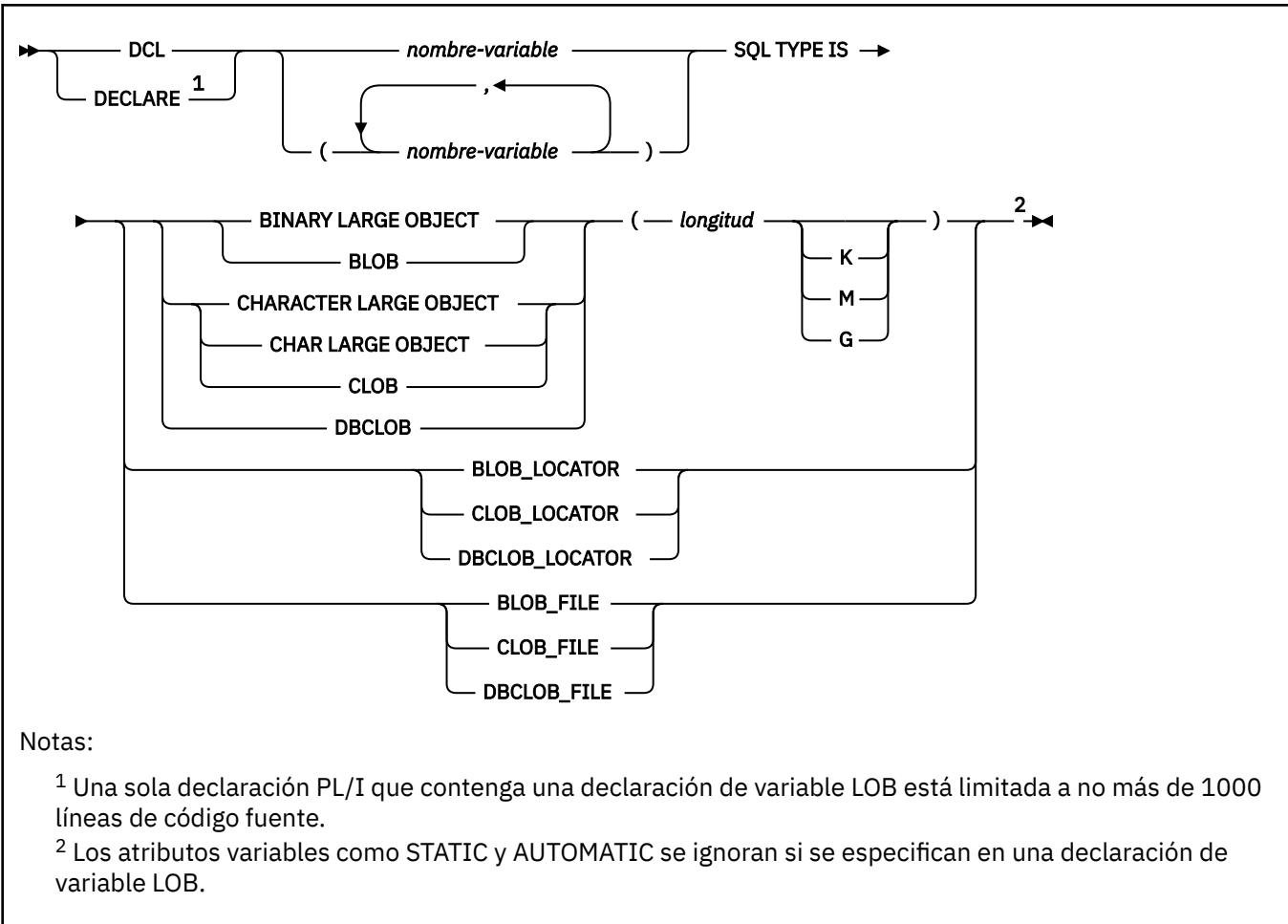
Localizadores de mesas

El siguiente diagrama muestra la sintaxis para declarar localizadores de tablas.



Variables LOB, localizadores y variables de referencia de archivos

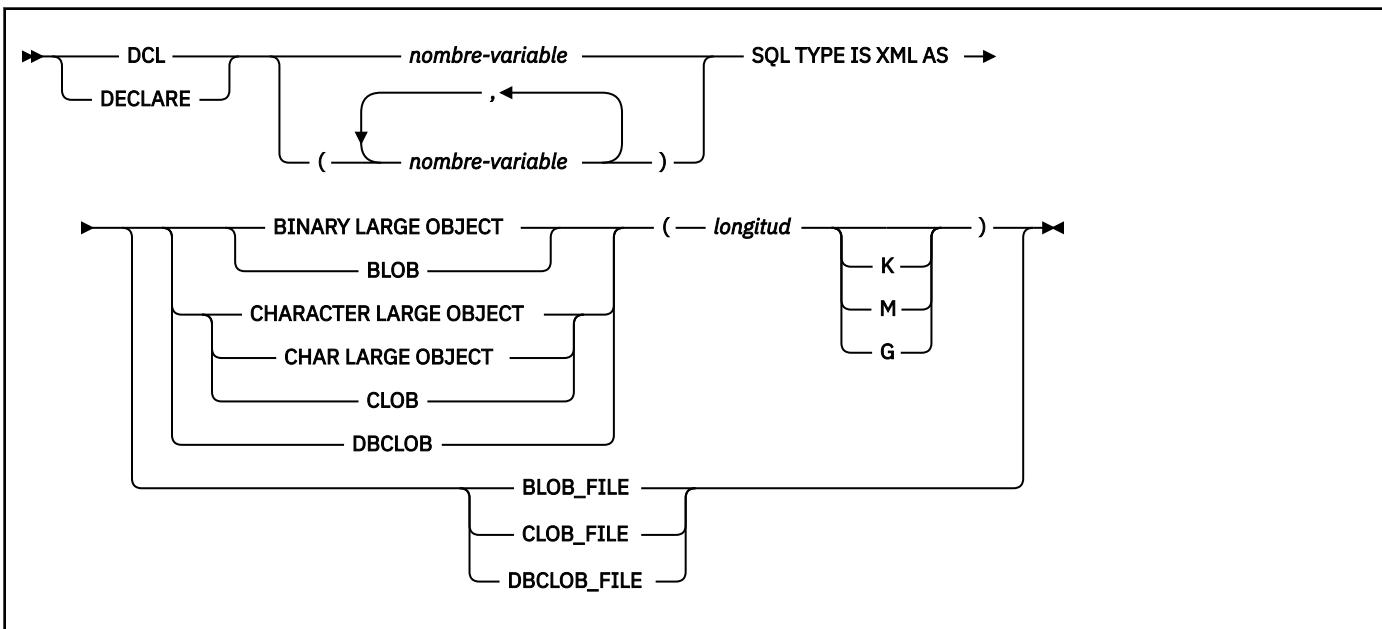
El siguiente diagrama muestra la sintaxis para declarar variables de host BLOB, CLOB y DBCLOB, localizadores y variables de referencia de archivos.



Nota: Los atributos variables como STATIC y AUTOMATIC se ignoran si se especifican en una declaración de variable LOB.

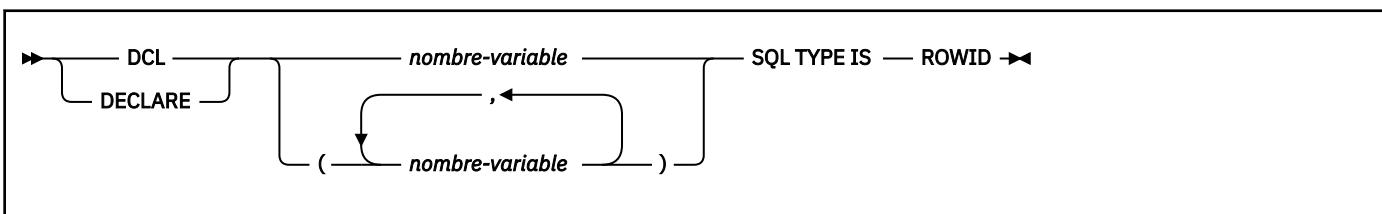
Variables de referencia de archivos y host de datos XML

El siguiente diagrama muestra la sintaxis para declarar variables de host BLOB, CLOB y DBCLOB y variables de referencia de archivo para tipos de datos XML.



Variables de host ROWID

El siguiente diagrama muestra la sintaxis para declarar variables de host ROWID.



Conceptos relacionados

Variables de host

Utilice variables host para pasar un único elemento de datos entre Db2 y la aplicación.

Uso de variables host en sentencias SQL

Utilice variables de host escalares en sentencias de SQL incorporado para representar un único valor. Las variables host son útiles para almacenar datos recuperados o para pasar valores que van a asignar o utilizar las comparaciones.

Tipos de datos numéricos (Db2 SQL)

Tareas relacionadas

Determinación de si un valor recuperado en una variable host es nulo o está truncado

Antes de que su aplicación manipule los datos recuperados de Db2 en una variable host, determine si el valor es nulo. Determine también si se han truncado al asignarse a la variable. Puede utilizar las variables indicadoras para obtener esta información.

Inserción de una sola fila utilizando una variable host

Utilice variables host en la sentencia INSERT cuando no conoce al menos algunos de los valores a insertar hasta que se ejecuta el programa.

Inserción de valores nulos en columnas utilizando las matrices o variables indicadoras

Si necesita insertar valores nulos en una columna, utilizar una matriz o variable indicadora es una forma fácil de hacerlo. Una variable o matriz de indicador está asociada a una variable de host o matriz de variables de lenguaje de host determinada.

Almacenamiento de datos LOB en tablas de Db2

Db2 maneja los datos LOB de manera diferente a otros tipos de datos. Como resultado, a veces es necesario realizar acciones adicionales al definir columnas LOB e insertar los datos LOB.

Recuperación de una única fila de datos en variables host

Si sabe que la consulta devuelve una sola fila, puede especificar una o más variables host que contienen los valores de columna de la fila recuperada.

Recuperación de una única fila de datos en una estructura de host

Si sabe que la consulta devuelve varios valores de columna para una única fila, puede especificar una estructura de host para contener los valores de columna.

Actualización de datos utilizando variables de host

Cuando quiera actualizar un valor en una tabla Db2, pero no conoce el valor exacto hasta que el programa se ejecute, utilice variables host. Db2 puede cambiar un valor de tabla para que coincida con el valor de la variable de host.

Matrices de variables de host en PL/I

En los programas PL/I, puede especificar matrices de variables de host de tipo numérico, carácter, gráfico, binario, LOB, XML y ROWID. También puede especificar localizadores LOB y variables de referencia de archivos LOB y XML.

Solo se puede hacer referencia a las matrices de variables de lenguaje principal como una referencia simple en los contextos siguientes. En los diagramas de sintaxis, *host-variable-array* designa una referencia a una matriz de variables de host.

- En una sentencia FETCH para una captación de varias filas. Consulte [FETCH declaración \(Db2 SQL\)](#).

- En la forma *FOR n ROWS* de la sentencia *INSERT* con una matriz de variables de host para los datos de origen. Consulte [INSERT declaración \(Db2 SQL\)](#).
- En una sentencia *MERGE* con varias filas de datos de origen. Consulte [MERGE declaración \(Db2 SQL\)](#).
- En una sentencia *EXECUTE* para proporcionar un valor para un marcador de parámetro en una forma dinámica *FOR n ROWS* de la sentencia *INSERT* o una sentencia *MERGE*. Consulte [EXECUTE declaración \(Db2 SQL\)](#).

Restricciones:

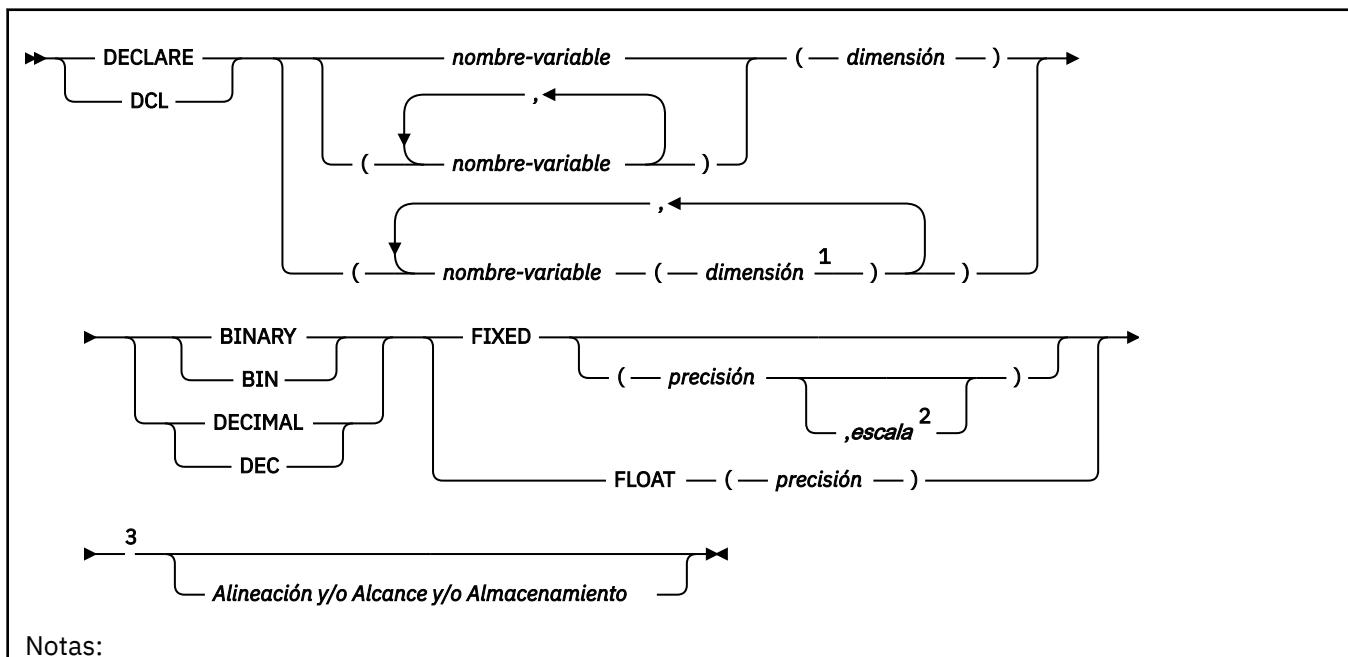
- Solo algunas de las declaraciones PL/I válidas son declaraciones de variables de host válidas. El precompilador utiliza los valores predeterminados de los atributos de datos que se especifican en la instrucción PL/I *DEFAULT*. Si la declaración de una variable host no es válida, cualquier instrucción SQL que haga referencia a la matriz de variables host podría dar como resultado el mensaje UNDECLARED HOST VARIABLE ARRAY.
- Los atributos de alineación, alcance y almacenamiento de las matrices de variables de host tienen las siguientes restricciones:
 - Una declaración con el atributo de ámbito EXTERNO y el atributo de almacenamiento ESTÁTICO también debe tener el atributo de almacenamiento INICIAL.
 - Si utiliza el atributo de almacenamiento BASED, debe seguirlo con una expresión de localización de elementos PL/I.
 - Las variables de host pueden ser de clase de almacenamiento ESTÁTICA, CONTROLADA, BASADA o AUTOMÁTICA, o bien opciones. Sin embargo, CICS requiere que los programas sean reentrantes.

Aunque el precompilador utiliza únicamente los nombres y atributos de datos de las matrices de variables e ignora los atributos de alineación, ámbito y almacenamiento, no debe ignorar estas restricciones. Si las ignora, podría tener problemas para compilar el código fuente PL/I que genera el precompilador.

- Debe especificar el atributo *ALIGNED* cuando declare matrices de caracteres de longitud variable o matrices gráficas de longitud variable que se vayan a utilizar en sentencias *INSERT* y *FETCH* de varias filas.

Matrices de variables de host numéricas

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host numéricas.



¹ la dimensión debe ser una constante entera en el rango de 1 a 32767.

² Puede especificar la escala solo para DECIMAL FIJO.

³ Puede especificar atributos de matriz de variables de host en cualquier orden que sea aceptable para PL/I. Por ejemplo, BIN FIXED(31), BINARY FIXED(31), BIN(31) FIXED y FIXED BIN(31) son todos aceptables.

Ejemplo

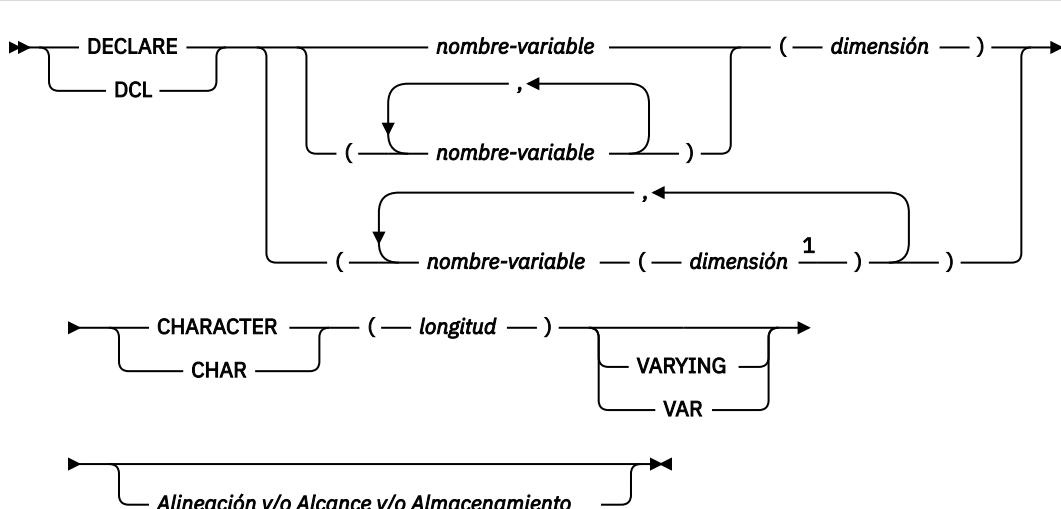
El siguiente ejemplo muestra una declaración de una matriz de indicadores.

```
DCL IND_ARRAY(100) BIN FIXED(15); /* DCL ARRAY of 100 indicator variables */
```

Para utilizar los tipos de datos de host de punto flotante decimal PL/I, es necesario utilizar las opciones de compilador FLOAT(DFP) y ARCH(7) y el coprocesador de punto flotante de precisión arbitraria (Db2). La precisión máxima para el FLOAT DECIMAL extendido será 34 (no 33 como para el float hexadecimal). La precisión máxima para el tipo de coma flotante DECIMAL FLOAT corto será 7 (no 6 como en el caso del tipo de coma flotante hexadecimal).

Matrices de variables de host de caracteres

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host de caracteres que no sean CLOB.



Notas:

¹ la dimensión debe ser una constante entera en el rango de 1 a 32767.

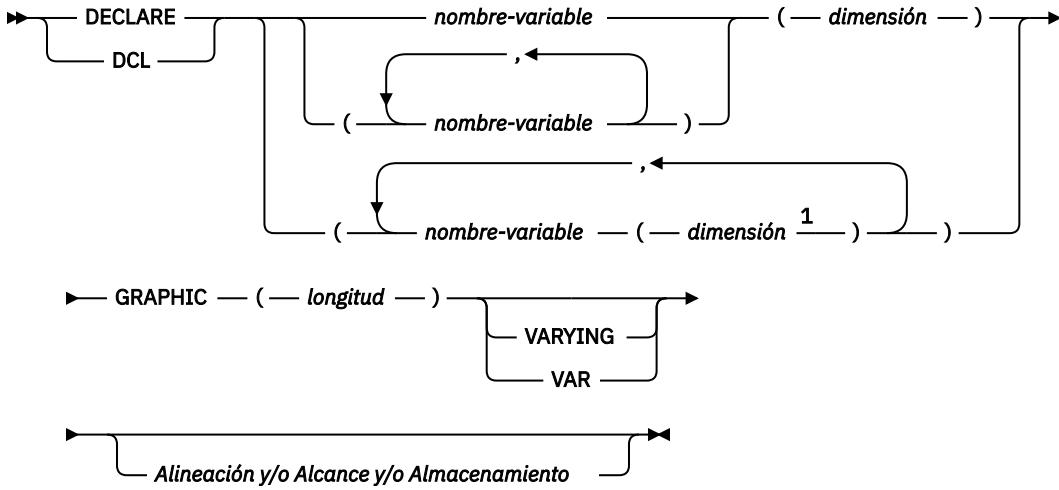
Ejemplo

El siguiente ejemplo muestra las declaraciones necesarias para recuperar 10 filas del número y nombre del departamento de la tabla de departamentos:

```
DCL DEPTNO(10) CHAR(3);          /* Array of ten CHAR(3) variables */  
DCL DEPTNAME(10) CHAR(29) VAR;    /* Array of ten VARCHAR(29) variables */
```

Matrices gráficas de variables de host

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host gráficas que no sean DBCLOB.

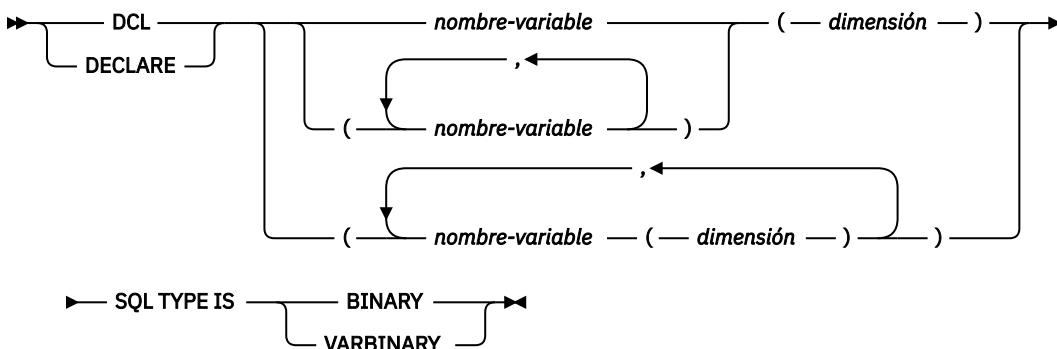


Notas:

¹ La dimensión debe ser una constante entera en el rango de 1 a 32767.

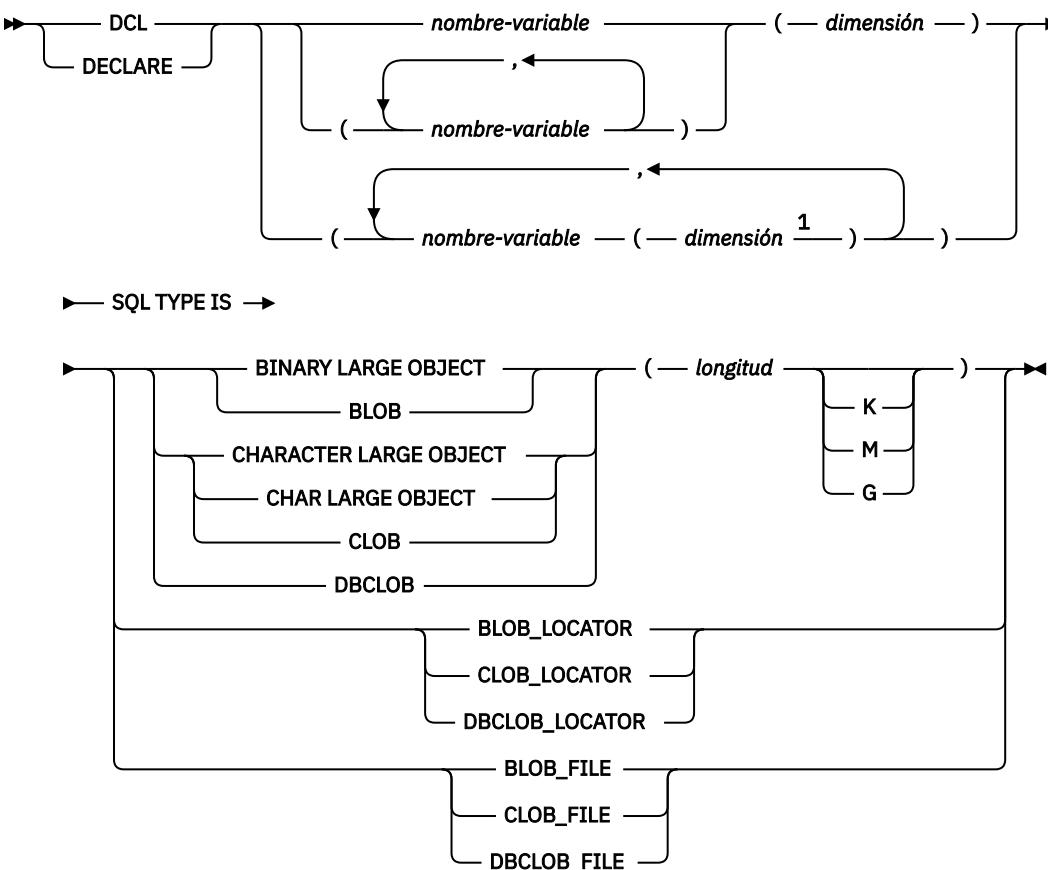
Matrices binarias de variables de host

El siguiente diagrama muestra la sintaxis para declarar matrices de variables binarias.



LOB, localizador y matrices de variables de referencia de archivos

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de referencia de archivo, localizadores y variables de host BLOB, CLOB y DBCLOB.

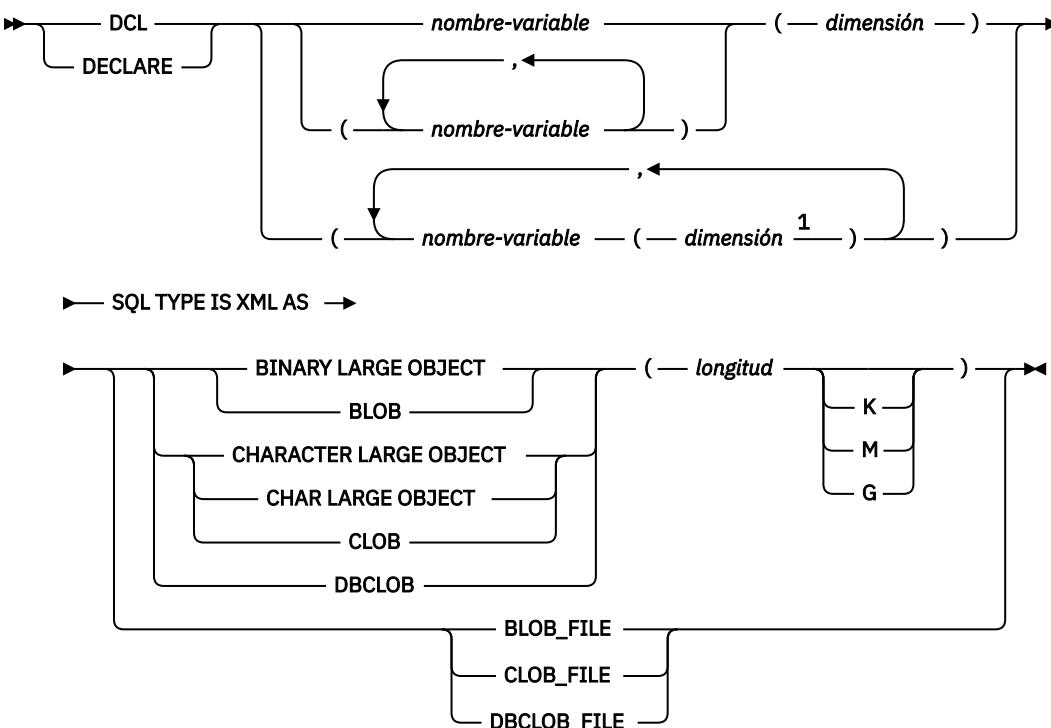


Notas:

¹ La dimensión debe ser una constante entera en el rango de 1 a 32767.

Variables de matriz de referencia de archivos y host XML

El siguiente diagrama muestra la sintaxis para declarar matrices de variables de host BLOB, CLOB y DBCLOB y matrices de variables de referencia de archivos para tipos de datos XML.

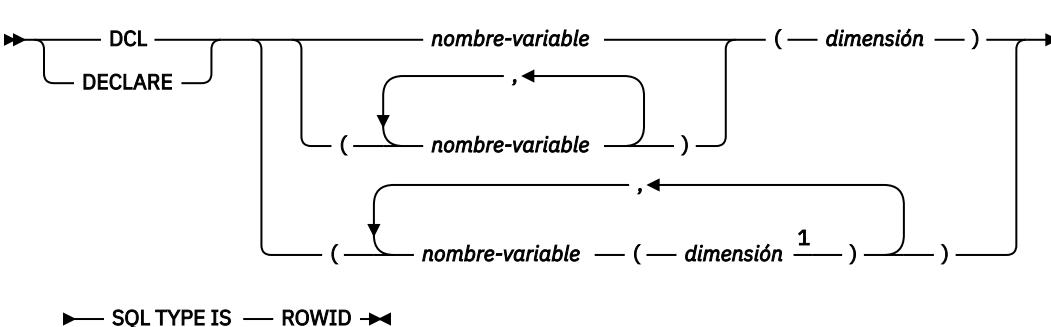


Notas:

¹ la dimensión debe ser una constante entera en el rango de 1 a 32767.

Matrices de variables ROWID

El siguiente diagrama muestra la sintaxis para declarar matrices de variables ROWID.



Notas:

¹ la dimensión debe ser una constante entera en el rango de 1 a 32767.

Conceptos relacionados

[Utilización de matrices de variables de host en sentencias SQL](#)

Utilice matrices de variables de lenguaje de host en sentencias de SQL incorporado para representar valores que el programa no conoce hasta que se ejecuta la consulta. Las matrices de variables de host son útiles para almacenar un conjunto de valores recuperados o para pasar un conjunto de valores que se van a insertar en una tabla.

[Matrices de variables de host](#)

Puede utilizar matrices de variables de host para pasar una matriz de datos entre Db2 y su aplicación. Una matriz de variables de host es una matriz de datos que se declara en el lenguaje de host para ser utilizada dentro de una instrucción SQL.

[Matrices de variables de lenguaje principal en PL/I, C, C++ y COBOL \(Db2 SQL\)](#)

[Tipos de datos numéricos \(Db2 SQL\)](#)

Tareas relacionadas

[Inserción de varias filas de datos desde matrices de variables de host](#)

Utilice las variables de host en la sentencia INSERT cuando no conozca al menos algunos de los valores que se han de insertar hasta que el programa se ejecuta.

[Almacenamiento de datos LOB en tablas de Db2](#)

Db2 maneja los datos LOB de manera diferente a otros tipos de datos. Como resultado, a veces es necesario realizar acciones adicionales al definir columnas LOB e insertar los datos LOB.

[Recuperación de varias filas de datos en matrices de variables de host](#)

Si sabe que la consulta devuelve varias filas, puede especificar matrices de variables de host para almacenar los valores de columna recuperados.

Estructuras de host en PL/I

Una estructura de host PL/I es una estructura de host que contiene niveles subordinados de escalares.

Puede utilizar el nombre de la estructura como notación abreviada para hacer referencia a la lista de escalares.

Requisitos: Las declaraciones de estructura de host en PL/I deben cumplir los siguientes requisitos:

- Las estructuras de host están limitadas a dos niveles.
- Debe terminar la variable de estructura host finalizando la declaración con un punto y coma, como en el siguiente ejemplo:

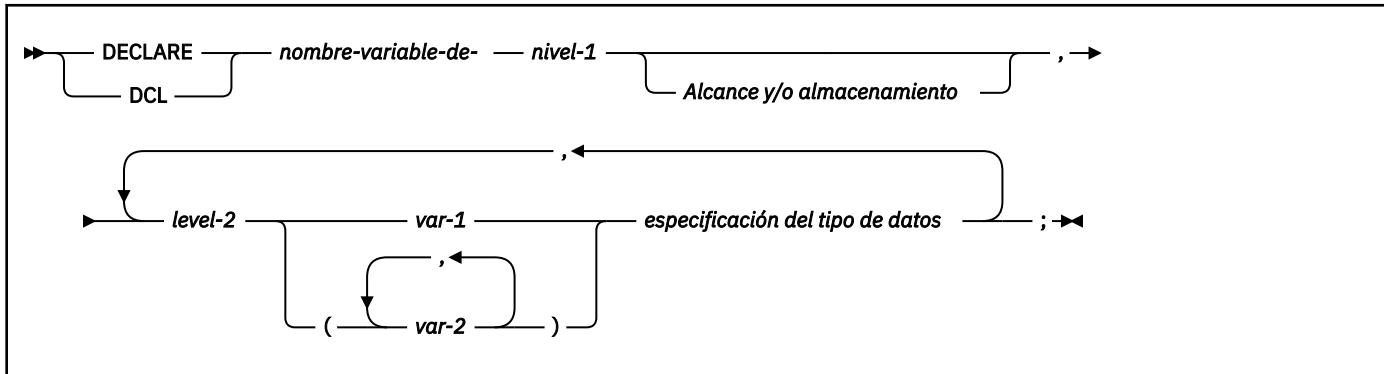
```
DCL 1 A,  
      2 B CHAR,  
      2 (C, D) CHAR;  
DCL (E, F) CHAR;
```

- Puede especificar atributos de variables de host en cualquier orden que sea aceptable para PL/I. Por ejemplo, BIN FIXED(31), BIN(31) FIXED y FIXED BIN(31) son todos aceptables.

Cuando haces referencia a una variable de host, puedes calificarla con un nombre de estructura. Por ejemplo, puede especificar STRUCTURE . FIELD.

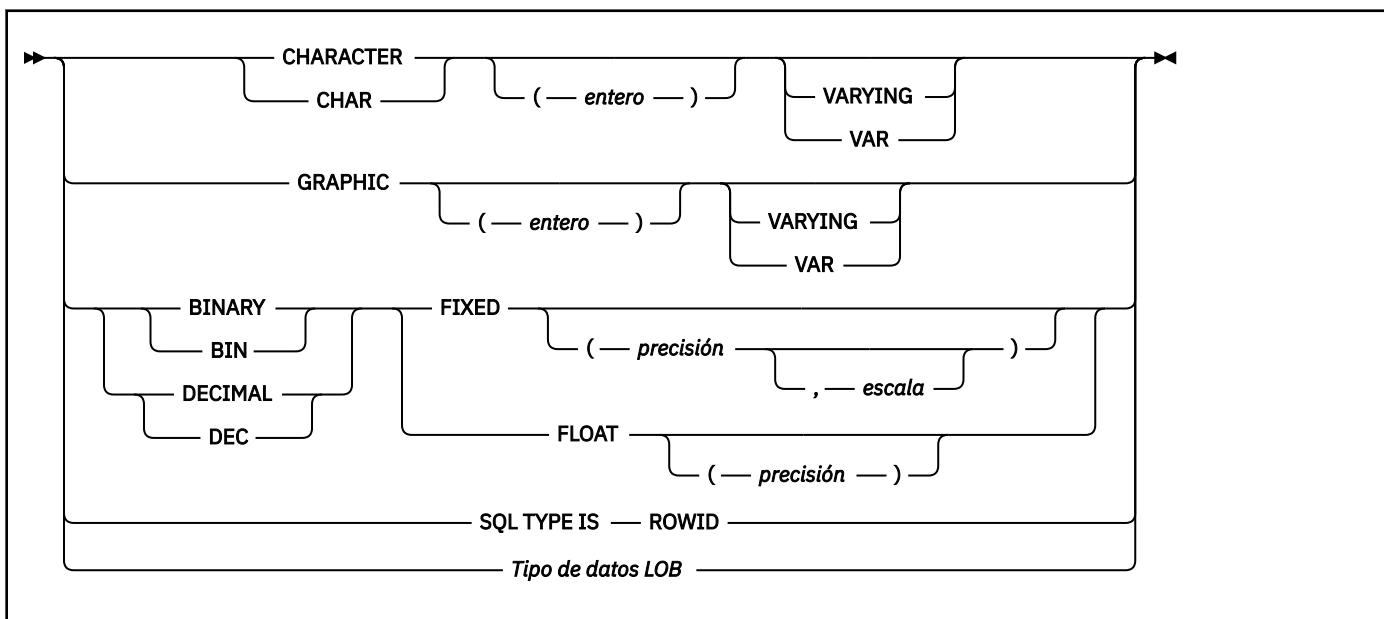
Estructuras host

El siguiente diagrama muestra la sintaxis para declarar estructuras de host.



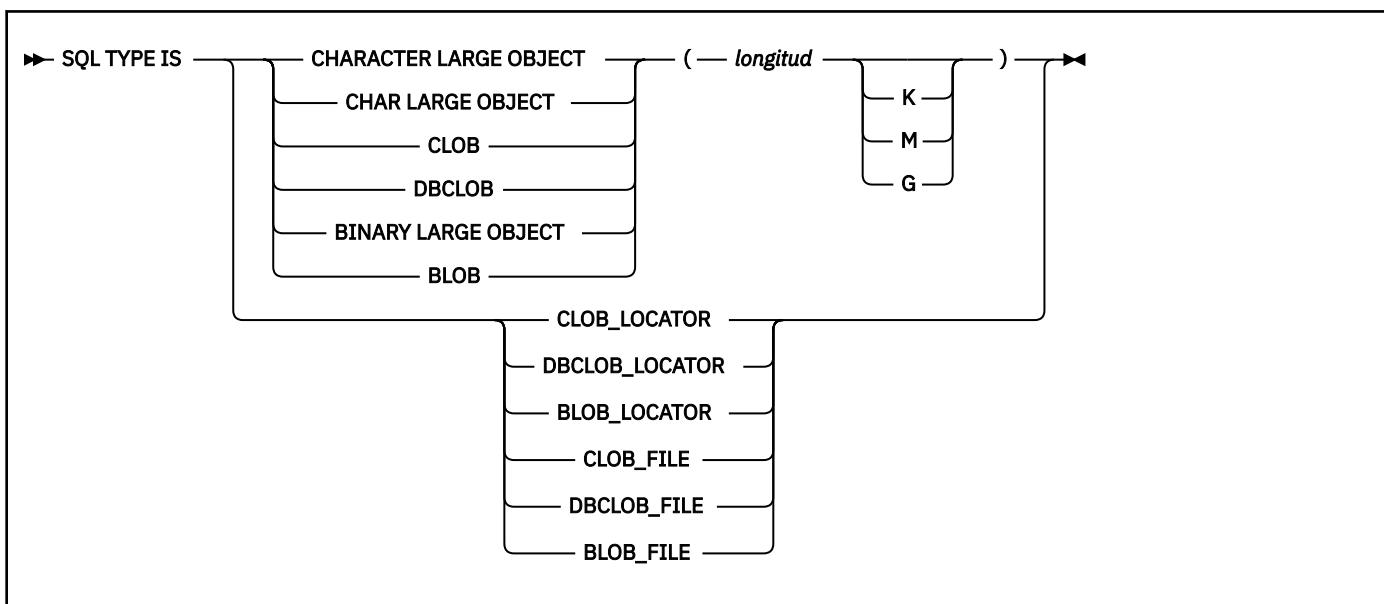
Tipos de datos

El siguiente diagrama muestra la sintaxis de los tipos de datos que se utilizan en las declaraciones de estructuras de host.



tipos de datos LOB

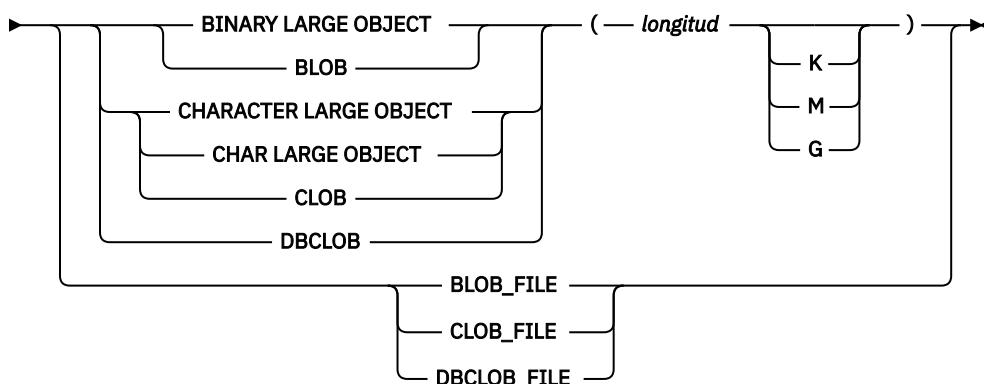
El siguiente diagrama muestra la sintaxis de los tipos de datos LOB que se utilizan en las declaraciones de estructuras de host.



Tipos de datos LOB para datos XML

El siguiente diagrama muestra la sintaxis de los tipos de datos LOB que se utilizan en las declaraciones de estructuras de host para datos XML.

► SQL TYPE IS XML AS →



Ejemplo

En el siguiente ejemplo, B es el nombre de una estructura host que contiene los escalares C1 y C2.

```
DCL 1 A,  
     2 B,  
     3 C1 CHAR(...),  
     3 C2 CHAR(...);
```

Conceptos relacionados

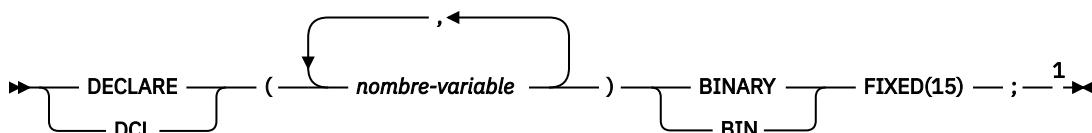
Estructuras host

Utilice las estructuras host para pasar un grupo de variables host entre Db2 y su aplicación.

Variables de indicador en PL/I

Una variable indicadora es un entero de 2 bytes (o un entero declarado como BIN FIXED(15)). Una matriz de variable indicadora es una matriz de enteros de 2 bytes. Declare las variables indicadoras de la misma forma que las variables host. Puede mezclar las declaraciones de dos tipos de variables.

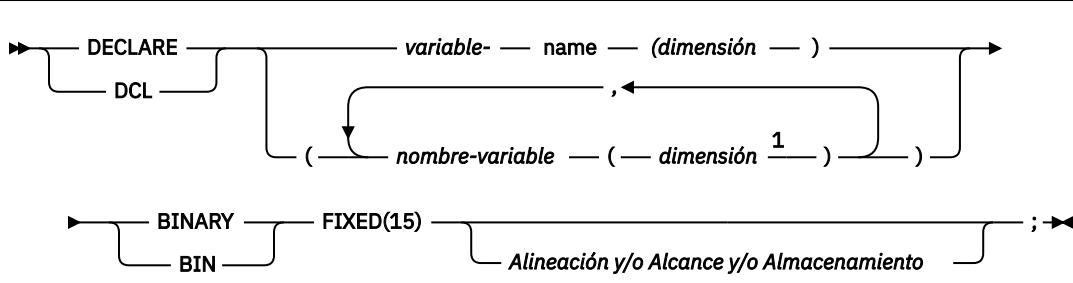
El siguiente diagrama muestra la sintaxis para declarar una variable indicadora en PL/I.



Notas:

¹ Puede especificar atributos de variables de host en cualquier orden que sea aceptable para PL/I. Por ejemplo, BIN FIXED(31), BIN(31) FIXED y FIXED BIN(31) son todos aceptables.

El siguiente diagrama muestra la sintaxis para declarar una matriz de indicadores en PL/I.



Notas:

¹ la dimensión debe ser una constante entera en el rango de 1 a 32767.

Ejemplo

El siguiente ejemplo muestra una sentencia FETCH con las declaraciones de las variables de host que se necesitan para la sentencia FETCH y sus variables indicadoras asociadas.

```
EXEC SQL FETCH CLS_CURSOR INTO :CLS_CD,
      :DAY :DAY_IND,
      :BGN :BGN_IND,
      :END :END_IND;
```

Puede declarar estas variables de la siguiente manera:

```
DCL  CLS_CD      CHAR(7);
DCL  DAY         BIN FIXED(15);
DCL  BGN         CHAR(8);
DCL  END         CHAR(8);
DCL  (DAY_IND,  BGN_IND,  END_IND)   BIN FIXED(15);
```

Conceptos relacionados

Variables de indicación, matrices y estructuras

Una variable de indicación se asocia con una variable host concreta. Cada variable de indicación contiene un valor entero pequeño que indica alguna información sobre la variable host asociada. Las estructuras y matrices de indicador tienen el mismo propósito para las estructuras y matrices de variables de host.

Tareas relacionadas

Inserción de valores nulos en columnas utilizando las matrices o variables indicadoras

Si necesita insertar valores nulos en una columna, utilizar una matriz o variable indicadora es una forma fácil de hacerlo. Una variable o matriz de indicador está asociada a una variable de host o matriz de variables de lenguaje de host determinada.

SQL equivalente y tipos de datos PL/I

Cuando declara variables host en los programas PL/I, el precompilador utiliza tipos de datos SQL equivalentes. Cuando recupere datos de un tipo de datos SQL concreto en una variable host, tendrá que asegurarse de que la variable host sea de un tipo de datos equivalente.

La siguiente tabla describe el tipo de datos SQL y los valores base SQLLEN y SQLTYPE que el precompilador utiliza para las variables de host en las sentencias SQL.

Tabla 114. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas PL/I

Tipo de datos de variable de host PL/I	SQLTYPE de la variable de host “1” en la página 765	SQLLEN de la variable de host	Tipos de datos SQL
BIN FIXED(<i>n</i>) $1 \leq n \leq 15$	500	2	SMALLINT
BIN FIXED(<i>n</i>) $16 \leq n \leq 31$	496	4	ENTERO
BIN FIJO(63)	492	8	BIGINT
DEC FIJO(<i>p,s</i>) $0 \leq p \leq 31$ y $0 \leq s \leq p$ “2” en la página 765	<i>p</i> en el byte 1, <i>s</i> en el byte 2		DECIMAL(<i>p,s</i>)
DEC FLOAT (<i>p</i>) donde $1 \leq p \leq 7$	996/997	4	DECFLOAT(16) “6” en la página 765
DEC FLOAT (<i>p</i>) donde $8 \leq p \leq 16$	996/997	8	DECFLOAT(16)

Tabla 114. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas PL/I (continuación)

Tipo de datos de variable de host PL/I	SQLTYPE de la variable de host “1” en la página 765	SQLLEN de la variable de host	Tipos de datos SQL
DEC FLOAT (p) donde $17 \leq p$	996/997	16	DECFLOAT(34)
BIN FLOAT(p) $1 \leq p \leq 21$	480	4	REAL o FLOAT(n) $1 \leq n \leq 21$
BIN FLOAT(p) $22 \leq p \leq 53$	480	8	DOBLE PRECISIÓN o FLOTANTE (n) $22 \leq n \leq 53$
DEC FLOAT (m) $1 \leq m \leq 6$	480	4	FLOAT (precisión simple)
DEC FLOAT (m) $7 \leq m \leq 16$	480	8	FLOAT (doble precisión)
CHAR (n)	452	n	CHAR (n)
CHAR(n) VARIABLE $1 \leq n \leq 255$	448	n	VARCHAR(n)
CHAR (n) VARIABLE $n > 255$	456	n	VARCHAR(n)
GRAPHIC (n)	468	n	GRAPHIC (n)
GRÁFICO VARIABLE(n)	464	n	VARGRAPHIC (n)
TIPO SQL ES BINARIO (n), $1 \leq n \leq 255$	912	n	BINARY(n)
TIPO SQL ES VARBINARIO (n), $1 \leq n \leq 32704$	908	n	VARBINARY(n)
TIPO SQL ES RESULT_SET_LOCATOR	972	4	Localizador de conjunto de resultados “3” en la página 765
SQL TYPE IS TABLE LIKE nombre-tabla AS LOCATOR	976	4	localizador de tablas “3” en la página 765
TIPO SQL ES BLOB_LOCATOR	960	4	localizador de BLOB “3” en la página 765
TIPO SQL ES CLOB_LOCATOR	964	4	localizador de CLOB “3” en la página 765
TIPO SQL ES DBCLOB_LOCATOR	968	4	localizador de DBCLOB “3” en la página 765
TIPO SQL ES BLOB(n) $1 \leq n \leq 2147483647$	404	n	BLOB (n)
TIPO SQL ES CLOB(n) $1 \leq n \leq 2147483647$	408	n	CLOB (n)
TIPO SQL ES DBCLOB(n) $1 \leq n \leq 1073741823$ “4” en la página 765	412	n	DBCLOB(n) “4” en la página 765
SQL TYPE IS XML AS BLOB(n)	404	0	XML
SQL TYPE IS XML AS CLOB(n)	408	0	XML
SQL TYPE IS XML AS DBCLOB(n)	412	0	XML

Tabla 114. Tipos de datos SQL, valores SQLLEN y valores SQLTYPE que el precompilador utiliza para variables de host en programas PL/I (continuación)

Tipo de datos de variable de host PL/I	SQLTYPE de la variable de host “1” en la página 765	SQLLEN de la variable de host	Tipos de datos SQL
TIPO SQL ES BLOB_FILE	916/917	267	Referencia de archivo BLOB “3” en la página 765
TIPO SQL ES CLOB_FILE	920/921	267	Referencia de archivo CLOB “3” en la página 765
EL TIPO SQL ES DBCLOB_FILE	924/925	267	Referencia de archivo DBCLOB “3” en la página 765
SQL TYPE IS XML AS BLOB_FILE	916/917	267	Referencia de archivo XML BLOB “3” en la página 765
SQL TYPE IS XML AS CLOB_FILE	920/921	267	Referencia de archivo XML CLOB “3” en la página 765
SQL TYPE IS XML AS DBCLOB_FILE	924/925	267	Referencia de archivo XML DBCLOB “3” en la página 765
EL TIPO SQL ES ROWID	904	40	ROWID
WIDECHAR(n)	468	n	GRÁFICO (s) “5” en la página 765
AMPLIA VARIACIÓN (n)	464	n	VARGRÁFICO(n) “5” en la página 765

La siguiente tabla muestra las variables de host PL/I equivalentes para cada tipo de datos SQL. Utilice esta tabla para determinar el tipo de datos PL/I para las variables de host que defina para recibir la salida de la base de datos. Por ejemplo, si recupera datos TIMESTAMP, puede definir una variable de tipo CHAR(n).

Esta tabla muestra las conversiones directas entre los tipos de datos SQL y los tipos de datos PL/I. Sin embargo, varios tipos de datos SQL son compatibles. Cuando realiza asignaciones o comparaciones de datos que tienen tipos de datos compatibles, Db2 convierte esos tipos de datos compatibles.

Tabla 115. PL/I variables equivalentes que puede utilizar al recuperar datos de un tipo de datos SQL concreto

Tipos de datos SQL	Variable de host PL/I equivalente	Observaciones
SMALLINT	BIN FIXED(n)	$1 \leq n \leq 15$
ENTERO	BIN FIXED(n)	$16 \leq n \leq 31$
BIGINT	BIN FIJO(63)	“7” en la página 765
DECIMAL(p,s) o NUMERIC(p,s)	Si $p < 16$: DEC FIXED(p) o DEC FIXED(p,s) Si $p > 15$, el compilador PL/I debe admitir variables decimales de 31 dígitos.	p es precisión; s es escala. $1 \leq p \leq 31$ y $0 \leq s \leq p$
DECFLOAT(16)	DEC FLOAT (p)	$1 \leq p \leq 7$
DECFLOAT(16)	DEC FLOAT (p)	$8 \leq p \leq 16$
DECFLOAT(34)	DEC FLOAT (p)	$17 \leq p$
REAL o FLOAT(n)	BIN FLOAT (p) o DEC FLOAT (m)	$1 \leq n \leq 21$, $1 \leq p \leq 21$ y $1 \leq m \leq 6$

Tabla 115. PL/I variables equivalentes que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host PL/I equivalente	Observaciones
DOBLE PRECISIÓN, DOBLE o <i>FLOAT(n)</i>	BIN FLOAT (<i>p</i>) o DEC FLOAT (<i>m</i>)	$22 \leq n \leq 53$, $22 \leq p \leq 53$ y $7 \leq m \leq 16$
CHAR (<i>n</i>)	CHAR (<i>n</i>)	$1 \leq n \leq 255$
VARCHAR(<i>n</i>)	CAR(<i>n</i>) VAR	
GRAPHIC (<i>n</i>)	GRAPHIC(<i>n</i>) o WIDECHAR(<i>n</i>) “2” en la página 765	<i>n</i> se refiere al número de caracteres de doble byte, no al número de bytes.
VARGRAPHIC (<i>n</i>)	GRAPHIC(<i>n</i>) VARYING o WIDECHAR(<i>n</i>) VARYING	<i>n</i> se refiere al número de caracteres de doble byte, no al número de bytes.
BINARY(<i>n</i>)	TIPO SQL ES BINARIO(<i>n</i>)	$1 \leq n \leq 255$
VARBINARY(<i>n</i>)	TIPO SQL ES VARBINARIO(<i>n</i>)	$1 \leq n \leq 32704$
FECHA	CHAR (<i>n</i>)	Si está utilizando una rutina de salida de fecha, esa rutina determina <i>n</i> ; de lo contrario, <i>n</i> debe ser al menos 10.
HORA	CHAR (<i>n</i>)	Si está utilizando una rutina de salida de tiempo, esa rutina determina <i>n</i> . De lo contrario, <i>n</i> debe ser al menos 6; para incluir segundos, <i>n</i> debe ser al menos 8.
TIMESTAMP	CHAR (<i>n</i>)	<i>n</i> debe tener al menos 19 años. Para incluir microsegundos, <i>n</i> debe ser 26; si <i>n</i> es menor que 26, la parte de microsegundos se trunca.
TIMESTAMP(0)	CHAR (<i>n</i>)	<i>n</i> debe tener al menos 19 años.
TIMESTAMP(<i>p</i>) <i>p</i> > 0	CHAR (<i>n</i>)	<i>n</i> debe tener al menos 19 años. Para incluir fracciones de segundo, <i>n</i> debe ser $20+x$, donde <i>x</i> es el número de fracciones de segundo que se van a incluir; si <i>x</i> es menor que <i>p</i> , se produce un truncamiento en la parte de las fracciones de segundo.
TIMESTAMP (0) WITH TIME ZONE	CAR(<i>n</i>) VAR	<i>n</i> debe tener al menos 25 años.
FECHA Y HORA (<i>p</i>) CON ZONA HORARIA	CAR(<i>n</i>) VAR	<i>n</i> debe tener al menos $26+p$.
Localizador de conjunto de resultados	TIPO SQL ES RESULT_SET_LOCATOR	Utilice este tipo de datos solo para recibir conjuntos de resultados. “3” en la página 765
Localizador de tablas	SQL TYPE IS TABLE LIKE <i>nombre-tabla</i> AS LOCATOR	Utilice este tipo de datos solo en una función definida por el usuario o en un procedimiento almacenado para recibir filas de una tabla de transición. “3” en la página 765

Tabla 115. PL/I variables equivalentes que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host PL/I equivalente	Observaciones
localizador de BLOB	TIPO SQL ES BLOB_LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas BLOB. “3” en la página 765 , “6” en la página 765 , “7” en la página 765
localizador de CLOB	TIPO SQL ES CLOB_LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas CLOB. “3” en la página 765 , “6” en la página 765 , “7” en la página 765
localizador de DBCLOB	TIPO SQL ES DBCLOB_LOCATOR	Utilice este tipo de datos solo para manipular datos en columnas DBCLOB. “3” en la página 765 , “6” en la página 765 , “7” en la página 765
BLOB (n)	TIPO SQL ES BLOB(n)	$1 \leq n \leq 2147483647$ “6” en la página 765 , “7” en la página 765
CLOB (n)	TIPO SQL ES CLOB(n)	$1 \leq n \leq 2147483647$ “6” en la página 765 , “7” en la página 765
DBCLOB (n)	TIPO SQL ES DBCLOB(n)	n es el número de caracteres de doble byte. $1 \leq n \leq 1073741823$ “5” en la página 765 , “7” en la página 765
XML	SQL TYPE IS XML AS BLOB(n)	$1 \leq n \leq 2147483647$
XML	SQL TYPE IS XML AS CLOB(n)	$1 \leq n \leq 2147483647$
XML	SQL TYPE IS XML AS DBCLOB(n)	n es el número de caracteres de doble byte. $1 \leq n \leq 1073741823$ “6” en la página 765
Referencia de archivo BLOB	TIPO SQL ES BLOB_FILE	Utilice este tipo de datos solo para manipular datos en columnas BLOB. “3” en la página 765 , “6” en la página 765 , “7” en la página 765
Referencia de archivo CLOB	TIPO SQL ES CLOB_FILE	Utilice este tipo de datos solo para manipular datos en columnas CLOB. “3” en la página 765 , “6” en la página 765 , “7” en la página 765
Referencia de archivo DBCLOB	EL TIPO SQL ES DBCLOB_FILE	Utilice este tipo de datos solo para manipular datos en columnas DBCLOB. “3” en la página 765 , “6” en la página 765 , “7” en la página 765
Referencia de archivo XML BLOB	SQL TYPE IS XML AS BLOB_FILE	Utilice este tipo de datos solo para manipular datos XML como archivos BLOB. “3” en la página 765
Referencia de archivo XML CLOB	SQL TYPE IS XML AS CLOB_FILE	Utilice este tipo de datos solo para manipular datos XML como archivos CLOB. “3” en la página 765
Referencia de archivo XML DBCLOB	SQL TYPE IS XML AS DBCLOB_FILE	Utilice este tipo de datos solo para manipular datos XML como archivos DBCLOB. “3” en la página 765

Tabla 115. PL/I variables equivalentes que puede utilizar al recuperar datos de un tipo de datos SQL concreto (continuación)

Tipos de datos SQL	Variable de host PL/I equivalente	Observaciones
ROWID	EL TIPO SQL ES ROWID	

Notas de la tabla:

Las siguientes notas se aplican como se indica a Tabla 114 en la página 760 y Tabla 115 en la página 762.

1. Si una variable de host incluye una variable de indicador, el valor SQLTYPE es el valor SQLTYPE base más 1.
2. Si $p=0$, Db2 lo interpreta como DECIMAL(31). Por ejemplo, Db2 interpreta un tipo de datos PL/I de DEC FIXED(0,0) como DECIMAL(31,0), que equivale al tipo de datos SQL de DECIMAL(31,0).
3. No utilice este tipo de datos como tipo de columna.
4. n es el número de caracteres de doble byte.
5. CCSID 1200 siempre se asigna a la variable de host de tipo WIDECHAR.
6. Las conversiones de tipo de datos solo pueden utilizarse si se utiliza el coprocesador Db2 , y se especifican las opciones del compilador PL/I FLOAT(DFP) y ARCH(7).
7. Especifique las siguientes opciones de compilación cuando compile su programa:
LIMITS(FIXEDBIN(63), FIXEDDEC(31)).

La siguiente tabla muestra las definiciones del lenguaje PL/I que se deben utilizar en los procedimientos almacenados PL/I y en las funciones definidas por el usuario, cuando los tipos de datos de los parámetros en las definiciones de rutina son LOB, ROWID o localizadores. Para otros tipos de datos de parámetros, las definiciones del lenguaje PL/I son las mismas que las del anterior e Tabla 115 en la página 762 .

Tabla 116. Declaraciones de lenguaje PL/I equivalentes para LOB, ROWID y localizadores en definiciones de rutinas definidas por el usuario

Tipo de datos SQL en la definición	PL/I
localizador de tablas	BIN FIXED(31)
localizador de BLOB	
localizador de CLOB	
localizador de DBCLOB	
BL OB (n)	<p>Si $n \leq 32767$:</p> <pre>01 var, 03 var_LENGTH BIN FIXED(31), 03 var_DATA CHAR(n);</pre> <p>Si $n > 32767$:</p> <pre>01 var, 02 var_LENGTH BIN FIXED(31), 02 var_DATA, 03 var_DATA1(n) CHAR(32767), 03 var_DATA2 CHAR(mod(n, 32767));</pre>

Tabla 116. Declaraciones de lenguaje PL/I equivalentes para LOB, ROWID y localizadores en definiciones de rutinas definidas por el usuario (continuación)

Tipo de datos SQL en la definición	PL/I
CLOB(n)	<p>Si n <= 32767:</p> <pre>01 var, 03 var_LENGTH BIN FIXED(31), 03 var_DATA CHAR(n);</pre> <p>Si n > 32767:</p> <pre>01 var, 02 var_LENGTH BIN FIXED(31), 02 var_DATA, 03 var_DATA1(n) CHAR(32767), 03 var_DATA2 CHAR(mod(n, 32767));</pre>
DBCL OB (n)	<p>Si n <= 16383:</p> <pre>01 var, 03 var_LENGTH BIN FIXED(31), 03 var_DATA GRAPHIC(n);</pre> <p>Si n > 16383:</p> <pre>01 var, 02 var_LENGTH BIN FIXED(31), 02 var_DATA, 03 var_DATA1(n) GRAPHIC(16383), 03 var_DATA2 GRAPHIC(mod(n, 16383));</pre>
ROWID	CHAR(40) VAR;

Conceptos relacionados

[Compatibilidad de SQL y tipos de datos de lenguaje](#)

Los tipos de datos de variable host que se utilizan en sentencias SQL deben ser compatibles con los tipos de datos de las columnas con las que tiene intención de utilizarlos.

[Declaraciones de variable de referencia de archivo LOB, variable host LOB y localizador LOB](#)

Cuando escribe aplicaciones para manipular datos LOB, necesita declarar variables hosto para contener los datos LOB o el localizador LOB. De lo contrario, debe declarar variables de referencia de archivo LOB para apuntar a datos LOB.

[Tipos de datos de variable host para datos XML en aplicaciones de SQL incorporado \(Db2 Programming for XML\)](#)

Aplicaciones REXX que emiten sentencias SQL

Puede codificar instrucciones SQL en un programa REXX siempre que pueda utilizar comandos REXX.

Db2 REXX Language Support admite todas las sentencias SQL dinámicas y las siguientes sentencias SQL estáticas:

- CALL

- CLOSE
- CONNECT
- DECLARE CURSOR
- DESCRIBE *la declaración o tabla preparada*
- DESCRIBE CURSOR
- DESCRIBE INPUT
- DESCRIBE PROCEDURE
- EXECUTE
- EXECUTE IMMEDIATE
- FETCH
- OPEN
- PREPARE
- CONEXIÓN de la LIBERACIÓN
- SET CONNECTION
- SET CURRENT PACKAGE PATH
- SET CURRENT PACKAGESET
- SET *host-variable* = FECHA ACTUAL
- SET *host-variable* = GRADO ACTUAL
- SET *host-variable* = CURRENT MEMBER
- SET *variable-host* = CURRENT PACKAGESET
- SET *host-variable* = CURRENT PATH
- SET *variable-host* = CURRENT SERVER
- SET *host-variable* = CURRENT SQLID
- SET *host-variable* = HORA ACTUAL
- SET *host-variable* = CURRENT TIMESTAMP
- SET *host-variable* = CURRENT TIMEZONE

Cada instrucción SQL en un programa REXX debe comenzar con EXECSQL, en mayúsculas, minúsculas o en mayúsculas y minúsculas. Uno de los siguientes elementos debe seguir a EXECSQL:

- Una instrucción SQL entre comillas simples o dobles.
- Variable REXX que contiene una instrucción SQL. La variable REXX no debe ir precedida de dos puntos.

Por ejemplo, puede utilizar cualquiera de los siguientes métodos para ejecutar la instrucción COMMIT en un programa REXX :

```
EXECSQL "COMMIT"
```

```
rexxvar="COMMIT"
EXECSQL rexxvar
```

Las siguientes sentencias dinámicas deben ejecutarse utilizando EXECUTE IMMEDIATE o PREPARE y EXECUTE bajo DSNREXX:

- DECLARE GLOBAL TEMPORARY TABLE
- SET CURRENT DEBUG MODE
- SET CURRENT DECFLOAT ROUNDING MODE
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
- SET CURRENT QUERY ACCELERATION
- SET CURRENT REFRESH AGE

- SET CURRENT ROUTINE VERSION
- SET SCHEMA

No puede ejecutar una instrucción SELECT, INSERT, UPDATE, MERGE o DELETE que contenga variables de host. En su lugar, debe ejecutar PREPARE en la instrucción, con marcadores de parámetros sustituidos por las variables del host, y luego usar las variables del host en una instrucción EXECUTE, OPEN o FETCH. Consulte “[Variables de host](#)” en la página 503 para obtener más información.

Una instrucción SQL sigue las reglas que se aplican a los comandos REXX. La instrucción SQL puede terminar opcionalmente con un punto y coma y puede ir entre comillas simples o dobles, como en el siguiente ejemplo:

```
'EXECSQL COMMIT';
```

Comentarios

No puede incluir comentarios REXX /* ... */ ni comentarios SQL -- dentro de las sentencias SQL. Sin embargo, puede incluir comentarios REXX en cualquier otro lugar del programa.

Delimitadores para instrucciones SQL

Delimite las sentencias SQL en el programa REXX precediendo la sentencia con EXECSQL. Si la declaración está en una cadena literal, escríbala entre comillas simples o dobles.

Continuación para instrucciones SQL

Las sentencias SQL que abarcan varias líneas siguen las reglas REXX para la continuación de sentencias. Puede dividir la declaración en varias cadenas, cada una de las cuales cabe en una línea, y separar las cadenas con comas o con operadores de concatenación seguidos de comas. Por ejemplo, cualquiera de las siguientes afirmaciones es válida:

```
EXECSQL ,
  "UPDATE DSN8C10.DEPT" ,
  "SET MGRNO = '000010'" ,
  "WHERE DEPTNO = 'D11'" 

"EXECSQL " ||
"  UPDATE DSN8C10.DEPT " ||
"  SET MGRNO = '000010'" ||
"  WHERE DEPTNO = 'D11'"
```

Incluir código

La instrucción EXECSQL INCLUDE no es válida para REXX. Por lo tanto, no puede incluir instrucciones SQL definidas externamente en un programa.

Márgenes

Al igual que los comandos REXX, las sentencias SQL pueden comenzar y terminar en cualquier lugar de una línea.

Puede utilizar cualquier nombre REXX válido que no termine con un punto como variable de host. Sin embargo, los nombres de las variables de host no deben comenzar con «SQL», «RDI», «DSN», «RXSQL» o «QRW». Los nombres de variables pueden tener como máximo 64 bytes.

Nulos

Un valor nulo REXX y un valor nulo SQL son diferentes. El lenguaje REXX tiene una cadena nula (una cadena de longitud 0) y una cláusula nula (una cláusula que contiene solo espacios en blanco y comentarios). El valor nulo SQL es un valor especial que se distingue de todos los valores no nulos y denota la ausencia de un valor. Asignar un valor nulo REXX a una columna de tipo "Db2" no hace que el valor de la columna sea nulo.

Etiquetas de extracto

Puede preceder una instrucción SQL con una etiqueta, de la misma manera que etiqueta los comandos REXX.

Manejo de códigos de error de SQL

Las aplicaciones de Rexx pueden solicitar más información sobre errores SQL a Db2. Para obtener más información, consulte “[Gestión de códigos de error de SQL en aplicaciones REXX](#)” en la página 793.

Tareas relacionadas

[Descripción general de las aplicaciones de programación que acceden a datos de Db2 for z/OS](#)
Las aplicaciones que interactúan con Db2, en primer lugar se deben conectar a Db2. Entonces pueden leer, añadir o modificar datos o manipular objetos de Db2.

[Inclusión de SQL dinámico en el programa](#)

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

[Manejo de códigos de error de SQL](#)

Los programas de aplicación pueden solicitar más información sobre los códigos de error SQL en Db2.

[Establecimiento de límites para el uso de recursos de sistema utilizando el recurso de límite de recursos \(Db2 Performance\)](#)

Ejemplos de programación de REXX

Puede escribir programas de Db2 en REXX. Estos programas pueden acceder a un subsistema Db2 local o remoto y pueden ejecutar sentencias de SQL estático o dinámico. Esta información contiene varios ejemplos de programación de ese tipo.

Para preparar y ejecutar estas aplicaciones, utilice el JCL de *prefijo.SDSNSAMP* como modelo para su JCL.

Referencia relacionada

[Ejemplos de programación de Assembler, C, C++, COBOL, PL/I y REXX \(Db2 Programming samples\)](#)

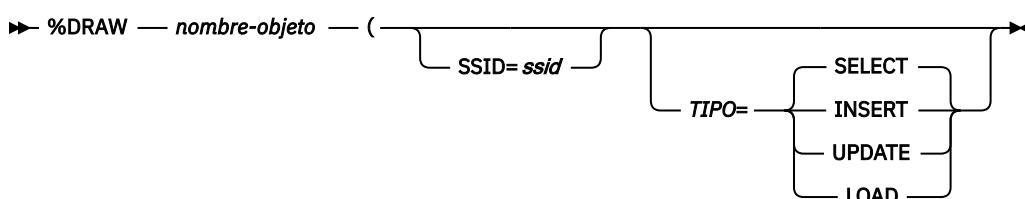
[Db2 for z/OS Cambio](#)

Ejemplo de aplicación REXX de Db2

Puede utilizar una aplicación REXX para aceptar un nombre de tabla como entrada y producir una instrucción SQL SELECT, INSERT o UPDATE o una instrucción de utilidad LOAD para la tabla especificada como salida.

El siguiente ejemplo muestra una aplicación REXX de Db2 a completa llamada DRAW. DRAW debe invocarse desde la línea de comandos de una sesión de edición de ISPF. DRAW toma una tabla o nombre de vista como entrada y produce una instrucción SQL SELECT, INSERT o UPDATE o una instrucción de control de utilidad LOAD que incluye las columnas de la tabla como salida.

Sintaxis DRAW:



Parámetros DRAW:

nombre-objeto

El nombre de la tabla o vista para la que DRAW construye una instrucción SQL o una instrucción de control de utilidad. El nombre puede constar de una, dos o tres partes. La tabla o vista a la que se refiere el *nombre del objeto* debe existir antes de que DRAW pueda ejecutarse.

nombre-objeto es un parámetro obligatorio.

SSID=ssid

Especifica el nombre del subsistema de Db2 e local.

S puede utilizarse como abreviatura de SSID.

Si invoca DRAW desde la línea de comandos de la sesión de edición en SPUFI, *SSID=ssid* es un parámetro opcional. DRAW utiliza el ID del subsistema del panel de valores predeterminados de DB2I.

TYPE=tipo-de-operación

El tipo de declaración que DRAW construye.

T puede utilizarse como abreviatura de TYPE.

operation-type tiene uno de los siguientes valores:

SELECT

Crea una sentencia SELECT en la que la tabla de resultados contiene todas las columnas de *nombre-objeto*.

S puede utilizarse como abreviatura de SELECT.

INSERT

Crea una plantilla para una instrucción INSERT que inserta valores en todas las columnas de *nombre-objeto*. La plantilla contiene comentarios que indican dónde puede colocar el usuario los valores de las columnas.

Se puede utilizar como abreviatura de INSERT.

UPDATE

Crea una plantilla para una instrucción UPDATE que actualiza las columnas de *nombre-objeto*.

La plantilla contiene comentarios que indican dónde puede colocar el usuario los valores de las columnas y calificar la operación de actualización para las filas seleccionadas.

U puede utilizarse como abreviatura de UPDATE (ACTUALIZAR).

LOAD

Crea una plantilla para una sentencia de control de utilidad LOAD para *nombre-objeto*.

L puede utilizarse como abreviatura de LOAD.

TYPE=operation-type es un parámetro opcional. El valor predeterminado es TYPE=SELECT.

Conjuntos de datos DRAW:

Editar conjunto de datos

El conjunto de datos desde el que emite el comando DRAW cuando se encuentra en una sesión de edición e ISPF. Si emite el comando DRAW desde una sesión SPUFI, este conjunto de datos es el conjunto de datos que especifica en el campo 1 del panel principal SPUFI (DSNEP01). El resultado del comando DRAW va a este conjunto de datos.

Códigos de devolución de DRAW:

Código de retorno

Significado

0

Realización satisfactoria.

12

Se produjo un error cuando DRAW editó el archivo de entrada.

20

Se ha producido uno de los errores siguientes:

- No se han especificado parámetros de entrada.
- Uno de los parámetros de entrada no era válido.
- Se produjo un error SQL al generar la sentencia de salida.

Ejemplos de invocación de DRAW:

Generar una instrucción SELECT para la tabla DSN8C10.EMP en el subsistema local. Utilice el ID de subsistema predeterminado de DB2I.

La invocación de DRAW es:

```
DRAW DSN8C10.EMP (TYPE=SELECT)
```

La salida es:

```
SELECT "EMPNO" , "FIRSTNME" , "MIDINIT" , "LASTNAME" , "WORKDEPT" ,
      "PHONENO" , "HIREDATE" , "JOB" , "EDLEVEL" , "SEX" , "BIRTHDATE" ,
      "SALARY" , "BONUS" , "COMM"
   FROM DSN8C10.EMP
```

Generar una plantilla para una instrucción INSERT que inserte valores en la tabla DSN8C10.EMP en la ubicación SAN_JOSE. El ID del subsistema local es DSN.

La invocación de DRAW es:

```
DRAW SAN_JOSE.DSN8C10.EMP (TYPE=INSERT SSID=DSN)
```

La salida es:

```
INSERT INTO SAN_JOSE.DSN8C10.EMP ( "EMPNO" , "FIRSTNME" , "MIDINIT" ,
      "LASTNAME" , "WORKDEPT" , "PHONENO" , "HIREDATE" , "JOB" ,
      "EDLEVEL" , "SEX" , "BIRTHDATE" , "SALARY" , "BONUS" , "COMM" )
VALUES (
-- ENTER VALUES BELOW          COLUMN NAME      DATA TYPE
, -- EMPNO                   CHAR(6) NOT NULL
, -- FIRSTNME                VARCHAR(12) NOT NULL
, -- MIDINIT                  CHAR(1) NOT NULL
, -- LASTNAME                 VARCHAR(15) NOT NULL
, -- WORKDEPT                 CHAR(3)
, -- PHONENO                  CHAR(4)
, -- HIREDATE                 DATE
, -- JOB                      CHAR(8)
, -- EDLEVEL                  SMALLINT
, -- SEX                      CHAR(1)
, -- BIRTHDATE                DATE
, -- SALARY                   DECIMAL(9,2)
, -- BONUS                    DECIMAL(9,2)
) -- COMM                     DECIMAL(9,2)
```

Generar una plantilla para una instrucción UPDATE que actualice los valores de la tabla DSN8C10.EMP. El ID del subsistema local es DSN.

La invocación de DRAW es:

```
DRAW DSN8C10.EMP (TYPE=UPDATE SSID=DSN)
```

La salida es:

```
UPDATE DSN8C10.EMP SET
-- COLUMN NAME      ENTER VALUES BELOW      DATA TYPE
"EMPNO"=          -- CHAR(6) NOT NULL
, "FIRSTNME"=      -- VARCHAR(12) NOT NULL
, "MIDINIT"=       -- CHAR(1) NOT NULL
, "LASTNAME"=      -- VARCHAR(15) NOT NULL
, "WORKDEPT"=      -- CHAR(3)
, "PHONENO"=       -- CHAR(4)
, "HIREDATE"=      -- DATE
, "JOB"=           -- CHAR(8)
, "EDLEVEL"=       -- SMALLINT
, "SEX"=            -- CHAR(1)
, "BIRTHDATE"=     -- DATE
, "SALARY"=         -- DECIMAL(9,2)
, "BONUS"=          -- DECIMAL(9,2)
, "COMM"=           -- DECIMAL(9,2)
WHERE
```

Generar una sentencia de control LOAD para cargar valores en la tabla DSN8C10.EMP. El ID del subsistema local es DSN.

La invocación del sorteo es:

```
DRAW DSN8C10.EMP (TYPE=LOAD SSID=DSN
```

La salida es:

```
LOAD DATA INDDN SYSREC INTO TABLE DSN8C10.EMP
( "EMPNO"           POSITION(   1) CHAR(6)
, "FIRSTNME"        POSITION(   8) VARCHAR
, "MIDINIT"         POSITION(  21) CHAR(1)
, "LASTNAME"        POSITION(  23) VARCHAR
, "WORKDEPT"        POSITION(  39) CHAR(3)
, "NULLIF(39)='?'"
, "PHONENO"         POSITION(  43) CHAR(4)
, "NULLIF(43)='?'"
, "HIREDATE"        POSITION(  48) DATE EXTERNAL
, "NULLIF(48)='?'"
, "JOB"              POSITION(  59) CHAR(8)
, "NULLIF(59)='?'"
, "EDLEVEL"          POSITION(  68) SMALLINT
, "NULLIF(68)='?'"
, "SEX"              POSITION(  71) CHAR(1)
, "NULLIF(71)='?'"
, "BIRTHDATE"        POSITION(  73) DATE EXTERNAL
, "NULLIF(73)='?'"
, "SALARY"           POSITION(  84) DECIMAL EXTERNAL(9,2)
, "NULLIF(84)='?'"
, "BONUS"            POSITION(  90) DECIMAL EXTERNAL(9,2)
, "NULLIF(90)='?'"
, "COMM"             POSITION(  96) DECIMAL EXTERNAL(9,2)
, "NULLIF(96)='?'"
)
```

Código fuente de DRAW:

```
/* REXX ****
L1 = WHEREAMI()
/*
DRAW creates basic SQL queries by retrieving the description of a
table. You must specify the name of the table or view to be queried.
You can specify the type of query you want to compose. You might need
to specify the name of the DB2 subsystem.
>>>DRAW----tablename-----|-----><
      | -(-|Ssid=subsystem-name-|-|
      |     +Select-+
      | -Type=|-Insert-|-|
      |     | -Update-|
      |     +--Load--+
Ssid=subsystem-name
    subsystem-name specified the name of a DB2 subsystem.
Select
    Composes a basic query for selecting data from the columns of a
    table or view. If TYPE is not specified, SELECT is assumed.
    Using SELECT with the DRAW command produces a query that would
    retrieve all rows and all columns from the specified table. You
    can then modify the query as needed.
    A SELECT query of EMP composed by DRAW looks like this:
SELECT "EMPNO" , "FIRSTNME" , "MIDINIT" , "LASTNAME" , "WORKDEPT" ,
       "PHONENO" , "HIREDATE" , "JOB" , "EDLEVEL" , "SEX" , "BIRTHDATE" ,
       "SALARY" , "BONUS" , "COMM"
FROM DSN8C10.EMP
    If you include a location qualifier, the query looks like this:
SELECT "EMPNO" , "FIRSTNME" , "MIDINIT" , "LASTNAME" , "WORKDEPT" ,
       "PHONENO" , "HIREDATE" , "JOB" , "EDLEVEL" , "SEX" , "BIRTHDATE" ,
       "SALARY" , "BONUS" , "COMM"
FROM STLEC1.DSN8C10.EMP
```

To use this SELECT query, type the other clauses you need. If you are selecting from more than one table, use a DRAW command for each table name you want represented.

Insert

Composes a basic query to insert data into the columns of a table or view.

The following example shows an INSERT query of EMP that DRAW composed:

```
INSERT INTO DSN8C10.EMP ( "EMPNO" , "FIRSTNME" , "MIDINIT" , "LASTNAME" ,
                           "WORKDEPT" , "PHONENO" , "HIREDATE" , "JOB" , "EDLEVEL" , "SEX" ,
```

```

        "BIRTHDATE" , "SALARY" , "BONUS" , "COMM" )
VALUES (
-- ENTER VALUES BELOW      COLUMN NAME      DATA TYPE
        , -- EMPNO          CHAR(6) NOT NULL
        , -- FIRSTNAME     VARCHAR(12) NOT NULL
        , -- MIDINIT        CHAR(1) NOT NULL
        , -- LASTNAME       VARCHAR(15) NOT NULL
        , -- WORKDEPT      CHAR(3)
        , -- PHONENO        CHAR(4)
        , -- HIREDATE       DATE
        , -- JOB            CHAR(8)
        , -- EDLEVEL        SMALLINT
        , -- SEX             CHAR(1)
        , -- BIRTHDATE      DATE
        , -- SALARY          DECIMAL(9,2)
        , -- BONUS           DECIMAL(9,2)
        ) -- COMM           DECIMAL(9,2)

```

To insert values into EMP, type values to the left of the column names.

Update

Composes a basic query to change the data in a table or view.
The following example shows an UPDATE query of EMP composed by DRAW:

```

UPDATE DSN8C10.EMP  SET
-- COLUMN NAME      ENTER VALUES BELOW      DATA TYPE
"EMPNO"=          -- CHAR(6) NOT NULL
,"FIRSTNAME"=      -- VARCHAR(12) NOT NULL
,"MIDINIT"=        -- CHAR(1) NOT NULL
,"LASTNAME"=       -- VARCHAR(15) NOT NULL
,"WORKDEPT"=       -- CHAR(3)
,"PHONENO"=        -- CHAR(4)
,"HIREDATE"=       -- DATE
,"JOB"=            -- CHAR(8)
,"EDLEVEL"=        -- SMALLINT
,"SEX"=             -- CHAR(1)
,"BIRTHDATE"=      -- DATE
,"SALARY"=          -- DECIMAL(9,2)
,"BONUS"=           -- DECIMAL(9,2)
,"COMM"=            -- DECIMAL(9,2)

```

WHERE

To use this UPDATE query, type the changes you want to make to the right of the column names, and delete the lines you do not need. Be sure to complete the WHERE clause.

Load

Composes a load statement to load the data in a table.
The following example shows a LOAD statement of EMP composed by DRAW:

```

LOAD DATA INDDN SYSREC INTO TABLE DSN8C10 .EMP
( "EMPNO"           POSITION(   1) CHAR(6)
, "FIRSTNAME"       POSITION(   8) VARCHAR
, "MIDINIT"         POSITION(  21) CHAR(1)
, "LASTNAME"        POSITION(  23) VARCHAR
, "WORKDEPT"        POSITION(  39) CHAR(3)
              NULLIF( 39)='?'
, "PHONENO"         POSITION(  43) CHAR(4)
              NULLIF( 43)='?'
, "HIREDATE"        POSITION(  48) DATE EXTERNAL
              NULLIF( 48)='?'
, "JOB"              POSITION(  59) CHAR(8)
              NULLIF( 59)='?'
, "EDLEVEL"          POSITION(  68) SMALLINT
              NULLIF( 68)='?'
, "SEX"              POSITION(  71) CHAR(1)
              NULLIF( 71)='?'
, "BIRTHDATE"        POSITION(  73) DATE EXTERNAL
              NULLIF( 73)='?'
, "SALARY"           POSITION(  84) DECIMAL EXTERNAL(9,2)
              NULLIF( 84)='?'
, "BONUS"            POSITION(  90) DECIMAL EXTERNAL(9,2)
              NULLIF( 90)='?'
, "COMM"             POSITION(  96) DECIMAL EXTERNAL(9,2)
              NULLIF( 96)='?'
)

```

To use this LOAD statement, type the changes you want to make, and delete the lines you do not need.

*/

L2 = WHEREAMI()

```

/* TRACE ?R */  

/*******/  

Address ISPEXEC  

"ISREDIT MACRO (ARGS) NOPROCESS"  

If ARGS = "" Then  

Do  

  Do I = L1+2 To L2-2;Say SourceLine(I);End  

  Exit (20)  

End  

Parse Upper Var Args Table "("Parms  

Parms = Translate(Parms," ",",")  

Type = "SELECT" /* Default */  

SSID = "" /* Default */  

"VGET (DSNEOV01)"  

If RC = 0 Then SSID = DSNEOV01  

If (Parms <> "") Then  

Do Until(Parms = "")  

Parse VarParms Var "=" ValueParms  

  If Var = "T" | Var = "TYPE" Then Type = Value  

  Else  

    If Var = "S" | Var = "SSID" Then SSID = Value  

  Else  

    Exit (20)  

End  

"CONTROL ERRORS RETURN"  

"ISREDIT (LEFTBND,RIGHTBND) = BOUNDS"  

"ISREDIT (LRECL) = DATA_WIDTH" /*LRECL*/  

BndSize = RightBnd - LeftBnd + 1  

If BndSize > 72 Then BndSize = 72  

"ISREDIT PROCESS DEST"  

Select  

  When rc = 0 Then  

    'ISREDIT (ZDEST) = LINENUM .ZDEST'  

  When rc <= 8 Then /* No A or B entered */  

    Do  

      zedmsg = 'Enter "A"/"B" line cmd'  

      zedlmsg = 'DRAW requires an "A" or "B" line command'  

      'SETMSG MSG(ISRZ001)'  

      Exit 12  

    End  

  When rc < 20 Then /* Conflicting line commands - edit sets message */  

    Exit 12  

  When rc = 20 Then  

    zdest = 0  

  Otherwise  

    Exit 12  

End

```

```

SQLTYPE. = "UNKNOWN TYPE"  

VCHTYPE = 448; SQLTYPES.VCHTYPE = 'VARCHAR'  

CHTYPE = 452; SQLTYPES.CHTYPE = 'CHAR'  

LVCHTYPE = 456; SQLTYPES.LVCHTYPE = 'VARCHAR'  

VGRTYP = 464; SQLTYPES.VGRTYP = 'VARGRAPHIC'  

GRTYP = 468; SQLTYPES.GRTYP = 'GRAPHIC'  

LVRGRTYP = 472; SQLTYPES.LVRGRTYP = 'VARGRAPHIC'  

FLOTYP = 480; SQLTYPES.FLOTYP = 'FLOAT'  

DCTYP = 484; SQLTYPES.DCTYP = 'DECIMAL'  

INTYP = 496; SQLTYPES.INTYP = 'INTEGER'  

SMITYP = 500; SQLTYPES.SMITYP = 'SMALLINT'  

DATYP = 384; SQLTYPES.DATYP = 'DATE'  

TITYP = 388; SQLTYPES.TITYP = 'TIME'  

TSTYP = 392; SQLTYPES.TSTYP = 'TIMESTAMP'  

Address TSO "SUBCOM DSNREXX" /* HOST CMD ENV AVAILABLE? */  

IF RC THEN /* NO, LET'S MAKE ONE */  

  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX') /* ADD HOST CMD ENV */  

Address DSNREXX "CONNECT" SSID  

If SQLCODE ^= 0 Then Call SQLCA  

Address DSNREXX "EXECSQL DESCRIBE TABLE :TABLE INTO :SQLDA"  

If SQLCODE ^= 0 Then Call SQLCA  

Address DSNREXX "EXECSQL COMMIT"  

Address DSNREXX "DISCONNECT"  

If SQLCODE ^= 0 Then Call SQLCA  

Select  

  When (Left(Type,1) = "S") Then  

    Call DrawSelect  

  When (Left(Type,1) = "I") Then  

    Call DrawInsert  

  When (Left(Type,1) = "U") Then  

    Call DrawUpdate  

  When (Left(Type,1) = "L") Then  

    Call DrawLoad

```

```

    Otherwise EXIT (20)
End
Do I = LINE.0 To 1 By -1
  LINE = COPIES(" ",LEFTBND-1)||LINE.I
  'ISREDIT LINE_AFTER 'zdest' = DATALINE (Line)'
End
line1 = zdest + 1
'ISREDIT CURSOR = 'line1 0
Exit

/*********************************************
WHEREAMI:; RETURN SIGL
/*********************************************
/* Draw SELECT */                                           */
/********************************************* */

DrawSelect:
  Line.0 = 0
  Line = "SELECT"
  Do I = 1 To SQLDA.SQLD
    If I > 1 Then Line = Line ','
    ColName = "'SQLDA.I.SQLNAME'"
    Null = SQLDA.I.SQLTYPE//2
    If Length(Line)+Length(ColName)+LENGTH(" ,") > BndSize THEN
      Do
        L = Line.0 + 1; Line.0 = L
        Line.L = Line
        Line = " "
      End
      Line = Line ColName
    End I
    If Line ^= "" Then
      Do
        L = Line.0 + 1; Line.0 = L
        Line.L = Line
        Line = " "
      End
      L = Line.0 + 1; Line.0 = L
      Line.L = "FROM" TABLE
      Return
    /* Draw INSERT */                                         */
  /* Draw INSERT */                                         */
DrawInsert:
  Line.0 = 0
  Line = "INSERT INTO" TABLE "("
  Do I = 1 To SQLDA.SQLD
    If I > 1 Then Line = Line ','
    ColName = "'SQLDA.I.SQLNAME'"
    If Length(Line)+Length(ColName) > BndSize THEN
      Do
        L = Line.0 + 1; Line.0 = L
        Line.L = Line
        Line = " "
      End
      Line = Line ColName
      If I = SQLDA.SQLD Then Line = Line ')'
    End I
    If Line ^= "" Then
      Do
        L = Line.0 + 1; Line.0 = L
        Line.L = Line
        Line = " "
      End
      L = Line.0 + 1; Line.0 = L
      Line.L = "VALUES (""
      L = Line.0 + 1; Line.0 = L
      Line.L =
      "-- ENTER VALUES BELOW          COLUMN NAME          DATA TYPE"
  Do I = 1 To SQLDA.SQLD
    If SQLDA.SQLD > 1 & I < SQLDA.SQLD Then
      Line = ", --"
    Else
      Line = ") --"
    Line = Line Left(SQLDA.I.SQLNAME,18)
    Type = SQLDA.I.SQLTYPE
    Null = Type//2
    If Null Then Type = Type - 1
    Len = SQLDA.I.SQLLEN
    Prcsn = SQLDA.I.SQLLEN.SQLPRECISION

```

```

Scale = SQLDA.I.SQLLEN.SQLSCALE
Select
When (Type = CHTYPE ,
|Type = VCHTYPE ,
|Type = LVCHTYPE ,
|Type = GRTYP ,
|Type = VRGRTYP
|Type = LVGRTYP ) THEN
  Type = SQLTYPES.Type"("STRIP(LEN)")"
When (Type = FLOTYPE ) THEN
  Type = SQLTYPES.Type"("STRIP((LEN*4)-11) ")"
When (Type = DCTYPE ) THEN
  Type = SQLTYPES.Type"("STRIP(PRCSN)","STRIP(SCALE))"
Otherwise
  Type = SQLTYPES.Type
End
Line = Line Type
If Null = 0 Then
  Line = Line "NOT NULL"
  L = Line.0 + 1; Line.0 = L
  Line.L = Line
End I
Return

```

```

/*********************************************
/* Draw UPDATE
/*********************************************
DrawUpdate:
  Line.0 = 1
  Line.1 = "UPDATE" TABLE "SET"
  L = Line.0 + 1; Line.0 = L
  Line.L =
  "-- COLUMN NAME           ENTER VALUES BELOW      DATA TYPE"
Do I = 1 To SQLDA.SQLD
  If I = 1 Then
    Line = " "
  Else
    Line = ", "
  Line = Line Left(''''SQLDA.I.SQLNAME'=' ,21)
  Line = Line Left(" ",20)
  Type = SQLDA.I.SQLTYPE
  Null = Type//2
  If Null Then Type = Type - 1
  Len = SQLDA.I.SQLLEN
  Prcsn = SQLDA.I.SQLLEN.SQLPRECISION
  Scale = SQLDA.I.SQLLEN.SQLSCALE
  Select
    When (Type = CHTYPE ,
    |Type = VCHTYPE ,
    |Type = LVCHTYPE ,
    |Type = GRTYP ,
    |Type = VRGRTYP
    |Type = LVGRTYP ) THEN
      Type = SQLTYPES.Type"("STRIP(LEN)")"
    When (Type = FLOTYPE ) THEN
      Type = SQLTYPES.Type"("STRIP((LEN*4)-11) ")"
    When (Type = DCTYPE ) THEN
      Type = SQLTYPES.Type"("STRIP(PRCSN)","STRIP(SCALE))"
    Otherwise
      Type = SQLTYPES.Type
  End
  Line = Line "--" Type
  If Null = 0 Then
    Line = Line "NOT NULL"
    L = Line.0 + 1; Line.0 = L
    Line.L = Line
  End I
  L = Line.0 + 1; Line.0 = L
  Line.L = "WHERE"
Return

```

```

/*********************************************
/* Draw LOAD
/*********************************************
DrawLoad:
  Line.0 = 1
  Line.1 = "LOAD DATA INDDN SYSREC INTO TABLE" TABLE
  Position = 1
Do I = 1 To SQLDA.SQLD
  If I = 1 Then

```

```

        Line = " (" 
Else
Line = " "
Line = Line Left(''''SQLDA.I.SQLNAME''' ,20)
Line = Line "POSITION("RIGHT(POSITION,5))"
Type = SQLDA.I.SQLTYPE
Null = Type//2
If Null Then Type = Type - 1
Len = SQLDA.I.SQLLEN
Prcsn = SQLDA.I.SQLLEN.SQLPRECISION
Scale = SQLDA.I.SQLLEN.SQLSCALE
Select
When (Type = CHTYPE      ,
      |Type = GRTYP      ) THEN
      Type = SQLTYPES.Type"("STRIP(LEN)""
When (Type = FLOTYPE    ) THEN
      Type = SQLTYPES.Type"("STRIP((LEN*4)-11)  "")"
When (Type = DCTYPE     ) THEN
Do
      Type = SQLTYPES.Type "EXTERNAL"
      Type = Type"("STRIP(PRCSN)" , "STRIP(SCALE)" )
      Len = (PRCSN+2)%2
End
When (Type = DATYPE     ,
      |Type = TITYPE     ,
      |Type = TSTYPE     ) THEN
      Type = SQLTYPES.Type "EXTERNAL"
Otherwise
      Type = SQLTYPES.Type
End
If   (Type = GRTYP      ,
      |Type = VGRYP      ,
      |Type = LVGRTYP    ) THEN
      Len = Len * 2
If   (Type = VCHTYPE    ,
      |Type = LVCHTYPE   ,
      |Type = VGRYP      ,
      |Type = LVGRTYP    ) THEN
      Len = Len + 2
Line = Line Type
L = Line.0 + 1; Line.0 = L

Line.L = Line
If Null = 1 Then
Do
Line = " "
Line = Line Left('' ,20)
Line = Line "  NULLIF("RIGHT(POSITION,5))=?"
L = Line.0 + 1; Line.0 = L
Line.L = Line
End
Position = Position + Len + 1
End I
L = Line.0 + 1; Line.0 = L
Line.L = " )"
Return
/*****************************************/
/* Display SQLCA                                */
/*****************************************/
SQLCA:
"ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLSTATE='SQLSTATE''"
"ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLWARN ='SQLWARN.0" ,",
      || SQLWARN.1" ,
      || SQLWARN.2" ,
      || SQLWARN.3" ,
      || SQLWARN.4" ,
      || SQLWARN.5" ,
      || SQLWARN.6" ,
      || SQLWARN.7" ,
      || SQLWARN.8" ,
      || SQLWARN.9" ,
      || SQLWARN.10" ,
"ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLERRD =SQLERRD.1" ,
      || SQLERRD.2" ,
      || SQLERRD.3" ,
      || SQLERRD.4" ,
      || SQLERRD.5" ,
      || SQLERRD.6" ,
"ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLERRP ='SQLERRP''"
"ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLERRMC ='SQLERRMC''"

```

```
"ISREDIT LINE_AFTER "zdest" = MSGLINE 'SQLCODE ="SQLCODE"'"
Exit 20
```

Ejemplo de cómo se utiliza una variable indicadora en un programa REXX

La forma en que se utilizan las variables indicadoras para las variables de host de entrada en los programas REXX es ligeramente diferente a la forma en que se utilizan las variables indicadoras en otros lenguajes. Cuando desee pasar un valor nulo a una columna de tipo "Db2", además de poner un valor negativo en una variable indicadora, también deberá poner un valor válido en la variable anfitriona correspondiente.

Por ejemplo, las siguientes sentencias establecen un valor en la columna WORKDEPT de la tabla EMP en nulo:

```
SQLSTMT="UPDATE EMP" ,
"SET WORKDEPT = ?"
HVWORKDEPT='000'
INDWORKDEPT=-1
"EXECSQL PREPARE S100 FROM :SQLSTMT"
"EXECSQL EXECUTE S100 USING :HVWORKDEPT :INDWORKDEPT"
```

En el siguiente programa, el número de teléfono del empleado Haas se selecciona en la variable HVPhone. Después de que se ejecute la instrucción SELECT, si no se encuentra ningún número de teléfono para el empleado Haas, la variable indicadora INDPhone contiene -1.

```
'SUBCOM DSNREXX'
IF RC THEN ,
  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
ADDRESS DSNREXX
'CONNECT' 'DSN'
SQLSTMT =
  "SELECT PHONENO FROM DSN8C10.EMP WHERE LASTNAME='HAAS'"
"EXECSQL DECLARE C1 CURSOR FOR S1"
"EXECSQL PREPARE S1 FROM :SQLSTMT"
Say "SQLCODE from PREPARE is "SQLCODE
"EXECSQL OPEN C1"
Say "SQLCODE from OPEN is "SQLCODE
"EXECSQL FETCH C1 INTO :HVPhone :INDPhone"
Say "SQLCODE from FETCH is "SQLCODE
If INDPhone < 0 Then ,
  Say 'Phone number for Haas is null.'
"EXECSQL CLOSE C1"
Say "SQLCODE from CLOSE is "SQLCODE
S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX')
```

Ejemplo de programas REXX para datos LOB

Db2 los programas en REXX pueden utilizar variables de host LOB y variables de referencia de archivo, pero no variables de localizador LOB.

Ejemplo de uso de variables de host LOB simples en un programa REXX

```
/* REXX exec to use a LOB in a host var */
ssid = "VA1A" ;

Address TSO "SUBCOM DSNREXX" ;
if rc then s_rc = RXSUBCOM("ADD","DSNREXX","DSNREXX") ;
say "rc from rxsubcom add=" rc

Address DSNREXX ;

"CONNECT" ssid ;
if sqlcode \= 0 then do ;
  say "CONNECT to" ssid "failed."
  call sqlca
  exit 8 ;
end ;

stmt = "DROP TABLE REXXCLOB" ;
Address DSNREXX ,
```

```

"EXECSQL EXECUTE IMMEDIATE :STMT" ;

say "RC/SQLCODE after DROP is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

stmt = "CREATE TABLE REXXCLOB (" || ,
       "C1 CLOB(2M))" ;

Address DSNREXX ,
"EXECSQL EXECUTE IMMEDIATE :STMT" ;

say "RC/SQLCODE after CREATE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

/* Insert into the CLOB table */

data = "THIS IS A SHORT CLOB, BUT IT IS A CLOB" ;

stmt = "INSERT INTO REXXCLOB (C1) VALUES(?) " ;
Address DSNREXX "EXECSQL PREPARE S1 FROM :STMT" ;

say "RC/SQLCODE after PREPARE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

mydata = copies('Z',75000) ;
say "length of :mydata="length(mydata) ;

Address DSNREXX "EXECSQL EXECUTE S1 USING :MYDATA" ;

say "RC/SQLCODE after EXECUTE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;
/*
var1 = copies(' ',2048000) ;
say "length of :var1="length(var1) ;
*/
stmt = "SELECT C1, LENGTH(C1) FROM REXXCLOB" ;
Address DSNREXX "EXECSQL PREPARE S1 FROM :STMT" ;

say "RC/SQLCODE after PREPARE (SELECT C1) is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1" ;

say "RC/SQLCODE after DECLARE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

Address DSNREXX "EXECSQL OPEN C1" ;

say "RC/SQLCODE after OPEN is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

Address DSNREXX "EXECSQL FETCH C1 INTO :VAR1, :VAR2" ;

say "RC/SQLCODE after FETCH is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

say "length(var1)="length(var1) ;
say "var2="var2 ;

Address DSNREXX "EXECSQL CLOSE C1" ;

say "RC/SQLCODE after CLOSE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

/*********************************************
* Disconnect from the DB2 system.          *
\*****************************************/
"DISCONNECT" ;
say "RC after DISCONNECT is" rc

s_rc = RXSUBCOM("DELETE", "DSNREXX", "DSNREXX") ;
exit 0 ;

sqlca_error:
call sqlca
exit 16

/*********************************************
/* Error handling routine for bad SQL codes - just report and end. */
\*****************************************/
SQLCA:
\*****************************************/

```

```

SAY "Error. SQLCODE = >"SQLCODE"<" 
SAY "      SQLSTATE = >"SQLSTATE"<" 
SAY "      SQLERRMC = >"SQLERRMC"<" 
SAY "      SQLERRP = >"SQLERRP"<" 
SAY "      SQLERRD.1= >"SQLERRD.1"<" 
SAY "      SQLERRD.2= >"SQLERRD.2"<" 
SAY "      SQLERRD.3= >"SQLERRD.3"<" 
SAY "      SQLERRD.4= >"SQLERRD.4"<" 
SAY "      SQLERRD.5= >"SQLERRD.5"<" 
SAY "      SQLERRD.6= >"SQLERRD.6"<" 
SAY "      SQLWARN.0= >"SQLWARN.0"<" 
SAY "      SQLWARN.1= >"SQLWARN.1"<" 
SAY "      SQLWARN.2= >"SQLWARN.2"<" 
SAY "      SQLWARN.3= >"SQLWARN.3"<" 
SAY "      SQLWARN.4= >"SQLWARN.4"<" 
SAY "      SQLWARN.5= >"SQLWARN.5"<" 
SAY "      SQLWARN.6= >"SQLWARN.6"<" 
SAY "      SQLWARN.7= >"SQLWARN.7"<" 
SAY "      SQLWARN.8= >"SQLWARN.8"<" 
SAY "      SQLWARN.9= >"SQLWARN.9"<" 
SAY "      SQLWARN.10= >"SQLWARN.10"<" 
return ;

```

Ejemplo de uso de datos LOB con un SQLDA en un programa REXX

```

/* REXX EXEC TO INSERT A LOB USING SQLDA */

Address TSO "SUBCOM DSNREXX" ;
if rc then s_rc = RXSUBCOM("ADD", "DSNREXX", "DSNREXX") ;
say "rc from rxsubcom add=" rc

ssid = "VA1A" ;
Address DSNREXX "CONNECT" ssid ;
if sqlcode \= 0 then do ;
  say "CONNECT to" ssid "failed." ;
  call sqlca
  exit 8 ;
end ;

stmt = "DROP TABLE REXXCLOB" ;
Address DSNREXX "EXECSQL EXECUTE IMMEDIATE :STMT" ;

say "RC/SQLCODE after DROP is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

stmt = "CREATE TABLE REXXCLOB (" || ,
" C1 CLOB(1M))" ;

Address DSNREXX "EXECSQL EXECUTE IMMEDIATE :STMT" ;

say "RC/SQLCODE after CREATE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

/* Insert into the CLOB table */

stmt = "INSERT INTO REXXCLOB (C1) VALUES(?) " ;
Address DSNREXX "EXECSQL PREPARE S1 INTO :D1 FROM :STMT" ;

say "RC/SQLCODE after PREPARE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

mydata = copies('A',1048560) ; /* ~1M */

d1.sqld = 1 ;
d1.1.sqltype = 408 ;
d1.1.sqllongl= length(mydata) ;
d1.1.sqldata = mydata ;

say "length of mydata is" length(mydata) ;
Address DSNREXX "EXECSQL EXECUTE S1 USING DESCRIPTOR :D1" ;

say "RC/SQLCODE after EXECUTE S1, USING D1 is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

stmt = "SELECT C1, LENGTH(C1) AS LENGTH FROM REXXCLOB" ;
Address DSNREXX "EXECSQL PREPARE S1 INTO :OUTDA FROM :STMT"

say "RC/SQLCODE after PREPARE (SELECT C1) is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

```

```

say "After PREPARE INTO, SQLDA looks like:"
say "  outda.sqld=>"outda.sqld"<" ;
do i = 1 to outda.sqld ;
  say " "
  say "  outda."i".sqlname=>"outda.i.sqlname"<" ;
  say "  outda."i".sqltype=>"outda.i.sqltype"<" ;
end ;
say " "

Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1" ;

say "RC/SQLCODE after DECLARE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

Address DSNREXX "EXECSQL OPEN C1" ;

say "RC/SQLCODE after OPEN is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

Do forever ;
  Address DSNREXX "EXECSQL FETCH C1 INTO DESCRIPTOR :OUTDA" ;

  say "RC/SQLCODE after FETCH is" rc"/"sqlcode ;
  if rc <> 0 then call sqlca ;
  if sqlcode = 100 then leave ; /* do forever */
  say "outda.sqld=>"outda.sqld"<" ;

  do i = 1 to outda.sqld ;
    say i": sqlname =>"outda.i.sqlname"<" ;
    say i": sqltype =>"outda.i.sqltype"<" ;
    say i": sqlen  =>"outda.i.sqllen"<" ;
    say i": sqllongl=>"outda.i.sqllongl"<" ;
    say i": length =>"length(outda.i.sqldata)"<" ;
    if length(outda.i.sqldata) > 62 then
      say i": sqldata =>"substr(outda.i.sqldata,1,62)"<... " ;
    else
      say i": sqldata =>"outda.i.sqldata"<" ;
    say ' ' ;
  end ;
end ; /* do forever */

Address DSNREXX "EXECSQL CLOSE C1" ;

say "RC/SQLCODE after CLOSE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

/******
 * Disconnect from the DB2 system.
\*****/
Address DSNREXX "DISCONNECT" ;
say "RC after DISCONNECT is" rc

s_rc = RXSUBCOM("DELETE", "DSNREXX", "DSNREXX") ;
exit 0 ;

sqlca_error:
call sqlca
exit 16

/******
/* Error handling routine for bad SQL codes - just report and end. */
\*****/
SQLCA:
/******
SAY "      SQLCODE  = >"SQLCODE"<" 
SAY "      SQLSTATE = >"SQLSTATE"<" 
SAY "      SQLERRMC = >"SQLERRMC"<" 
SAY "      SQLERRP  = >"SQLERRP"<" 
SAY "      SQLERRD.1= >"SQLERRD.1"<" 
SAY "      SQLERRD.2= >"SQLERRD.2"<" 
SAY "      SQLERRD.3= >"SQLERRD.3"<" 
SAY "      SQLERRD.4= >"SQLERRD.4"<" 
SAY "      SQLERRD.5= >"SQLERRD.5"<" 
SAY "      SQLERRD.6= >"SQLERRD.6"<" 
SAY "      SQLWARN.0= >"SQLWARN.0"<" 
SAY "      SQLWARN.1= >"SQLWARN.1"<" 
SAY "      SQLWARN.2= >"SQLWARN.2"<" 
SAY "      SQLWARN.3= >"SQLWARN.3"<" 
SAY "      SQLWARN.4= >"SQLWARN.4"<" 
SAY "      SQLWARN.5= >"SQLWARN.5"<" 
SAY "      SQLWARN.6= >"SQLWARN.6"<" 

```

```

SAY "      SQLWARN.7= >"SQLWARN.7"<" 
SAY "      SQLWARN.8= >"SQLWARN.8"<" 
SAY "      SQLWARN.9= >"SQLWARN.9"<" 
SAY "      SQLWARN.10= >"SQLWARN.10"<" 
return ;

```

Ejemplo de uso de variables de referencia de archivo LOB en un programa REXX

```

/* REXX EXEC TO USE A CLOB FILE REFERENCE VARIABLE */

ssid = "VA1A" ;

Address TSO "SUBCOM DSNREXX" ;
if rc then s_rc = RXSUBCOM("ADD", "DSNREXX", "DSNREXX") ;
say "rc from rxsubcom add=" rc

Address DSNREXX ;

"CONNECT" ssid ;
if sqlcode \= 0 then do ;
  say "CONNECT to" ssid "failed." ;
  call sqlca_error
  exit 8 ;
end ;

stmt = "DROP TABLE REXXFRV" ;
Address DSNREXX ,
"EXECSQL EXECUTE IMMEDIATE :STMT" ;

say "RC/SQLCODE after DROP is" rc"/"sqlcode ;
if rc <> 0 & sqlcode <> -204 then call sqlca_error ;

stmt = "CREATE TABLE REXXFRV (" || ,
"  C1 CLOB(2M))" ;

Address DSNREXX ,
"EXECSQL EXECUTE IMMEDIATE :STMT" ;

say "RC/SQLCODE after CREATE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca_error ;

/*
   Write the CLOB to the preallocated file
*/
lines = 1500; /* enough 80 byte lines to make 2,000,000 bytes */
data.1 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 01" ;
data.2 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 02" ;
data.3 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 03" ;
data.4 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 04" ;
data.5 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 05" ;
data.6 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 06" ;
data.7 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 07" ;
data.8 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 08" ;
data.9 = "THIS IS A SHORT CLOB, BUT IT IS A CLOB 09" ;
data.10= "THIS IS A SHORT CLOB, BUT IT IS A CLOB 10" ;
data.0 = 10 ;

say 'data. stem initialized' ;

Do i = 1 to data.0 ;
  data.i = left(data.i,131) ;
end ;

say 'data. stem padded to 131' ;

Do i = 1 to lines ;
  Address MVS "EXECIO" data.0 "DISKW FRVFILE (stem data." ;
  if rc <> 0 then do ;
    say 'rc from execio='rc ;
    signal bad_write ;
  end ;
end ;

/* Close the file */

Address MVS "EXECIO 0 DISKW FRVFILE (FINIS" ;

/*
   The file now has to be freed. Otherwise, a

```

```

SQLCODE -452, reason 12 at location 210 will be
issued.
*/
Address TSO "FREE FI(FRFILE)" ;
if rc <> 0 then signal bad_free ;

stmt = "INSERT INTO REXXFRV (C1) VALUES(?) " ;
Address DSNREXX "EXECSQL PREPARE S1 FROM :STMT" ;

say "RC/SQLCODE after PREPARE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca_error ;

/*
   Build the special SQLDA used by REXX for working with
   LOBs.
*/
mysqlda.sqld = 1 ;
mysqlda.1.sqltype = 920      /* clob file ref var */
mysqlda.1.sqlind  = 0 ;      /* not null */

/*
   Note for a file reference variable, there is
   no SQLDATA value. Just use SQLDATA as part of the stem
   for .name and .fileoption, which are required for FRVs.
*/
mysqlda.1.sqldata.name = "SYSADM.FRV" ; /* file name */

/*
   There are 4 fileoptions that can be set, and you can
   specify the value via text or a number. Here are the
   allowable values:
      SQL_FILE_READ      or 2
      SQL_FILE_CREATE    or 8
      SQL_FILE_OVERWRITE or 16
      SQL_FILE_APPEND    or 32
*/
mysqlda.1.sqldata.fileoption = "SQL_FILE_READ" ;

/*
   sqllen is the length of the file name
*/
mysqlda.1.sqllen = length(mysqlda.1.sqldata.name) ;

Address DSNREXX "EXECSQL EXECUTE S1 USING DESCRIPTOR :MYSQLDA";
say "RC/SQLCODE after EXECUTE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca_error ;

stmt = "SELECT C1, LENGTH(C1) FROM REXXFRV" ;
Address DSNREXX "EXECSQL PREPARE S1 FROM :STMT" ;

say "RC/SQLCODE after PREPARE (SELECT C1) is" rc"/"sqlcode ;
if rc <> 0 then call sqlca_error ;

Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1" ;

say "RC/SQLCODE after DECLARE is" rc"/"sqlcode ;
if rc <> 0 then call sqlca_error ;

Address DSNREXX "EXECSQL OPEN C1" ;

say "RC/SQLCODE after OPEN is" rc"/"sqlcode ;
if rc <> 0 then call sqlca_error;

Address DSNREXX "EXECSQL FETCH C1 INTO :VAR1, :VAR2" ;

say "RC/SQLCODE after FETCH is" rc"/"sqlcode ;
if rc <> 0 then call sqlca ;

say "length(var1)="length(var1) ;
say "var1=" ;
say var1 ;

Address DSNREXX "EXECSQL CLOSE C1" ;

say "RC/SQLCODE after CLOSE is" rc"/"sqlcode ;

```

```

if rc <> 0 then call sqlca ;

/*****\*
 * Disconnect from the DB2 system. *
\*****/
"DISCONNECT" ;
say "RC after DISCONNECT is" rc

s_rc = RXSUBCOM("DELETE", "DSNREXX", "DSNREXX") ;
exit 0 ;

sqlca_error:
call sqlca
if sqlcode > 0 then return ;
exit 8 ;

/*****\*
/* Error handling routine for bad SQL codes - just report and end. */
\*****/
SQLCA:
/*****\*
SAY "Error. SQLCODE = >"SQLCODE"<" 
SAY "      SQLSTATE = >"SQLSTATE"<" 
SAY "      SQLERRMC = >"SQLERRMC"<" 
SAY "      SQLERRP = >"SQLERRP"<" 
SAY "      SQLERRD.1= >"SQLERRD.1"<" 
SAY "      SQLERRD.2= >"SQLERRD.2"<" 
SAY "      SQLERRD.3= >"SQLERRD.3"<" 
SAY "      SQLERRD.4= >"SQLERRD.4"<" 
SAY "      SQLERRD.5= >"SQLERRD.5"<" 
SAY "      SQLERRD.6= >"SQLERRD.6"<" 
SAY "      SQLWARN.0= >"SQLWARN.0"<" 
SAY "      SQLWARN.1= >"SQLWARN.1"<" 
SAY "      SQLWARN.2= >"SQLWARN.2"<" 
SAY "      SQLWARN.3= >"SQLWARN.3"<" 
SAY "      SQLWARN.4= >"SQLWARN.4"<" 
SAY "      SQLWARN.5= >"SQLWARN.5"<" 
SAY "      SQLWARN.6= >"SQLWARN.6"<" 
SAY "      SQLWARN.7= >"SQLWARN.7"<" 
SAY "      SQLWARN.8= >"SQLWARN.8"<" 
SAY "      SQLWARN.9= >"SQLWARN.9"<" 
SAY "      SQLWARN.10= >"SQLWARN.10"<" 
return ;

```

Definición del área de comunicaciones de SQL, SQLSTATE y SQLCODE en REXX

Cuando Db2 prepara un programa REXX que contiene sentencias SQL, Db2 incluye automáticamente un SQLCA en el programa.

Acerca de esta tarea

El REXX SQLCA difiere del SQLCA para otros lenguajes. El REXX SQLCA consiste en un conjunto de variables separadas, en lugar de una estructura.

El SQLCA tiene los siguientes formularios:

- Un conjunto de variables simples
- Un conjunto de variables compuestas que comienzan con la raíz SQLCA

Las variables simples son la forma predeterminada de SQLCA. El uso de CALL SQLEXEC da como resultado las variables de raíz compuesta. De lo contrario, el comando de adjunto utilizado determina la forma del SQLCA. Si utiliza la sintaxis ADDRESS DSNREXX 'CONNECT' ssid para conectarse a Db2, las variables SQLCA son un conjunto de variables simples. Si utiliza la sintaxis CALL SQLDBS 'ATTACH TO' para conectarse a Db2, las variables SQLCA son variables compuestas que comienzan con la raíz SQLCA.

No se recomienda cambiar las formas de la SQLCA dentro de una aplicación.

Tareas relacionadas

[Comprobación de la ejecución de sentencias SQL](#)

Después de ejecutar una sentencia SQL, el programa debe comprobar si hay errores antes de confirmar los datos y manejar los errores que representan.

Comprobación de la ejecución de sentencias SQL utilizando SQLCA

Un modo de comprobar si una sentencia SQL se ha ejecutado correctamente es utilizar el área de comunicación SQL (SQLCA). Esta área se distingue de la comunicación con Db2.

Comprobación de la ejecución de sentencias SQL utilizando SQLCODE y SQLSTATE

Siempre que se ejecuta una sentencia SQL, los campos SQLCODE y SQLSTATE de SQLCA reciben un código de retorno.

Definición de elementos que su programa puede utilizar para comprobar si una sentencia SQL se ha ejecutado correctamente

Si su programa contiene sentencias SQL, el programa debe definir determinada infraestructura para poder comprobar si las sentencias se han ejecutado correctamente. También puede incluir un área de comunicación SQL (SQLCA), que contenga variables SQLCODE y SQLSTATE, o declarar variables host SQLCODE y SQLSTATE.

Definición de áreas de descriptor de SQL (SQLDA) en REXX

Si su programa incluye determinadas sentencias SQL, debe definir al menos un área de descriptor SQL (SQLDA). Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Procedimiento

Codifique las declaraciones SQLDA directamente en su programa.

Cada SQLDA consta de un conjunto de variables REXX con una raíz común. El vástago debe ser un nombre de variable REXX que no contenga puntos y sea igual al valor *del nombre-descriptor* que especifique cuando utilice el SQLDA en una instrucción SQL. Para obtener más información, consulte [El REXX SQLDA \(Db2 SQL\)](#).

Restricciones:

- Debe colocar las declaraciones SQLDA antes de la primera instrucción SQL que haga referencia al descriptor de datos, a menos que utilice la opción de procesamiento TWOPASS SQL.
- No puede utilizar la instrucción SQL INCLUDE para SQLDA, porque no es compatible con COBOL.

Tareas relacionadas

Definición de áreas de descriptor de SQL (SQLDA)

Si su programa incluye ciertas sentencias SQL, debe definir al menos *un área de descriptor SQL (SQLDA)*. Dependiendo del contexto en el que se utilice, SQLDA almacena información sobre las sentencias SQL o variables de entorno preparadas. Esta información puede ser leída por el programa de aplicación o por Db2.

Referencia relacionada

[El REXX SQLDA \(Db2 SQL\)](#)

[Área de descriptor de SQL \(SQLDA\) \(Db2 SQL\)](#)

SQL equivalente y tipos de datos REXX

Todos los datos REXX son datos de cadena. Por lo tanto, cuando un programa REXX asigna datos de entrada a una columna, Db2 convierte los datos de un tipo de cadena al tipo de columna. Cuando un programa REXX asigna datos de columna a una variable de salida, Db2 convierte los datos del tipo de columna a un tipo de cadena.

Cuando asignas datos de entrada a una columna de una tabla de Db2 , puedes dejar que Db2 determine el tipo que representan tus datos de entrada, o puedes usar un SQLDA para indicar a Db2 el tipo previsto de los datos de entrada.

Cuando un programa REXX asigna datos a una columna, puede dejar que un Db2 o determine el tipo de datos o utilizar un SQLDA para especificar el tipo de datos deseado. Si el programa permite que Db2 asigne un tipo de datos para los datos de entrada, Db2 basa su elección en el formato de la cadena de entrada.

La siguiente tabla muestra los tipos de datos SQL que Db2 asigna a los datos de entrada y los formatos correspondientes para esos datos. Los dos valores SQLTYPE que se enumeran para cada tipo de datos son el valor de una columna que no acepta valores nulos y el valor de una columna que acepta valores nulos.

Tabla 117. Tipos de datos de entrada SQL y formatos de datos REXX

Tipo de datos SQL asignado por Db2	SQLTYPE para el tipo de datos	Formato de datos de entrada REXX
ENTERO	496/497	Una cadena de números que no contiene un punto decimal o un identificador de exponente. El primer carácter puede ser un signo más (+) o menos (-). El número que se representa debe estar entre -2147483648 y 2147483647, ambos inclusive.
BIGINT	492/493	Una cadena de números que no contiene un punto decimal ni un identificador de exponente. El primer carácter puede ser un signo más (+) o menos (-). El número que se representa debe estar entre -9223372036854775808 y -2147483648, ambos inclusive, o entre 2147483648 y 9223372036854775807.
DECIMAL(<i>p,s</i>)	484/485	Uno de los siguientes formatos: <ul style="list-style-type: none"> • Una cadena de números que contiene un punto decimal pero ningún identificador de exponente. <i>p</i> representa la precisión y <i>s</i> representa la escala del número decimal que representa la cadena. El primer carácter puede ser un signo más (+) o menos (-). • Una cadena de números que no contiene un punto decimal ni un identificador de exponente. El primer carácter puede ser un signo más (+) o menos (-). El número representado es menor que -9223372036854775808 o mayor que 9223372036854775807.
FLOAT	480/481	Cadena que representa un número en notación científica. La cadena consta de una serie de números seguidos de un identificador exponencial (una E o e seguida de un signo opcional más (+) o menos (-) y una serie de números). La cadena puede comenzar con un signo más (+) o menos (-).
VARCHAR (<i>n</i>)	448/449	Uno de los siguientes formatos: <ul style="list-style-type: none"> • Una cadena de longitud <i>n</i>, entre comillas simples o dobles. • El carácter X o x, seguido de una cadena entre comillas simples o dobles. La cadena entre comillas tiene una longitud de $2*n$ bytes y es la representación hexadecimal de una cadena de <i>n</i> caracteres. • Una cadena de longitud <i>n</i> que no tiene un formato numérico o gráfico y no cumple ninguna de las condiciones anteriores.

Tabla 117. Tipos de datos de entrada SQL y formatos de datos REXX (continuación)

Tipo de datos SQL asignado por Db2	SQLTYPE para el tipo de datos	Formato de datos de entrada REXX
VARGRAPHIC (n)	464/465	<p>Uno de los siguientes formatos:</p> <ul style="list-style-type: none"> • El carácter G, g, N o n, seguido de una cadena entre comillas simples o dobles. La cadena entre comillas comienza con un carácter de desplazamiento hacia afuera ('X'OE') y termina con un carácter de retorno ('X'OF'). Entre el carácter de desplazamiento hacia fuera y el carácter de desplazamiento hacia dentro hay <i>caracteres de doble byte</i>. • Los caracteres GX, Gx, gX, o gx, seguidos de una cadena entre comillas simples o dobles. La cadena entre comillas tiene una longitud de $4*n$ bytes y es la representación hexadecimal de una cadena de n caracteres de doble byte.

Por ejemplo, cuando Db2 ejecuta las siguientes sentencias para actualizar la columna MIDINIT de la tabla EMP, Db2 debe determinar un tipo de datos para HVMIDINIT:

```
SQLSTMT="UPDATE EMP" ,
"SET MIDINIT = ?" ,
"WHERE EMPNO = '000200'" 
"EXECSQL PREPARE S100 FROM :SQLSTMT"
HVMIDINIT='H'
"EXECSQL EXECUTE S100 USING" ,
":HVMIDINIT"
```

Debido a que los datos que se asignan a HVMIDINIT tienen un formato que se ajusta a un tipo de datos de carácter, Db2 REXX Language Support asigna un tipo VARCHAR a los datos de entrada.

Si no asigna un valor a una variable de host antes de asignar la variable de host a una columna, Db2 devuelve un código de error.

Conceptos relacionados

[Compatibilidad de SQL y tipos de datos de lenguaje](#)

Los tipos de datos de variable host que se utilizan en sentencias SQL deben ser compatibles con los tipos de datos de las columnas con las que tiene intención de utilizarlos.

Acceso a la aplicación de programación de interfaces de lenguaje REXX de Db2

Db2 REXX Language Support incluye varias interfaces de programación de aplicaciones que permiten que su programa REXX se conecte a un subsistema de base de datos relacional (Db2) y ejecute sentencias SQL.

Acerca de esta tarea

Db2 REXX Language Support incluye las siguientes interfaces de programación de aplicaciones:

DSNREXX CONNECT

Identifica la tarea REXX como un usuario conectado del subsistema Db2 especificado. Los recursos del plan DSNREXX se asignan estableciendo un hilo conductor.

No debe confundir el comando DSNREXX CONNECT con la instrucción SQL CONNECT de Db2 .

Debe ejecutar el comando DSNREXX CONNECT antes de que su programa REXX pueda ejecutar sentencias SQL. No utilice el comando DSNREXX CONNECT desde un procedimiento almacenado.

Una tarea REXX actualmente conectada debe desconectarse antes de cambiar a un subsistema de Db2 e diferente.

La sintaxis del comando DSNREXX CONNECT es:



Notas:

1. LLAMAR A SQLDBS 'ATTACH TO' *ssid* es una alternativa a ADDRESS DSNREXX 'CONNECT' *ssid*.
2. La *variable REXX* o la cadena de caracteres «*subsystem-ID* » también pueden ser un nombre de miembro único en un grupo de intercambio de datos o el nombre de archivo adjunto del grupo.

El siguiente ejemplo ilustra cómo establecer conexiones remotas a través de la interfaz DSNREXX.

```
/* REXX */
/* Sample to connect to remote subsystems */
/* Connect to the local subsystem */
ADDRESS DSNREXX 'CONNECT' 'DB01'
/* Now connect to multiple remote subsystems */
ADDRESS DSNREXX 'EXECSQL CONNECT TO REMOTESYS1'
.
.
.
ADDRESS DSNREXX 'EXECSQL CONNECT TO REMOTESYS2'
.
.
```

DSNREXX EXECSQL

Ejecuta instrucciones SQL en programas REXX.

La sintaxis del comando DSNREXX EXECSQL es:



Notas:

1. CALL 'SQLEXEC' "SQL-statement" es una alternativa a ADDRESS DSNREXX 'EXECSQL' "SQL-statement".
2. «EXECSQL» y «instrucción SQL» pueden escribirse entre comillas simples o dobles.

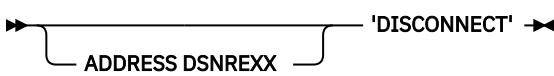
DSNREXX DESCONECTAR

Desasigna el plan DSNREXX y elimina la tarea REXX como usuario conectado de Db2.

Debe ejecutar el comando DSNREXX DISCONNECT para liberar los recursos que están retenidos por Db2. De lo contrario, los recursos no se liberan hasta que finaliza la tarea REXX.

No utilice el comando DSNREXX DISCONNECT desde un procedimiento almacenado.

La sintaxis del comando DSNREXX DISCONNECT es:



Nota: CALL SQLDBS 'DETACH' es una alternativa a ADDRESS DSNREXX 'DISCONNECT'.

Estas interfaces de programación de aplicaciones están disponibles a través del entorno de comandos del host DSNREXX. Para que DSNREXX esté disponible para la aplicación, invoque la función RXSUBCOM. La sintaxis es la siguiente:

► RXSUBCOM — (— 'ADD' — , — 'DSNREXX' — , — 'DSNREXX' —) ►

La función ADD añade DSNREXX a la tabla de entorno de comandos del host REXX. La función DELETE elimina DSNREXX de la tabla de entorno de comandos del host REXX.

El siguiente ejemplo ilustra el código REXX que pone DSNREXX a disposición de una aplicación.

Conceptos relacionados

Procedimientos almacenados de REXX

Un procedimiento almacenado REXX es similar a cualquier otro procedimiento REXX y sigue las mismas reglas que los procedimientos almacenados en otros lenguajes. Un procedimiento almacenado REXX recibe parámetros de entrada, ejecuta comandos REXX, ejecuta opcionalmente sentencias SQL y devuelve como máximo un parámetro de salida. Sin embargo, hay algunas diferencias.

Asegurarse de que Db2 interprete correctamente los datos de entrada de caracteres en los programas REXX

Db2 REXX Language Support podría interpretar incorrectamente los literales de caracteres como literales gráficos o numéricos a menos que los marque correctamente.

Procedimiento

Los caracteres literales deben ir precedidos y seguidos de comillas dobles, seguidas de comillas simples, seguidas de otras comillas dobles (" ' ").

Por ejemplo, especifique la cadena 100 como " "100" ".

Encerrar la cadena entre apóstrofes no es adecuado, porque REXX elimina los apóstrofes cuando asigna un literal a una variable. Por ejemplo, supongamos que desea pasar el valor de una variable de host llamada stringvar a Db2. El valor que desea pasar es la cadena '100'. En primer lugar, asigna la cadena a la variable host emitiendo el siguiente comando REXX :

```
stringvar = '100'
```

Después de que se ejecute el comando, stringvar contiene los caracteres 100 (sin las comillas). Db2 REXX Language Support y luego pasa el valor numérico 100 a Db2, que no es lo que pretendías.

Sin embargo, supongamos que escribe el siguiente comando:

```
stringvar = """100""
```

En este caso, REXX asigna la cadena '100' a stringvar, incluidas las comillas simples. Db2 REXX Language Support y luego pasa la cadena '100' a Db2, que es el resultado que quieres.

Pasar el tipo de datos de un tipo de datos de entrada a Db2 para programas REXX

En ciertas situaciones, debe indicar a Db2 el tipo de datos que debe utilizar para los datos de entrada en un programa REXX. Por ejemplo, si va a asignar o comparar datos de entrada con columnas de tipo SMALLINT, CHAR o GRAPHIC, debe indicarle a Db2 que utilice esos tipos de datos.

Acerca de esta tarea

Db2 no asigna tipos de datos SMALLINT, CHAR o GRAPHIC a los datos de entrada. Si asigna o compara estos datos con columnas de tipo SMALLINT, CHAR o GRAPHIC, Db2 debe realizar más trabajo que si los tipos de datos de los datos de entrada y las columnas coinciden.

Procedimiento

Utilice un SQLDA.

ejemplos

Ejemplo: Especificar CHAR como tipo de datos de entrada

Supongamos que desea indicar a Db2 que los datos con los que actualiza la columna MIDINIT de la tabla EMP son de tipo CHAR, en lugar de VARCHAR. Debe configurar un SQLDA que contenga una descripción de una columna CHAR y, a continuación, preparar y ejecutar la instrucción UPDATE utilizando ese SQLDA, como se muestra en el siguiente ejemplo.

```
INSQLDA.SQLD = 1          /* SQLDA contains one variable */
INSQLDA.1.SQLTYPE = 453   /* Type of the variable is CHAR, */
                          /* and the value can be null */
INSQLDA.1.SQLLEN = 1      /* Length of the variable is 1 */
INSQLDA.1.SQLDATA = 'H'   /* Value in variable is H */
INSQLDA.1.SQLIND = 0     /* Input variable is not null */
SQLSTMT="UPDATE EMP",
        "SET MIDINIT = ?",
        "WHERE EMPNO = '0002000'"
"EXECSQL PREPARE S100 FROM :SQLSTMT"
"EXECSQL EXECUTE S100 USING DESCRIPTOR :INSQLDA"
```

Ejemplo: especificar el tipo de datos de entrada como DECIMAL con precisión y escala

Supongamos que desea indicar a Db2 que los datos son de tipo DECIMAL con precisión y escala distinta de cero. Debe configurar un SQLDA que contenga una descripción de una columna DECIMAL, como se muestra en el siguiente ejemplo.

```
INSQLDA.SQLD = 1          /* SQLDA contains one variable */
INSQLDA.1.SQLTYPE = 484   /* Type of variable is DECIMAL */
INSQLDA.1.SQLLEN.SQLPRECISION = 18 /* Precision of variable is 18 */
INSQLDA.1.SQLLEN.SQLSCALE = 8  /* Scale of variable is 8 */
INSQLDA.1.SQLDATA = 9876543210.87654321 /* Value in variable */
```

Referencia relacionada

[Área de descriptor de SQL \(SQLDA\) \(Db2 SQL\)](#)

[El REXX SQLDA \(Db2 SQL\)](#)

Configuración del nivel de aislamiento de las sentencias SQL en un programa REXX

Los niveles de aislamiento especifican el comportamiento de bloqueo para sentencias SQL. Puede establecer el nivel de aislamiento de las sentencias SQL en su programa REXX en lectura repetible (RR), estabilidad de lectura (RS), estabilidad del cursor (CS) o lectura no confirmada (UR).

Procedimiento

Ejecute la instrucción SET CURRENT PACKAGESET para seleccionar uno de los siguientes paquetes de Db2 REXX Language Support s con el nivel de aislamiento que necesite.

Tabla 118. Db2 REXX Language Support paquetes y niveles de aislamiento asociados

Nombre del paquetena	Nivel de aislamiento
DSNREXR	Lectura repetible (RR)
DSNREXR	Estabilidad de lectura (RS)
DSNREXC	estabilidad del cursor (CS)
DSNREXUR	lectura no confirmada (UR)

Nota:

- Estos paquetes permiten que su programa acceda a Db2 y se vinculan cuando instala Db2 REXX Language Support.

Por ejemplo, para cambiar el nivel de aislamiento a estabilidad del cursor, ejecute la siguiente instrucción SQL:

```
"EXEC SQL SET CURRENT PACKAGESET='DSNREXC' "
```

Recuperación de datos de tablas de Db2 es en programas REXX

Todos los datos de salida en los programas REXX son datos de cadena. Sin embargo, puede determinar el tipo de datos que representan a partir de su formato y del tipo de datos de la columna de la que se recuperaron.

Acerca de esta tarea

La siguiente tabla muestra el formato de cada tipo de datos de salida.

Tabla 119. Tipos de datos de salida SQL y formatos de datos REXX

Tipos de datos SQL	Formato de datos de salida REXX
SMALLINT ENTERO BIGINT	Una cadena de números que no contiene ceros a la izquierda, un punto decimal o un identificador de exponente. Si la cadena representa un número negativo, comienza con un signo menos (-). El valor numérico está entre -9223372036854775808 y 9223372036854775807, ambos inclusive.
DECIMAL(p,s)	Una cadena de números con uno de los siguientes formatos: <ul style="list-style-type: none">• Contiene un punto decimal, pero no un identificador de exponente. La cadena se rellena con ceros para que coincida con la escala de la columna de la tabla correspondiente. Si el valor representa un número negativo, comienza con un signo menos (-).• No contiene un punto decimal ni un identificador de exponente. El valor numérico es menor que -9223372036854775808 o mayor que 9223372036854775807. Si el valor es negativo, comienza con un signo menos (-).

Tabla 119. Tipos de datos de salida SQL y formatos de datos REXX (continuación)

Tipos de datos SQL	Formato de datos de salida REXX
FLOTAR (n) REAL DOUBLE	Cadena que representa un número en notación científica. La cadena consta de un número, un punto decimal, una serie de números y un identificador de exponente. El identificador exponencial es una E seguida de un signo menos (-) y una serie de números si el número está entre 1 y e -1. De lo contrario, el identificador exponencial es una E seguida de una serie de números. Si la cadena representa un número negativo, comienza con un signo menos (-).
DECFLOAT	REXX emula el tipo de datos DECFLOAT con DOUBLE, por lo que la compatibilidad con DECFLOAT se limita a la compatibilidad de REXX con DOUBLE. Los siguientes valores especiales no son compatibles: <ul style="list-style-type: none"> • INFINITY • SNAN • NAN
CHAR (n) VARCHAR (n) CLOB(n) BL OB (n)	Una cadena de caracteres o valor LOB de longitud n bytes. La cadena no está entre comillas simples o dobles.
GRAPHIC (n) VARGRAPHIC (n) DBCL OB (n)	Una cadena de longitud 2*n bytes. Cada par de bytes representa un carácter de doble byte. Esta cadena no contiene una G inicial, no está entre comillas y no contiene caracteres de desplazamiento hacia fuera o hacia dentro.

Dado que no se puede utilizar la instrucción SELECT INTO en un procedimiento REXX, para recuperar datos de una tabla de tipo " Db2 " debe preparar una instrucción SELECT, abrir un cursor para la instrucción preparada y, a continuación, obtener filas en variables de host o en un SQLDA utilizando el cursor. El siguiente ejemplo muestra cómo se pueden recuperar datos de una tabla de la base de datos de SQL Server (Db2) utilizando un SQLDA:

```

SQLSTMT=
'SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME, '
' WORKDEPT, PHONENO, HIREDATE, JOB, '
' EDLEVEL, SEX, BIRTHDATE, SALARY, '
' BONUS, COMM'
' FROM EMP'
EXECSQL DECLARE C1 CURSOR FOR S1
EXECSQL PREPARE S1 INTO :OUTSQLDA FROM :SQLSTMT
EXECSQL OPEN C1
Do Until(SQLCODE >= 0)
  EXECSQL FETCH C1 USING DESCRIPTOR :OUTSQLDA
  If SQLCODE = 0 Then Do
    Line = ''
    Do I = 1 To OUTSQLDA.SQLD
      Line = Line OUTSQLDA.I.SQLDATA
    End I
    Say Line
  End
End

```

Cursos y nombres de sentencias en REXX

En las aplicaciones REXX que contienen sentencias SQL, debe utilizar un conjunto predefinido de nombres para cursos o sentencias preparadas.

Los siguientes nombres son válidos para cursos y sentencias preparadas en aplicaciones REXX :

c1 a c100

Nombres de cursor para las sentencias DECLARE CURSOR, OPEN, CLOSE y FETCH. De forma predeterminada, c1 a c100 se definen con la cláusula WITH RETURN, y c51 a c100 se definen con la cláusula WITH HOLD. Puede utilizar la cláusula ATTRIBUTES de la sentencia PREPARE para anular estos atributos o añadir atributos adicionales. Por ejemplo, es posible que desee añadir atributos para que el cursor se pueda desplazar.

c101 a c200

Nombres de cursor para las sentencias ALLOCATE, DESCRIBE, FETCH y CLOSE que se utilizan para recuperar conjuntos de resultados en un programa que llama a un procedimiento almacenado.

s1 a s100

Nombres de instrucciones preparadas para las instrucciones DECLARE STATEMENT, PREPARE, DESCRIBE y EXECUTE.

Utilice únicamente los nombres predefinidos para cursores y sentencias. Cuando asocias un nombre de cursor con un nombre de sentencia en una sentencia DECLARE CURSOR, el nombre del cursor y la sentencia deben tener el mismo número. Por ejemplo, si declara cursor c1, debe declararlo para la sentencia s1:

```
EXECSQL 'DECLARE C1 CURSOR FOR S1'
```

No utilice ninguno de los nombres predefinidos como nombres de variables de host.

Gestión de códigos de error de SQL en aplicaciones REXX

Las aplicaciones REXX pueden solicitar más información sobre los códigos de error de SQL utilizando la subrutina DSNTIAR o emitiendo una sentencia GET DIAGNOSTICS.

Procedimiento

Db2 no admite la instrucción SQL WHENEVER en un programa REXX. Para gestionar los errores y advertencias de SQL, utilice los siguientes métodos:

- Para comprobar si hay errores o advertencias de SQL, compruebe el valor SQLCODE o SQLSTATE y el SQLWARN. valores después de cada llamada EXECSQL. Este método no detecta errores en la interfaz REXX a Db2.
- Para comprobar si hay errores o advertencias de SQL o errores o advertencias de la interfaz REXX a Db2, compruebe la variable REXX RC después de cada llamada EXECSQL.

La siguiente tabla enumera los valores de la variable RC.

Tabla 120. Códigos de retorno REXX después de sentencias SQL

Código de retorno	Significado
0	No se produjo ninguna advertencia o error de SQL.
+1	Se ha producido una advertencia SQL.
-1	Se ha producido un error de SQL.
-3	El primer token después de ADDRESS DSNREXX está en error. Para una descripción de los tokens permitidos, consulte “ Acceso a la aplicación de programación de interfaces de lenguaje REXX de Db2 ” en la página 787.

También puede utilizar las instrucciones de palabra clave REXX SIGNAL ON ERROR y SIGNAL ON FAILURE para detectar valores negativos de la variable RC y transferir el control a una rutina de error.

Tareas relacionadas

[Manejo de códigos de error de SQL](#)

Los programas de aplicación pueden solicitar más información sobre los códigos de error SQL en Db2.

Referencia relacionada

[GET DIAGNOSTICS declaración \(Db2 SQL\)](#)

Capítulo 5. Invocación de un procedimiento almacenado desde una aplicación

Para ejecutar un procedimiento almacenado, puede llamarlo desde un programa cliente o invocarlo desde el Db2 command line processor.

Antes de empezar

Antes de llamar a un procedimiento almacenado, asegúrese de que dispone de todas las autorizaciones necesarias para ejecutar el procedimiento almacenado:

- Autorización para ejecutar el procedimiento almacenado al que se hace referencia en la instrucción CALL.

Las autorizaciones que necesita dependen de si la forma de la instrucción CALL es CALL *nombre-procedimiento* o CALL :*variable-host*.

- Autorización para ejecutar cualquier activador o función definida por el usuario que invoque el procedimiento almacenado.
- Autorización para ejecutar el paquete de procedimientos almacenados y cualquier paquete bajo el paquete de procedimientos almacenados.

Por ejemplo, si el procedimiento almacenado invoca alguna función definida por el usuario, necesita autorización para ejecutar los paquetes para esas funciones definidas por el usuario.

Acerca de esta tarea

Un programa de aplicación que llama a un procedimiento almacenado puede realizar una o más de las siguientes acciones:

- Llama a más de un procedimiento almacenado.
- Llamar a un único procedimiento almacenado más de una vez en el mismo nivel o en diferentes niveles de anidación. Sin embargo, no dé por sentado que las variables de los procedimientos almacenados persisten entre llamadas.

Si un procedimiento almacenado se ejecuta como programa principal, antes de cada llamada, el Entorno de lenguaje reinicializa el almacenamiento que utiliza el procedimiento almacenado. Las variables del programa para el procedimiento almacenado no persisten entre llamadas.

Si un procedimiento almacenado se ejecuta como un subprograma, el Entorno de lenguaje no inicializa el almacenamiento entre llamadas. Las variables del programa para el procedimiento almacenado pueden persistir entre llamadas. Sin embargo, no debe suponer que sus variables de programa están disponibles desde una llamada de procedimiento almacenado a otra llamada por las siguientes razones:

- Los procedimientos almacenados de otros usuarios pueden ejecutarse en una instancia de Entorno de lenguaje entre dos ejecuciones de su procedimiento almacenado.
- Las ejecuciones consecutivas de un procedimiento almacenado podrían ejecutarse en diferentes espacios de direcciones de procedimientos almacenados.
- El z/OS operador podría actualizar el entorno de lenguaje entre dos ejecuciones de su procedimiento almacenado.
- Llamar a un procedimiento almacenado local o remoto.

Si tanto el cliente como los entornos de aplicación del servidor admiten la confirmación en dos fases, el coordinador controla las actualizaciones entre la aplicación, el servidor y los procedimientos almacenados. Si alguna de las partes no admite el compromiso en dos fases, las actualizaciones fallan.

- Mezclar sentencias CALL con otras sentencias SQL.
- Utilizar cualquiera de los servicios de adjuntos de Db2 .

Db2 ejecuta procedimientos almacenados bajo el hilo de ejecución (Db2) de la aplicación que realiza la llamada, lo que significa que los procedimientos almacenados forman parte de la unidad de trabajo de la aplicación que realiza la llamada.

JDBC y aplicaciones de ODBC: Estas instrucciones no se aplican a las aplicaciones JDBC y ODBC. En su lugar, consulte la siguiente información para saber cómo llamar a los procedimientos almacenados desde esas aplicaciones:

- Para solicitudes de información (ODBC), consulte [Llamadas de procedimiento almacenado en una aplicación Db2 ODBC \(Db2 Programming for ODBC\)](#).
- Para solicitudes de JDBC, consulte [Llamada a procedimientos almacenados en aplicaciones JDBC \(Db2 Application Programming for Java\)](#)

Procedimiento

Para llamar a un procedimiento almacenado desde su aplicación:

1. Asigne valores a los parámetros IN y INOUT.
2. Opcional: Para mejorar el rendimiento de la aplicación, inicialice la longitud de los parámetros de salida LOB a cero.
3. Si el procedimiento almacenado existe en una ubicación remota, realice las siguientes acciones:
 - a) Asigne valores a los parámetros OUT.

Cuando se llama a un procedimiento almacenado en una ubicación remota, el servidor de Db2 local no puede determinar si los parámetros son de entrada (IN) o de salida (OUT o INOUT).

Por lo tanto, debe inicializar los valores de todos los parámetros de salida antes de llamar a un procedimiento almacenado en una ubicación remota.

- b) Opcional: Emitir una declaración CONNECT explícita para conectarse al servidor remoto.

Si no emite esta declaración explícitamente, puede conectarse implícitamente al servidor utilizando un nombre de tres partes para identificar el procedimiento almacenado en el siguiente paso.

La ventaja de emitir una declaración CONNECT explícita es que su declaración CALL, que se describe en el siguiente paso, es portable a otros sistemas operativos. La ventaja de conectar de forma implícita es que no es necesario emitir esta declaración CONNECT adicional.

Requisito: Al decidir si conectarse implícita o explícitamente al servidor remoto, tenga en cuenta el requisito de los programas que ejecutan las sentencias ASSOCIATE LOCATORS o DESCRIBE PROCEDURE. Debe utilizar la misma forma del nombre del procedimiento en la sentencia CALL y en la sentencia ASSOCIATE LOCATORS o DESCRIBE PROCEDURE.

4. Invoque el procedimiento almacenado con la instrucción SQL CALL. Asegúrese de que los tipos de datos de los parámetros que transmite son compatibles.

Si el procedimiento almacenado existe en un servidor remoto y no emitió una instrucción CONNECT explícita, especifique un nombre de tres partes para identificar el procedimiento almacenado y conéctese implícitamente al servidor donde se encuentra el procedimiento almacenado.

Para los procedimientos SQL nativos, se invoca por defecto la versión activa del procedimiento almacenado. Opcionalmente, puede especificar una versión del procedimiento almacenado distinta de la versión activa.

Para permitir valores nulos para los parámetros, utilice variables indicadoras.

5. Opcional: [Recuperar el estado del procedimiento](#).
6. Procesar cualquier salida, incluidos los parámetros OUT e INOUT.
7. Si el procedimiento almacenado devuelve varios conjuntos de resultados, obtenga esos resultados.

Recomendación: Cierre los conjuntos de resultados después de recuperarlos y realice confirmaciones frecuentes para evitar la falta de almacenamiento en el sistema de gestión de bases de datos (Db2) y las condiciones de EDM POOL FULL.

8. Para aplicaciones PL/I, realice también las siguientes acciones:

a) Incluya la opción de tiempo de ejecución NOEXECOPS en su código fuente.

b) Especifique la opción de tiempo de compilación SYSTEM(MVS).

Estos pasos adicionales garantizan que las convenciones de vinculación funcionen correctamente en z/OS.

9. Para aplicaciones C, incluya la siguiente línea en su código fuente:

```
#pragma runopts(PLIST(OS))
```

Este código garantiza que las convenciones de vinculación funcionen correctamente en z/OS.

Esta opción no es aplicable a otros sistemas operativos. Si planea utilizar un procedimiento almacenado C en otras plataformas además de z/OS, utilice una de las formas de compilación condicional, como se muestra en el siguiente ejemplo, para incluir esta opción solo cuando compile en z/OS.

Formulario 1

```
#ifdef MVS  
#pragma runopts(PLIST(OS))  
#endif
```

Formulario 2

```
#ifndef WKSTN  
#pragma runopts(PLIST(OS))  
#endif
```

10. Prepare la aplicación como lo haría con cualquier otra aplicación, precompilando, compilando y editando los enlaces de la aplicación y vinculando el DBRM.

Si la aplicación llama a un procedimiento almacenado remoto, realice los siguientes pasos adicionales cuando vincule el DBRM:

- Envuelva el DBRM en un paquete en el servidor de correo local (Db2). Utilice la opción de enlace DBPROTOCOL DRDA. Si el nombre del procedimiento almacenado no se puede resolver hasta el momento de la ejecución, especifique también la opción de enlace VALIDATE(RUN). El nombre del procedimiento almacenado podría no resolverse en tiempo de ejecución si utiliza una variable para el nombre del procedimiento almacenado o si el procedimiento almacenado existe en un servidor remoto.
- Envuelva el DBRM en un paquete en el servidor de Db2 remoto. Si su programa de cliente accede a varios servidores, vincule el programa a cada servidor.
- Enlazar todos los paquetes en un plan en el servidor local Db2. Utilice la opción de enlace DBPROTOCOL DRDA.

11. Asegúrese de que el procedimiento almacenado se haya completado correctamente.

Si un procedimiento almacenado finaliza de forma anormal, Db2 realiza las siguientes acciones:

- El programa de llamada recibe un error SQL como notificación de que el procedimiento almacenado ha fallado.
- Db2 coloca la unidad de trabajo del programa de llamada en un estado de reversión obligatoria.
- Db2 detiene el procedimiento almacenado y las llamadas posteriores fallan, en cualquiera de las siguientes condiciones:
 - El número de terminaciones anómalas es igual al valor DETENER DESPUÉS DE *n* FALLOS para el procedimiento almacenado.
 - El número de terminaciones anormales es igual al valor predeterminado de MAX ABEND COUNT (NÚMERO MÁXIMO DE TERMINACIONES ANORMALES) para el subsistema.
- El procedimiento almacenado no maneja la condición de terminación anormal y Db2 refresca el entorno para que Language Environment recupere el almacenamiento que utiliza la aplicación. En la mayoría de los casos, no es necesario reiniciar el entorno.

- Un conjunto de datos se asigna en la instrucción DD CEEDUMP en el procedimiento JCL que inicia el espacio de direcciones de procedimientos almacenados. En este caso, el entorno de lenguaje escribe un pequeño volcado de diagnóstico en este conjunto de datos. Utilice la información del volcado para depurar el procedimiento almacenado.
- En un entorno de intercambio de datos, el procedimiento almacenado se coloca en estado STOPABN solo en el miembro donde se produjeron los errores. Un programa de llamada puede invocar el procedimiento almacenado desde otros miembros del grupo de intercambio de datos. El estado de todos los demás miembros es INICIADO.

ejemplos

Ejemplo 1: instrucción CALL simple

El siguiente ejemplo muestra una simple instrucción CALL que puede utilizar para invocar el procedimiento almacenado A:

```
EXEC SQL CALL A (:EMP, :PRJ, :ACT, :EMT, :EMS, :EME, :TYPE, :CODE);
```

En este ejemplo, :EMP, :PRJ, :ACT, :EMT, :EMS, :EME, :TYPE y :CODE son variables de host que ha declarado anteriormente en su programa de aplicación.

Ejemplo 2: Uso de una estructura de host para múltiples valores de parámetros

En lugar de pasar cada parámetro por separado, como se muestra en el ejemplo de una instrucción CALL simple, puede pasarlo juntos como una estructura de host. Por ejemplo, supongamos que ha definido la siguiente estructura de host en su aplicación:

```
struct {
    char EMP[7];
    char PRJ[7];
    short ACT;
    short EMT;
    char EMS[11];
    char EME[11];
} empstruc;
```

A continuación, puede emitir la siguiente instrucción CALL para invocar el procedimiento almacenado A:

```
EXEC SQL CALL A (:empstruc, :TYPE, :CODE);
```

Ejemplo 3: Llamar a un procedimiento almacenado remoto

- El siguiente ejemplo muestra cómo conectarse explícitamente a LOCA y, a continuación, emitir una instrucción CALL:

```
EXEC SQL CONNECT TO LOCA;
EXEC SQL CALL SCHEMAA.A (:EMP, :PRJ, :ACT, :EMT, :EMS, :EME,
                           :TYPE, :CODE);
```

- El siguiente ejemplo muestra cómo conectarse implícitamente a LOCA especificando el nombre de tres partes para el procedimiento almacenado A en la instrucción CALL:

```
EXEC SQL CALL LOCA.SCHEMAA.A (:EMP, :PRJ, :ACT, :EMT, :EMS,
                               :EME, :TYPE, :CODE);
```

Ejemplo 4: Pasar parámetros que pueden tener valores nulos

Los ejemplos anteriores suponen que ninguno de los parámetros de entrada puede tener valores nulos. El siguiente ejemplo muestra cómo permitir valores nulos para los parámetros pasando variables indicadoras en la lista de parámetros:

```
EXEC SQL CALL A (:EMP :IEMP, :PRJ :IPRJ, :ACT :IACT,
                  :EMT :IEMT, :EMS :IEMS, :EME :IEME,
                  :TYPE :ITYPE, :CODE :ICODE);
```

En este ejemplo, :IEMP, :IPRJ, :IACT, :IEMT, :IEMS, :IEME, :ITYPE y :ICODE son variables indicadoras de los parámetros.

Ejemplo 5: Pasar constantes de cadena y valores nulos

La siguiente instrucción CALL de ejemplo pasa constantes de cadena de caracteres y enteros, un valor nulo y varias variables de host:

```
EXEC SQL CALL A ('000130', 'IF1000', 90, 1.0, NULL, '2009-10-01',
:TYPE, :CODE);
```

Ejemplo 6: de usar una variable de host para el nombre del procedimiento almacenado

La siguiente instrucción CALL de ejemplo utiliza una variable de host para el nombre del procedimiento almacenado:

```
EXEC SQL CALL :procnm (:EMP, :PRJ, :ACT, :EMT, :EMS, :EME,
:TYPE, :CODE);
```

Supongamos que el nombre del procedimiento almacenado es A. La variable host *procnm* es una variable de caracteres de longitud 255 o menos que contiene el valor 'A'. Utilice esta técnica si no conoce de antemano el nombre del procedimiento almacenado, pero sí conoce la convención de la lista de parámetros.

Ejemplo 7: Uso de un SQLDA para pasar parámetros en una estructura única

La siguiente instrucción CALL de ejemplo muestra cómo pasar parámetros en una sola estructura, la SQLDA, en lugar de como variables de host separadas:

```
EXEC SQL CALL A USING DESCRIPTOR :sqlda;
```

sqlda es el nombre de un SQLDA.

Una ventaja de utilizar un SQLDA es que puede cambiar el esquema de codificación de los valores de los parámetros del procedimiento almacenado. Por ejemplo, si el subsistema en el que se ejecuta el procedimiento almacenado tiene un esquema de codificación EBCDIC y desea recuperar datos en ASCII CCSID 437, puede especificar los CCSID para los parámetros de salida en los campos SQLVAR del SQLDA.

Esta técnica para anular los CCSID de los parámetros es la misma que la técnica para anular los CCSID de las variables. Esta técnica implica incluir SQL dinámico para instrucciones SELECT de lista variable en su programa. Cuando utilice esta técnica, el esquema de codificación definido del parámetro debe ser diferente del esquema de codificación que especifique en el SQLDA. De otro modo, no tiene lugar ninguna conversión.

El esquema de codificación definido para el parámetro es el esquema de codificación que especifique en la instrucción CREATE PROCEDURE. Si no especifica un esquema de codificación en esta declaración, el esquema de codificación definido para el parámetro es el esquema de codificación predeterminado para el subsistema.

Ejemplo 8: instrucción CALL reutilizable

Dado que la siguiente instrucción CALL de ejemplo utiliza un nombre de variable de host para el procedimiento almacenado y un SQLDA para la lista de parámetros, puede reutilizarse para llamar a diferentes procedimientos almacenados con diferentes listas de parámetros:

```
EXEC SQL CALL :procnm USING DESCRIPTOR :sqlda;
```

Su programa de cliente debe asignar un nombre de procedimiento almacenado a la variable host *procnm* y cargar el SQLDA con la información de los parámetros antes de emitir la instrucción SQL CALL.

Conceptos relacionados

Parámetros de procedimientos almacenados

Puede pasar información entre un procedimiento almacenado y el programa de aplicación de llamada utilizando parámetros. Las aplicaciones pasan los parámetros necesarios en la sentencia CALL de SQL. Opcionalmente, la aplicación también puede incluir una variable de indicador con cada parámetro para permitir valores nulos o pasar valores gramdes de parámetro de salida.

Tareas relacionadas

[Inclusión de SQL dinámico para sentencias SELECT de lista variable en el programa](#)

Una sentencia SELECT de lista variable devuelve filas que contienen un número desconocido de valores de tipo desconocido. Cuando utiliza este tipo de sentencia, no sabe de antemano exactamente qué clase de variables host debe declarar para almacenar los resultados.

[Preparación de una aplicación para ejecutarse en Db2 for z/OS](#)

Para preparar y ejecutar aplicaciones que contengan sentencias SQL estáticas incrustadas o sentencias SQL dinámicas, debe procesar, compilar, editar vínculos y vincular las sentencias SQL.

[Gestión de autorizaciones para procedimientos almacenados \(Managing Security\)](#)

[Omisión temporal de la versión activa de un procedimiento de SQL nativo](#)

Para que una llamada concreta a un procedimiento de SQL nativo utilice una versión distinta de la versión activa, se puede omitir temporalmente la versión activa. Esta omisión podría resultar útil al probar una versión nueva de un procedimiento de SQL nativo.

Referencia relacionada

[Sentencias \(Db2 SQL \)](#)

[Procedimientos que se suministran con Db2 \(Db2 SQL\)](#)

Pase de parámetros de salida de gran tamaño a procedimientos almacenados mediante variables de indicador

Si algún parámetro de salida ocupa un gran volumen de almacenamiento, el pase del área de almacenamiento completa a un procedimiento almacenado puede disminuir el rendimiento.

Acerca de esta tarea

En el programa de llamada, puede especificar variables indicadoras para que los parámetros de salida grandes pasen solo un área de 2 bytes al procedimiento almacenado, pero reciba el área de datos de salida completa del procedimiento almacenado. Cuando Db2 procesa la instrucción CALL, inspecciona los parámetros antes de mover cualquier dato. Si un parámetro de salida tiene un indicador NULL, el sistema de gestión de memoria (Db2) determina que no es necesario copiar el área de datos asociada al espacio de direcciones de la memoria virtual (Db2), lo que evita la necesidad de adquirir búferes adicionales o realizar movimientos de memoria cruzada.

Puede utilizar el siguiente procedimiento independientemente de si la convención de vinculación para el procedimiento almacenado es GENERAL, GENERAL WITH NULLS o SQL.

Procedimiento

Para pasar parámetros de salida grandes a procedimientos almacenados mediante variables de indicador:

1. Declare una variable indicadora para cada parámetro de salida grande en el procedimiento almacenado.

Si utiliza la convención GENERAL WITH NULLS o SQL linkage, debe declarar variables indicadoras para todos sus parámetros. En este caso, no es necesario declarar otra variable indicadora.

2. Asigne un valor negativo a cada variable indicadora que esté asociada con una variable de salida grande.
3. Incluya las variables indicadoras en la instrucción CALL.

Ejemplo

Por ejemplo, supongamos que un procedimiento almacenado que se define con la convención de vinculación GENERAL toma un parámetro de entrada entero y un parámetro de salida de caracteres de longitud 6000. No desea pasar el área de almacenamiento de 6000 bytes al procedimiento almacenado. El siguiente programa PL/I de ejemplo pasa solo 2 bytes al procedimiento almacenado para la variable de salida y recibe los 6000 bytes del procedimiento almacenado:

```

DCL INTVAR BIN FIXED(31);      /* This is the input variable */
DCL BIGVAR(6000);              /* This is the output variable */
DCL I1 BIN FIXED(15);          /* This is an indicator variable */
:
I1 = -1;                      /* Setting I1 to -1 causes only */
                               /* a two byte area representing */
                               /* I1 to be passed to the */
                               /* stored procedure, instead of */
                               /* the 6000 byte area for BIGVAR*/
EXEC SQL CALL PROCX(:INTVAR, :BIGVAR INDICATOR :I1);

```

Referencia relacionada

[Convenciones de enlace para procedimientos de almacenamiento externo](#)

La convención de enlace para un procedimiento almacenado puede ser GENERAL, GENERAL WITH NULLS o SQL. Estas convenciones de enlace se aplican únicamente a procedimientos de almacenamiento externos.

Tipos de datos para llamar a procedimientos almacenados

Los tipos de datos que están disponibles para las aplicaciones de llamada son los mismos que los tipos de datos que se utilizan al recuperar o actualizar procedimientos almacenados.

El formato de los parámetros que se pasan en la instrucción CALL en una aplicación debe ser compatible con los tipos de datos de los parámetros en la instrucción CREATE PROCEDURE.

Para idiomas distintos del REXX

Para todos los tipos de datos excepto LOB, ROWID, localizadores y VARCHAR (para lenguaje C), consulte las tablas enumeradas en la siguiente tabla para los tipos de datos de host que son compatibles con los tipos de datos en la definición del procedimiento almacenado.

Tabla 121. Lista de tablas de tipos de datos compatibles

Idioma	Tabla de tipos de datos compatibles
Assembler	“Tipos de datos SQL y ensamblador equivalentes” en la página 594
C	“SQL equivalente y tipos de datos C” en la página 644
COBOL	“SQL equivalente y tipos de datos COBOL” en la página 714
PL/I	“SQL equivalente y tipos de datos PL/I” en la página 760

Llamar a un procedimiento almacenado desde un procedimiento REXX

El formato de los parámetros que se pasan en la instrucción CALL en un procedimiento REXX debe ser compatible con los tipos de datos de los parámetros en la instrucción CREATE PROCEDURE.

La siguiente tabla enumera cada tipo de datos SQL que puede especificar para los parámetros en la instrucción CREATE PROCEDURE y el formato correspondiente para un parámetro REXX que representa ese tipo de datos.

Tabla 122. Formatos de parámetros para una instrucción CALL en un procedimiento REXX

Tipos de datos SQL	Formato REXX
SMALLINT	Una cadena de números que no contiene un punto decimal o un identificador de exponente. El primer carácter puede ser un signo más o menos. Este formato también se aplica a las variables indicadoras que se pasan como parámetros.
ENTERO	
BIGINT	

Tabla 122. Formatos de parámetros para una instrucción CALL en un procedimiento REXX (continuación)

Tipos de datos SQL	Formato REXX
DECIMAL(<i>p,s</i>) NUMÉRICO (<i>p, s</i>)	Una cadena de números que tiene un punto decimal pero ningún identificador de exponente. El primer carácter puede ser un signo más o menos.
REAL FLOTAR (<i>n</i>) DOUBLE DECFLOAT	Cadena que representa un número en notación científica. La cadena consta de una serie de números seguidos de un identificador exponencial (una E o e seguida de un signo opcional más o menos y una serie de números).
CARÁCTER (<i>n</i>) VARCHAR (<i>n</i>) VARCHAR(<i>n</i>) PARA DATOS DE BIT	Una cadena de longitud <i>n</i> , entre comillas simples.
GRAPHIC (<i>n</i>) VARGRAPHIC (<i>n</i>)	El carácter G seguido de una cadena entre comillas simples. La cadena entre comillas comienza con un carácter de desplazamiento hacia fuera ('X'OE') y termina con un carácter de retorno ('X'OF'). Entre el carácter de desplazamiento hacia fuera y el carácter de desplazamiento hacia dentro hay <i>n</i> caracteres de doble byte.
BINARY VARBINARY	<p>Recomendación: Pase valores BINARIOS y VARBINARIOS utilizando el SQLDA.</p> <p>Si especifica un SQLDA al llamar al procedimiento almacenado, establezca el SQLTYPE en el SQLDA. SQLDATA es una cadena de caracteres.</p> <p>Si utiliza variables de host, el formato REXX de datos BINARY y VARBINARY es BX seguido de una cadena entre comillas simples.</p>
FECHA	Una cadena de longitud 10, entre comillas simples. El formato de la cadena depende del valor del campo DATE FORMAT que especifique al instalar Db2.
HORA	Una cadena de longitud 8, entre comillas simples. El formato de la cadena depende del valor del campo TIME FORMAT que especifique al instalar Db2.
TIMESTAMP	Una cadena de longitud de 19 a 32, entre comillas simples. La cadena tiene el formato yyyy-mm-dd-hh.mm.ss o yyyy-mm-dd-hh.mm.ss.nnnnnnnnnnnn, donde el número de dígitos de segundos fraccionarios puede variar entre 0 y 12.
TIMESTAMP WITH TIME ZONE	Una cadena de longitud 148 a 161, entre comillas simples. La cadena tiene el formato aaaamm- dd-hh.mm.ss.nnnnnnnnnnnn ±th:tm o aaaamm- dd-hh.mm.ss.nnnnnnnnnnnn ±th:tm, donde el número de dígitos de segundos fraccionarios puede oscilar entre 0 y 12.
XML	No hay equivalente.

La siguiente figura muestra cómo un procedimiento REXX llama al procedimiento almacenado en “Procedimientos almacenados en REXX” en la página 302. El procedimiento REXX realiza las siguientes acciones:

- Se conecta al subsistema de la base de datos relacional (Db2) que fue especificado por el invocador del procedimiento REXX.
- Llama al procedimiento almacenado para ejecutar un comando de ejecución remota (Db2) especificado por el invocador del procedimiento REXX.

- Recupera filas de un conjunto de resultados que contiene los mensajes de salida del comando.

```

/* REXX */
PARSE ARG SSID COMMAND
/* Get the SSID to connect to */
/* and the DB2 command to be */
/* executed */
/*****************************************/
/* Set up the host command environment for SQL calls. */
/*****************************************/
"SUBCOM DSNREXX"           /* Host cmd env available? */
IF RC THEN                 /* No--make one */ */
  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
/*/*****************************************/
/* Connect to the DB2 subsystem. */
/*/*****************************************/
ADDRESS DSNREXX "CONNECT" SSID
IF SQLCODE ~= 0 THEN CALL SQLCA
PROC = 'COMMAND'
RESULTSIZE = 32703
RESULT = LEFT(' ',RESULTSIZE,' ')
/*/*****************************************/
/* Call the stored procedure that executes the DB2 command. */
/* The input variable (COMMAND) contains the DB2 command. */
/* The output variable (RESULT) will contain the return area */
/* from the IFI COMMAND call after the stored procedure */
/* executes. */
/*/*****************************************/
ADDRESS DSNREXX "EXECSQL",
"CALL" PROC "(:COMMAND, :RÉSULT)"
IF SQLCODE < 0 THEN CALL SQLCA
SAY 'RETCODE =' RETCODE
SAY 'SQLCODE =' SQLCODE
SAY 'SQLERRMC =' SQLERRMC
SAY 'SQLERRP =' SQLERRP
SAY 'SQLERRD =' SQLERRD.1', '
  || SQLERRD.2', '
  || SQLERRD.3', '
  || SQLERRD.4', '
  || SQLERRD.5', '
  || SQLERRD.6
SAY 'SQLWARN =' SQLWARN.0', '
  || SQLWARN.1', '
  || SQLWARN.2', '
  || SQLWARN.3', '
  || SQLWARN.4', '
  || SQLWARN.5', '
  || SQLWARN.6', '
  || SQLWARN.7', '
  || SQLWARN.8', '
  || SQLWARN.9', '
  || SQLWARN.10
SAY 'SQLSTATE=' SQLSTATE
SAY C2X(RESULT) """||RESULT||"""
/*/*****************************************/
/* Display the IFI return area in hexadecimal. */
/*/*****************************************/
OFFSET = 4+1
TOTLEN = LENGTH(RESULT)
DO WHILE ( OFFSET < TOTLEN )
  LEN = C2D(SUBSTR(RESULT,OFFSET,2))
  SAY SUBSTR(RESULT,OFFSET+4,LEN-4-1)
  OFFSET = OFFSET + LEN
END
/*/*****************************************/
/* Get information about result sets returned by the */
/* stored procedure. */
/*/*****************************************/
ADDRESS DSNREXX "EXECSQL DESCRIBE PROCEDURE :PROC INTO :SQLDA"
IF SQLCODE ~= 0 THEN CALL SQLCA
DO I = 1 TO SQLDA.SQLD
  SAY "SQLDA."I".SQLNAME" ="SQLDA.I.SQLNAME";"
  SAY "SQLDA."I".SQLTYPE" ="SQLDA.I.SQLTYPE";"
  SAY "SQLDA."I".SQLLOCATOR" ="SQLDA.I.SQLLOCATOR";"
END I
/*/*****************************************/
/* Set up a cursor to retrieve the rows from the result */
/* set. */
/*/*****************************************/
ADDRESS DSNREXX "EXECSQL ASSOCIATE LOCATOR (:RESULT) WITH PROCEDURE :PROC"

```

```

IF SQLCODE ~= 0 THEN CALL SQLCA
SAY RESULT
ADDRESS DSNREXX "EXECSQL ALLOCATE C101 CURSOR FOR RESULT SET :RESULT"
IF SQLCODE ~= 0 THEN CALL SQLCA
CURSOR = 'C101'
ADDRESS DSNREXX "EXECSQL DESCRIBE CURSOR :CURSOR INTO :SQLDA"
IF SQLCODE ~= 0 THEN CALL SQLCA
/* Retrieve and display the rows from the result set, which
/* contain the command output message text.
DO UNTIL(SQLCODE ~= 0)
ADDRESS DSNREXX "EXECSQL FETCH C101 INTO :SEQNO, :TEXT"
IF SQLCODE = 0 THEN
  DO
    SAY TEXT
  END
END
IF SQLCODE ~= 0 THEN CALL SQLCA
ADDRESS DSNREXX "EXECSQL CLOSE C101"
IF SQLCODE ~= 0 THEN CALL SQLCA
ADDRESS DSNREXX "EXECSQL COMMIT"
IF SQLCODE ~= 0 THEN CALL SQLCA

/* Disconnect from the DB2 subsystem.
ADDRESS DSNREXX "DISCONNECT"
IF SQLCODE ~= 0 THEN CALL SQLCA
/* Delete the host command environment for SQL.
S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX') /* REMOVE CMD ENV */
RETURN
/* Routine to display the SQLCA
SQLCA:
TRACE 0
SAY 'SQLCODE ='SQLCODE
SAY 'SQLERRMC ='SQLERRMC
SAY 'SQLERRP ='SQLERRP
SAY 'SQLERRD ='SQLERRD.1',',
      || SQLERRD.2',',
      || SQLERRD.3',',
      || SQLERRD.4',',
      || SQLERRD.5',',
      || SQLERRD.6
SAY 'SQLWARN ='SQLWARN.0',',
      || SQLWARN.1',',
      || SQLWARN.2',',
      || SQLWARN.3',',
      || SQLWARN.4',',
      || SQLWARN.5',',
      || SQLWARN.6',',
      || SQLWARN.7',',
      || SQLWARN.8',',
      || SQLWARN.9',',
      || SQLWARN.10
SAY 'SQLSTATE='SQLSTATE
EXIT

```

Conceptos relacionados

Procedimientos almacenados de REXX

Un procedimiento almacenado REXX es similar a cualquier otro procedimiento REXX y sigue las mismas reglas que los procedimientos almacenados en otros lenguajes. Un procedimiento almacenado REXX recibe parámetros de entrada, ejecuta comandos REXX, ejecuta opcionalmente sentencias SQL y devuelve como máximo un parámetro de salida. Sin embargo, hay algunas diferencias.

Preparación de un programa cliente que invoca un procedimiento almacenado remoto

Si llama a un procedimiento de almacenamiento remoto desde una aplicación SWL incorporada, debe realizar unos pasos extra cuando prepara el programa cliente. No tiene que realizar pasos extra cuando prepara el procedimiento almacenado.

Antes de empezar

Para una aplicación ODBC o CLI, asegúrese de que los paquetes y el plan Db2 que están asociados con el controlador ODBC están vinculados a Db2. Estos paquetes y planes deben estar vinculados antes de que pueda ejecutar su aplicación.

Procedimiento

Para preparar un programa de cliente que llame a un procedimiento almacenado remoto:

1. Precompilar, compilar y editar el enlace del programa cliente en el subsistema de Db2 local.
2. Enlazar el DBRM resultante en un paquete en el subsistema de Db2 local mediante el comando BIND PACKAGE con la opción DBPROTOCOL(DRDA).
3. Enlazar el mismo DBRM, el del programa cliente, en un paquete en la ubicación remota utilizando el comando BIND PACKAGE y especificando un nombre de ubicación. Si su programa de cliente necesita acceder a varios servidores, vincule el programa a cada servidor.

Por ejemplo, supongamos que desea que un programa cliente llame a un procedimiento almacenado en la ubicación LOCA. Precompila el programa para producir DBRM A. A continuación, puede utilizar el siguiente comando para vincular DBRM A a la colección de paquetes COLLA en la ubicación LOCA:

```
BIND PACKAGE (LOCA.COLLA) MEMBER(A)
```

4. Enlazar todos los paquetes en un plan en el subsistema de Db2 local. Especifique la opción de enlace DBPROTOCOL (DRDA).
5. Enlazar cualquier procedimiento almacenado que se ejecute en Db2 ODBC en un servidor de base de datos remoto Db2 como un paquete en el sitio remoto.

No es necesario que esos procedimientos se incluyan en el plan de Db2 ODBC.

Tareas relacionadas

[Vinculación de DBRM para crear paquetes \(Db2 Programming for ODBC\)](#)

Referencia relacionada

[BIND PACKAGE subcomando \(DSN\) \(Comandos de Db2\)](#)

Cómo determina Tiffany & Co. (Db2) qué procedimiento almacenado ejecutar

Un procedimiento se identifica de forma exclusiva por su nombre y el nombre de esquema que lo califica. Puede indicar a Db2 exactamente qué procedimiento almacenado ejecutar calificándolo con su nombre de esquema cuando lo llame. De lo contrario, Db2 determina qué procedimiento almacenado ejecutar.

Sin embargo, si no se identifica el nombre del procedimiento almacenado, Db2 utiliza el siguiente método para determinar qué procedimiento almacenado ejecutar:

1. Db2 busca en la lista de nombres de esquemas de la opción de enlace PATH o del registro especial CURRENT PATH de izquierda a derecha hasta que encuentra un nombre de esquema para el que existe una definición de procedimiento almacenado con el nombre en la instrucción CALL.

Db2 utiliza nombres de esquema de la opción de enlace PATH para sentencias CALL de la siguiente forma:

```
CALL procedure-name
```

Db2 utiliza nombres de esquema del registro especial CURRENT PATH para sentencias CALL de la siguiente forma:

```
CALL host-variable
```

2. Cuando Db2 encuentra una definición de procedimiento almacenado, Db2 ejecuta ese procedimiento almacenado si se cumplen las siguientes condiciones:
 - La persona que llama está autorizada a ejecutar el procedimiento almacenado.
 - El procedimiento almacenado tiene el mismo número de parámetros que en la instrucción CALL.Si no se cumplen ambas condiciones, Db2 continúa recorriendo la lista de esquemas hasta que encuentra un procedimiento almacenado que cumpla ambas condiciones o llega al final de la lista.
3. Si Db2 no puede encontrar un procedimiento almacenado adecuado, devuelve un código de error SQL para la instrucción CALL.

Llamar a diferentes versiones de un procedimiento almacenado desde una sola aplicación

Puede llamar a diferentes versiones de un procedimiento almacenado desde el mismo programa de aplicación, aunque todas esas versiones tengan el mismo nombre de módulo de carga.

Procedimiento

Para llamar a diferentes versiones de un procedimiento almacenado desde una sola aplicación:

1. Cuando defina cada versión del procedimiento almacenado, utilice el mismo nombre de procedimiento almacenado pero diferentes nombres de esquema, diferentes valores COLLID y diferentes entornos WLM.
2. En el programa que invoca el procedimiento almacenado, especifique el nombre del procedimiento almacenado no cualificado en la instrucción CALL.
3. Utilice el parámetro de configuración de procedimiento almacenado (Vía SQL) para indicar qué versión del procedimiento almacenado debe llamar el programa cliente. Puede elegir el Vía SQL de varias maneras:
 - Si el programa cliente no es una aplicación de ODBC o JDBC, utilice uno de los siguientes métodos:
 - Utilice la forma *CALL procedimiento-nombre* de la sentencia CALL. Cuando vincula planes o paquetes para el programa que llama al procedimiento almacenado, vincule un plan o paquete para cada versión del procedimiento almacenado que deseé llamar. En la opción de enlace PATH para cada plan o paquete, especifique el nombre del esquema del procedimiento almacenado al que desea llamar.
 - Utilice la forma *de variable de host CALL* de la sentencia CALL. En el programa cliente, utilice la instrucción SET PATH para especificar el nombre de esquema del procedimiento almacenado al que desea llamar.
 - Si el programa cliente es una aplicación de ODBC o JDBC, elija uno de los siguientes métodos:
 - Utilice la instrucción SET PATH para especificar el nombre de esquema del procedimiento almacenado que desea llamar.
 - Cuando vincula los paquetes de procedimientos almacenados, especifique una colección diferente para cada paquete de procedimientos almacenados. Utilice el valor COLLID que especificó al definir el procedimiento almacenado para Db2.
4. Cuando ejecute el programa cliente, especifique el plan o paquete con el valor PATH que coincide con el nombre de esquema del procedimiento almacenado al que desea llamar.

Resultados

Por ejemplo, supongamos que desea escribir un programa, PROGY, que llame a una de las dos versiones de un procedimiento almacenado llamado PROCX. El módulo de carga para ambos procedimientos almacenados se llama SUMMOD. Cada versión de SUMMOD se encuentra en una biblioteca de carga diferente. Los procedimientos almacenados se ejecutan en diferentes entornos WLM, y el JCL de inicio para cada entorno WLM incluye una concatenación STEPLIB que especifica la biblioteca de carga correcta para el módulo de procedimiento almacenado.

En primer lugar, defina los dos procedimientos almacenados en diferentes esquemas y diferentes entornos WLM:

```
CREATE PROCEDURE TEST.PROCX(IN V1 INTEGER, OUT V2 CHAR(9))
LANGUAGE C
EXTERNAL NAME SUMMOD
WLM ENVIRONMENT TESTENV;
```

```
CREATE PROCEDURE PROD.PROCX(IN V1 INTEGER, OUT V2 CHAR(9))
LANGUAGE C
EXTERNAL NAME SUMMOD
WLM ENVIRONMENT PRODENV;
```

Cuando escriba sentencias CALL para PROCX en el programa PROGY, utilice la forma no cualificada del nombre del procedimiento almacenado:

```
CALL PROCX(V1,V2);
```

Enlazar dos planes para PROGY. En una sentencia BIND, especifique PATH(TEST). En la otra sentencia BIND, especifique PATH(Prod).

Para llamar a TEST.PROCX, ejecute PROGY con el plan que vinculó con PATH(TEST). Para llamar a PROD.PROCX, ejecute PROGY con el plan que vinculó con PATH(Prod).

Invocar varias instancias de un procedimiento almacenado

Su programa de aplicación puede emitir varias sentencias CALL al mismo procedimiento almacenado local o remoto. Suponga que su procedimiento almacenado devuelve conjuntos de resultados y la aplicación que lo llama deja esos conjuntos de resultados abiertos antes de la siguiente llamada a ese mismo procedimiento almacenado. En ese caso, cada instrucción CALL invoca una instancia única del procedimiento almacenado.

Acerca de esta tarea

Cuando invoca varias instancias de un procedimiento almacenado, cada instancia se ejecuta en serie dentro del mismo hilo de ejecución de procedimientos almacenados (Db2) y abre sus propios conjuntos de resultados. Estas llamadas múltiples invocan múltiples instancias de cualquier paquete que se invoque mientras se ejecuta el procedimiento almacenado. Estas instancias se invocan en el mismo nivel o en un nivel diferente de anidamiento bajo una conexión o hilo e Db2.

Para los procedimientos almacenados locales que emiten SQL remoto, se crean instancias de las aplicaciones en el sitio del servidor remoto. Estas instancias se crean independientemente de si existen conjuntos de resultados o si se dejan abiertos entre llamadas.

Si llama a demasiados casos de un procedimiento almacenado o si abre demasiados cursos, podrían producirse situaciones de escasez de almacenamiento (Db2) y de EDM POOL FULL. Si el procedimiento almacenado emite sentencias SQL remotas a otro servidor de Db2, estas condiciones pueden darse tanto en el cliente de Db2 como en el servidor de Db2.

Procedimiento

Para invocar varias instancias de un procedimiento almacenado:

1. Para optimizar el uso del almacenamiento y evitar la escasez de almacenamiento, asegúrese de especificar los valores adecuados para los dos parámetros del subsistema siguientes:

MAX_ST_PROC

Controla el número máximo de instancias de procedimientos almacenados que puede invocar dentro del mismo hilo.

MAX_NUM_CUR

Controla el número máximo de cursos que pueden abrirse en el mismo hilo.

Cuando se supera cualquiera de los valores de estos parámetros del subsistema mientras se ejecuta una aplicación, la instrucción CALL o la instrucción OPEN recibe SQLCODE -904.

2. En su aplicación, emita sentencias CALL al procedimiento almacenado.
3. En la aplicación de llamada para el procedimiento almacenado, cierre los conjuntos de resultados y emita confirmaciones frecuentes. Incluso las aplicaciones de solo lectura deberían realizar estas acciones.

Las aplicaciones que no cierran los conjuntos de resultados o que no emiten un número adecuado de confirmaciones podrían finalizar de forma anormal con una escasez de almacenamiento de memoria intermedia (Db2) y condiciones de EDM POOL FULL.

Referencia relacionada

[MAX OPEN CURSORS \(parámetro del subsistemaMAX_NUM_CUR\) \(Db2 Instalación y migración\)](#)

[MAX STORED PROCS \(parámetro del subsistemaMAX_ST_PROC\) \(Db2 Instalación y migración\)](#)

[CALL declaración \(Db2 SQL\)](#)

Designación de la versión activa de un procedimiento de SQL nativo

Cuando se invoca un procedimiento de SQL nativo, Db2 utiliza la versión que se designa como versión activa.

Acerca de esta tarea

Cuando se crea un procedimiento SQL nativo, esa primera versión es, por defecto, la versión activa. Si crea versiones adicionales de un procedimiento almacenado, puede designar otra versión como la versión activa.

Excepción: Si un proceso sigue utilizando una versión activa existente, la nueva versión activa no se utilizará hasta la siguiente llamada a ese procedimiento.

Procedimiento

Para designar la versión activa de un procedimiento SQL nativo, emita una instrucción ALTER PROCEDURE con los siguientes elementos:

- El nombre del procedimiento nativo de SQL cuya versión activa desea cambiar.
- La cláusula ACTIVATE VERSION con el nombre de la versión que desea que esté activa.

Cuando se confirma la instrucción ALTER, la nueva versión del procedimiento se convierte en la versión activa y se utiliza en la siguiente llamada de ese procedimiento.

Ejemplo

La siguiente instrucción ALTER PROCEDURE convierte la versión V2 del procedimiento UPDATE_BALANCE en la versión activa.

```
ALTER PROCEDURE UPDATE_BALANCE  
ACTIVATE VERSION V2;
```

Omisión temporal de la versión activa de un procedimiento de SQL nativo

Para que una llamada concreta a un procedimiento de SQL nativo utilice una versión distinta de la versión activa, se puede omitir temporalmente la versión activa. Esta omisión podría resultar útil al probar una versión nueva de un procedimiento de SQL nativo.

Acerca de esta tarea

Recomendación: Si desea que todas las llamadas a un procedimiento SQL nativo utilicen una versión concreta, no anule temporalmente la versión activa en cada llamada. En su lugar, haga que esa versión sea la versión activa. De lo contrario, el rendimiento podría ser más lento.

Procedimiento

Para anular temporalmente la versión activa de un procedimiento SQL nativo, especifique las siguientes instrucciones en su programa:

1. La instrucción SET CURRENT ROUTINE VERSION con el nombre de la versión del procedimiento que desea utilizar. Si la versión especificada no existe, se utiliza la versión activa.
2. La sentencia CALL con el nombre del procedimiento.

Ejemplo

La siguiente instrucción CALL invoca la versión V1 del procedimiento UPDATE_BALANCE, independientemente de cuál sea la versión activa actual de ese procedimiento.

```
SET CURRENT ROUTINE VERSION = V1;
SET procname = 'UPDATE_BALANCE';
CALL :procname USING DESCRIPTOR :x;
```

Especificación del número de procedimientos almacenados que pueden ejecutarse de forma simultánea

Se pueden ejecutar varios procedimientos almacenados simultáneamente, cada uno bajo su propio z/OS bloque de control de tareas (TCB). El z/OS Workload Manager (WLM) gestiona cuántos procedimientos almacenados simultáneos pueden ejecutarse en un espacio de direcciones. El número de procedimientos almacenados simultáneos en un espacio de direcciones no puede sobrepasar el valor del campo NUMTCB que se especificó en el panel de instalación DSNTIPX durante la instalación de Db2.

Procedimiento

Puede anular ese valor de las siguientes maneras:

- Edite los procedimientos JCL que inician espacios de direcciones de procedimientos almacenados y modifique el valor del parámetro NUMTCB.
- Especifique el siguiente parámetro en el campo Parámetros de inicio del panel Crear un entorno de aplicación cuando configure un entorno de aplicación WLM:

```
NUMTCB=number-of-TCBs
```

Casos especiales:

- Para los procedimientos almacenados REXX, debe establecer el parámetro NUMTCB en 1.
- Los procedimientos almacenados que invocan utilidades solo pueden invocar una utilidad a la vez en un único espacio de direcciones. En consecuencia, el valor del parámetro NUMTCB se fuerza a 1 para esos procedimientos.

Conceptos relacionados

[Paso de instalación 21 : Configurar Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario \(Db2 Instalación y migración\)](#)

[Paso de migración 23 : Configurar Db2 para ejecutar procedimientos almacenados y funciones definidas por el usuario \(opcional\) \(Instalación y migración Db2 \)](#)

Tareas relacionadas

[Cómo maximizar el número de procedimientos o funciones que se ejecutan en un espacio de direcciones \(Db2 Performance\)](#)

Recuperación del estado de procedimiento

Cuando un procedimiento SQL devuelve el control al programa de llamada, también devuelve el estado de procedimiento. El estado es un valor entero que indica el éxito del procedimiento.

Acerca de esta tarea

Db2 establece el estado en 0 o en " -1 " (error de acceso a la base de datos) dependiendo del valor de SQLCODE. De forma alternativa, un procedimiento SQL puede establecer el valor de estado entero mediante la instrucción RETURN. En este caso, Db2 establece el SQLCODE en el SQLCA en 0.

Procedimiento

Para recuperar el estado del procedimiento, realice una de las siguientes acciones en el programa de llamada:

- Emitir la declaración GET DIAGNOSTICS con el elemento DB2_RETURN_STATUS. La variable de host especificada en la instrucción GET DIAGNOSTICS se establece en uno de los siguientes valores:

0

Este valor indica que el procedimiento ha devuelto un SQLCODE mayor o igual que cero. Puede acceder al valor directamente desde el SQLCA recuperando el valor de SQLERRD(1). Para aplicaciones C, recupere SQLERRD[0].

-1

Este valor indica que el procedimiento ha devuelto un SQLCODE inferior a cero. En este caso, el valor SQLERRD(1) en el SQLCA no está establecido. Db2 -1.

n

Cualquier valor distinto de 0 o de " -1 " es el valor de retorno que se estableció explícitamente en el procedimiento con la instrucción RETURN.

Por ejemplo, el siguiente código SQL crea un procedimiento SQL llamado TESTIT, que llama a otro procedimiento SQL llamado TRYIT. El procedimiento TRYIT devuelve un valor de estado. El procedimiento TESTIT recupera ese valor con el elemento " DB2_RETURN_STATUS " de la instrucción GET DIAGNOSTICS.

```
CREATE PROCEDURE TESTIT ()
LANGUAGE SQL
A1:BEGIN
DECLARE RETVAL INTEGER DEFAULT 0;
...
CALL TRYIT;
GET DIAGNOSTICS RETVAL = DB2_RETURN_STATUS;
IF RETVAL <> 0 THEN
...
LEAVE A1;
ELSE
...
END IF;
END A1
```

- Recuperar el valor de SQLERRD(1) en SQLCA. Para aplicaciones C, recupere SQLERRD[0]. Este campo contiene el valor entero que se estableció mediante la instrucción RETURN en el procedimiento SQL. Este método no es aplicable si el estado fue establecido por Db2.

Conceptos relacionados

[Área de comunicación de SQL \(SQLCA\) \(Db2 SQL\)](#)

Referencia relacionada

[GET DIAGNOSTICS declaración \(Db2 SQL\)](#)

Escribir un programa para recibir los conjuntos de resultados de un procedimiento almacenado

Puede escribir un programa para recibir los resultados de un procedimiento almacenado para un número fijo de conjuntos de resultados, cuando conoce el contenido, o un número variable de conjuntos de resultados, cuando no conoce el contenido.

Acerca de esta tarea

Un programa para un número fijo de conjuntos de resultados es más sencillo de escribir que un programa para un número variable de conjuntos de resultados. Sin embargo, si escribe un programa para un número variable de conjuntos de resultados, no necesita realizar modificaciones en el programa si cambia el procedimiento almacenado.

Si su programa llama a un procedimiento SQL que devuelve conjuntos de resultados, debe escribir el programa para un número fijo de conjuntos de resultados.

En los siguientes pasos, no es necesario que se conecte a la ubicación remota cuando ejecute estas instrucciones:

- DESCRIBE PROCEDURE
- ASSOCIATE LOCATORS
- ALLOCATE CURSOR
- DESCRIBE CURSOR
- FETCH
- CLOSE

Procedimiento

Para escribir un programa que reciba los conjuntos de resultados de un procedimiento almacenado:

1. Declare una variable localizadora para cada conjunto de resultados que se vaya a devolver.

Si no sabe cuántos conjuntos de resultados se van a devolver, declare suficientes localizadores de conjuntos de resultados para el número máximo de conjuntos de resultados que podrían devolverse.

2. Llama al procedimiento almacenado y comprueba el código de retorno SQL.

Si el SQLCODE de la instrucción CALL es +466, el procedimiento almacenado ha devuelto conjuntos de resultados.

3. Determinar cuántos conjuntos de resultados devuelve el procedimiento almacenado.

Si ya sabe cuántos conjuntos de resultados devuelve el procedimiento almacenado, omita este paso.

Utilice la instrucción SQL DESCRIBE PROCEDURE para determinar el número de conjuntos de resultados. DESCRIBE PROCEDURE coloca información sobre los conjuntos de resultados en un SQLDA. Haga que este SQLDA sea lo suficientemente grande como para contener el número máximo de conjuntos de resultados que el procedimiento almacenado podría devolver. Cuando se complete la instrucción DESCRIBE PROCEDURE, los campos del SQLDA contendrán los siguientes valores:

- SQLD contiene el número de conjuntos de resultados devueltos por el procedimiento almacenado.
- Cada entrada SQLVAR proporciona la siguiente información sobre un conjunto de resultados:
 - El campo SQLNAME contiene el nombre del cursor SQL que utiliza el procedimiento almacenado para devolver el conjunto de resultados.

- El campo SQLIND contiene el valor -1, que indica que no hay disponible ninguna estimación del número de filas en el conjunto de resultados.
- El campo SQLDATA contiene el valor del localizador del conjunto de resultados, que es la dirección del conjunto de resultados.

4. Vincular localizadores de conjuntos de resultados a conjuntos de resultados realizando una de las siguientes acciones:

- Utilice la declaración de LOCALIZADORES ASOCIADOS. Debe incrustar esta declaración en una aplicación o procedimiento SQL. La sentencia ASSOCIATE LOCATORS asigna valores a las variables de localización del conjunto de resultados. Si especifica más localizadores que el número de conjuntos de resultados que se devuelven, Db2 ignora los localizadores adicionales.
- Si ejecutó la instrucción DESCRIBE PROCEDURE anteriormente, los valores del localizador de conjunto de resultados se encuentran en los campos SQLDATA del SQLDA. Puede copiar los valores de los campos SQLDATA a los localizadores del conjunto de resultados manualmente, o puede ejecutar la instrucción ASSOCIATE LOCATORS para que lo haga por usted.

El nombre del procedimiento almacenado que especifique en una sentencia ASSOCIATE LOCATORS o DESCRIBE PROCEDURE debe coincidir con el nombre del procedimiento almacenado en la sentencia CALL de la siguiente manera:

- Si el nombre no está calificado en la instrucción CALL, no lo califique.
- Si el nombre está calificado con un nombre de esquema en la instrucción CALL, califíquelo con el nombre de esquema.
- Si el nombre está calificado con un nombre de ubicación y un nombre de esquema en la instrucción CALL, califíquelo con un nombre de ubicación y un nombre de esquema.

5. Asignar cursosres para recuperar filas de los conjuntos de resultados.

Utilice la instrucción SQL ALLOCATE CURSOR para vincular cada conjunto de resultados con un cursor. Ejecutar una sentencia ALLOCATE CURSOR para cada conjunto de resultados. Los nombres de los cursosres pueden diferir de los nombres de los cursosres en el procedimiento almacenado.

Para utilizar la instrucción ALLOCATE CURSOR, debe incrustarla en una aplicación o procedimiento SQL.

6. Determinar el contenido de los conjuntos de resultados.

Si ya conoce el formato del conjunto de resultados, omita este paso.

Utilice la instrucción SQL DESCRIBE CURSOR para determinar el formato de un conjunto de resultados y poner esta información en un SQLDA. Para cada conjunto de resultados, necesita un SQLDA que sea lo suficientemente grande como para contener descripciones de todas las columnas del conjunto de resultados.

Puede utilizar DESCRIBE CURSOR solo para aquellos cursosres para los que ejecutó previamente ALLOCATE CURSOR.

Después de ejecutar DESCRIBE CURSOR, si el cursor para el conjunto de resultados se declara WITH HOLD, el bit de orden superior del byte 8 del campo SQLDAID en el SQLDA se establece en 1.

7. Recuperar filas de los conjuntos de resultados en variables de host mediante los cursosres que asignó con las sentencias ALLOCATE CURSOR.

Obtener filas de un conjunto de resultados es lo mismo que obtener filas de una tabla.

Si ejecutó la instrucción DESCRIBE CURSOR, realice los siguientes pasos antes de recuperar las filas:

- a. Asignar almacenamiento para variables de host y variables de indicador. Utilice el contenido de la SQLDA de la instrucción DESCRIBE CURSOR para determinar cuánto almacenamiento necesita para cada variable de host.
- b. Ponga la dirección del almacenamiento para cada variable host en el campo SQLDATA apropiado del SQLDA.
- c. Introduzca la dirección del almacenamiento para cada variable indicadora en el campo SQLIND correspondiente del SQLDA.

Ejemplo

Los siguientes ejemplos muestran el código en lenguaje C que lleva a cabo cada uno de estos pasos. La codificación para otros idiomas es similar.

El siguiente ejemplo muestra cómo recibir conjuntos de resultados cuando se sabe cuántos conjuntos de resultados se devuelven y qué hay en cada conjunto de resultados.

```
*****  
/* Declare result set locators. For this example, */  
/* assume you know that two result sets will be returned. */  
/* Also, assume that you know the format of each result set. */  
*****  
EXEC SQL BEGIN DECLARE SECTION;  
    static volatile SQL TYPE IS RESULT_SET_LOCATOR *loc1, *loc2;  
EXEC SQL END DECLARE SECTION;  
:  
*****  
/* Call stored procedure P1. */  
/* Check for SQLCODE +466, which indicates that result sets */  
/* were returned. */  
*****  
EXEC SQL CALL P1(:parm1, :parm2, ...);  
if(SQLCODE==+466)  
{  
*****  
/* Establish a link between each result set and its */  
/* locator using the ASSOCIATE LOCATORS. */  
*****  
    EXEC SQL ASSOCIATE LOCATORS (:loc1, :loc2) WITH PROCEDURE P1;  
:  
*****  
/* Associate a cursor with each result set. */  
*****  
    EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :loc1;  
    EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :loc2;  
*****  
/* Fetch the result set rows into host variables. */  
*****  
    while(SQLCODE==0)  
    {  
        EXEC SQL FETCH C1 INTO :order_no, :cust_no;  
    }  
    while(SQLCODE==0)  
    {  
        EXEC SQL FETCH C2 :order_no, :item_no, :quantity;  
    }  
}
```

El siguiente ejemplo muestra cómo recibir conjuntos de resultados cuando no se sabe cuántos conjuntos de resultados se devuelven o qué hay en cada conjunto de resultados.

```
*****  
/* Declare result set locators. For this example, */  
/* assume that no more than three result sets will be */  
/* returned, so declare three locators. Also, assume */  
/* that you do not know the format of the result sets. */  
*****  
EXEC SQL BEGIN DECLARE SECTION;  
    static volatile SQL TYPE IS RESULT_SET_LOCATOR *loc1, *loc2, *loc3;  
EXEC SQL END DECLARE SECTION;  
:  
*****  
/* Call stored procedure P2. */  
/* Check for SQLCODE +466, which indicates that result sets */  
/* were returned. */  
*****  
EXEC SQL CALL P2(:parm1, :parm2, ...);  
if(SQLCODE==+466)  
{  
*****  
/* Determine how many result sets P2 returned, using the */  
/* statement DESCRIBE PROCEDURE. :proc_da is an SQLDA */  
/* with enough storage to accommodate up to three SQLVAR */  
}
```

```

/* entries. */
/***********************************************/
EXEC SQL DESCRIBE PROCEDURE P2 INTO :proc_da;
:
/***********************************************/
/* Now that you know how many result sets were returned,      */
/* establish a link between each result set and its          */
/* locator using the ASSOCIATE LOCATORS. For this example,  */
/* we assume that three result sets are returned.           */
/***********************************************/
EXEC SQL ASSOCIATE LOCATORS (:loc1, :loc2, :loc3) WITH PROCEDURE P2;
:
/***********************************************/
/* Associate a cursor with each result set.                 */
/***********************************************/
EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :loc1;
EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :loc2;
EXEC SQL ALLOCATE C3 CURSOR FOR RESULT SET :loc3;

/***********************************************/
/* Use the statement DESCRIBE CURSOR to determine the       */
/* format of each result set.                            */
/***********************************************/
EXEC SQL DESCRIBE CURSOR C1 INTO :res_da1;
EXEC SQL DESCRIBE CURSOR C2 INTO :res_da2;
EXEC SQL DESCRIBE CURSOR C3 INTO :res_da3;
:
/***********************************************/
/* Assign values to the SQLDATA and SQLIND fields of the   */
/* SQLDAs that you used in the DESCRIBE CURSOR statements. */
/* These values are the addresses of the host variables and */
/* indicator variables into which DB2 will put result set   */
/* rows.                                                 */
/***********************************************/
:
/***********************************************/
/* Fetch the result set rows into the storage areas        */
/* that the SQLDAs point to.                            */
/***********************************************/
while(SQLCODE==0)
{
  EXEC SQL FETCH C1 USING :res_da1;
  :
}
while(SQLCODE==0)
{
  EXEC SQL FETCH C2 USING :res_da2;
  :
}
while(SQLCODE==0)
{
  EXEC SQL FETCH C3 USING :res_da3;
  :
}
}

```

El siguiente ejemplo muestra cómo puede utilizar un procedimiento SQL para recibir conjuntos de resultados. La lógica asume que no existe ningún controlador para interceptar el SQLCODE +466, como DECLARE CONTINUE HANDLER FOR SQLWARNING Este controlador hace que SQLCODE se restablezca a cero. Entonces la prueba para IF SQLCODE = 466 nunca es verdadera y las declaraciones en el cuerpo IF nunca se ejecutan.

```

DECLARE RESULT1 RESULT_SET_LOCATOR VARYING;
DECLARE RESULT2 RESULT_SET_LOCATOR VARYING;
DECLARE AT_END, VAR1, VAR2 INT DEFAULT 0;
DECLARE SQLCODE INTEGER DEFAULT 0;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET AT_END = 99;
SET TOTAL1 = 0;
SET TOTAL2 = 0;
CALL TARGETPROCEDURE();
IF SQLCODE = 466 THEN
  ASSOCIATE RESULT_SET LOCATORS(RESULT1,RESULT2)
    WITH PROCEDURE SPDG3091;
  ALLOCATE RSCUR1 CURSOR FOR RESULT1;
  ALLOCATE RSCUR2 CURSOR FOR RESULT2;
  WHILE AT_END = 0 DO
    FETCH RSCUR1 INTO VAR1;
    SET TOTAL1 = TOTAL1 + VAR1;

```

```

        SET VAR1 = 0;    /* Reset so the last value fetched is not added after AT_END
*/
        END WHILE;
        SET AT_END = 0;  /* Reset for next loop */
        WHILE AT_END = 0 DO
          FETCH RSCUR2 INTO VAR2;
          SET TOTAL2 = TOTAL2 + VAR2;
          SET VAR2 = 0;   /* Reset so the last value fetched is not added after AT_END
*/
        END WHILE;
      END IF;

```

Conceptos relacionados

[Programas de ejemplo que llaman a procedimientos almacenados](#)

Los ejemplos se pueden utilizar como modelos cuando escribe aplicaciones que llaman a procedimientos almacenados. Asimismo, *prefijo.SDSNSAMP* contiene trabajos de ejemplo de DSNTEJ6P y DSNTEJ6S y programas DSN8EP1 y DSN8EP2 de ejemplo, que puede ejecutar.

Referencia relacionada

[ALLOCATE CURSOR declaración \(Db2 SQL\)](#)

[ASSOCIATE LOCATORS declaración \(Db2 SQL\)](#)

[CALL declaración \(Db2 SQL\)](#)

[DESCRIBE CURSOR declaración \(Db2 SQL\)](#)

[DESCRIBE PROCEDURE declaración \(Db2 SQL\)](#)

[Área de descriptor de SQL \(SQLDA\) \(Db2 SQL\)](#)

Capítulo 6. Métodos de codificación de datos distribuidos

Puede acceder a los datos distribuidos con nombres de tabla de tres partes o sentencias de conexión explícitas.

Conceptos introductorios

[Datos distribuidos \(Introducción a DB2 para z/OS\)](#)

[Efectos de los datos distribuidos en la programación \(Introducción a Db2 para z/OS\)](#)

[Acceso a datos distribuidos \(Introducción a Db2 para z/OS\)](#)

Los nombres de tablas de tres partes se describen en “[Acceso a datos distribuidos utilizando nombres de tablas de tres partes](#)” en la página 817. Las sentencias de conexión explícitas se describen en “[Acceso a datos distribuidos utilizando sentencias CONNECT explícitas](#)” en la página 820.

Estos dos métodos de codificación de aplicaciones para acceso distribuido se ilustran en el siguiente ejemplo.

Spiffy Computer tiene una tabla de proyectos principal que proporciona información sobre todos los proyectos que están activos actualmente en toda la empresa. Spiffy tiene varias sucursales en diferentes lugares del mundo, cada una de ellas un lugar e Db2 e que mantiene una copia de la tabla de proyectos llamada DSN8C10.PROJ. La sucursal principal inserta ocasionalmente datos en todas las copias de la tabla. La aplicación que realiza las inserciones utiliza una tabla de nombres de ubicaciones. Por cada fila que se inserta, la aplicación ejecuta una instrucción INSERT en un archivo.PROJ de l DSN8C10.

Copiar una tabla desde una ubicación remota

Para copiar una tabla de una ubicación a otra, puede escribir su propio programa de aplicación o utilizar el producto Db2 DataPropagator.

Conceptos relacionados

[Supervisión de Db2 en entornos distribuidos \(Db2 Performance\)](#)

Tareas relacionadas

[Mejora del rendimiento para las aplicaciones que acceden a datos distribuidos \(Db2 Performance\)](#)

Acceso a datos distribuidos utilizando nombres de tablas de tres partes

Puede utilizar nombres de tabla de tres partes para acceder a datos en una ubicación remota a través del acceso DRDA.

Cuando utilice nombres de tabla de tres partes, debe crear copias del paquete que utilizó en el sitio local en todas las ubicaciones remotas posibles a las que se pueda acceder mediante las referencias de nombre de tabla de tres partes. También debe especificar explícita o genéricamente los paquetes remotos en el PKLIST del PLAN que utiliza la aplicación.

Recomendación: Utilice siempre un alias, que se resuelve en un nombre de tabla de tres partes, en lugar de especificar un nombre de tabla de tres partes específico en una instrucción SQL. El uso de un alias le permitirá mover físicamente la ubicación de la mesa según sea necesario. Al utilizar un alias, puede eliminarlo y volver a crearlo especificando la nueva ubicación remota de las tablas y, a continuación, volver a vincular los paquetes de la aplicación.

En un nombre de tabla de tres partes, la primera parte indica la ubicación. El Db2 local establece y rompe una conexión implícita con un servidor remoto según sea necesario.

Cuando se analiza un nombre de tres partes y se reenvía a una ubicación remota, cualquier configuración especial de registro se propaga automáticamente al servidor remoto. Esto permite que las sentencias SQL se procesen de la misma manera, independientemente del sitio en el que se ejecute una sentencia.

Ejemplo

El siguiente ejemplo asume que todos los sistemas involucrados implementan el compromiso en dos fases. Este ejemplo sugiere actualizar varios sistemas en un bucle y finalizar la unidad de trabajo confirmando solo cuando el bucle se haya completado. Las actualizaciones se coordinan en todo el conjunto de sistemas.

La aplicación de Spiffy utiliza un nombre de ubicación para construir un nombre de tabla de tres partes en una instrucción INSERT. A continuación, prepara el estado y lo ejecuta dinámicamente. Los valores que se van a insertar se transmiten a la ubicación remota y se sustituyen por los marcadores de parámetros en la instrucción INSERT.

La siguiente descripción general muestra cómo la aplicación utiliza alias para nombres de tres partes:

```
Read in the alias values
Do for all locations
    Read location name
    Set up statement to prepare
    Prepare statement
    a    Execute statement
End loop
Commit
```

Después de que la aplicación obtenga el siguiente alias de una tabla remota que se va a insertar, por ejemplo, REGION1PROJ (que es la tabla DSN8C10.PROJ en la ubicación SAN_JOSE), crea la siguiente cadena de caracteres:

```
INSERT INTO REGION1PROJ VALUES (?,?,?,?,?,?)
```

El alias se crea de la siguiente manera:

```
CREATE ALIAS REGION1PROJ FOR SAN_JOSE.DSN8C10.PROJ
```

La aplicación asigna la cadena de caracteres a la variable INSERTX y, a continuación, ejecuta estas instrucciones:

```
EXEC SQL
    PREPARE STMT1 FROM :INSERTX;
EXEC SQL
    EXECUTE STMT1 USING :PROJNO, :PROJNAME, :DEPTNO, :RESPEMP,
                      :PRSTAFF, :PRSTDAT, :PRENDATE, :MAJPROJ;
```

Las variables de host para la tabla de proyectos de Spiffy coinciden con la declaración de la tabla de proyectos de muestra.

Para mantener la coherencia de los datos en todas las ubicaciones, la aplicación solo confirma el trabajo cuando el bucle se ha ejecutado para todas las ubicaciones. O bien todos los emplazamientos han realizado la INSERCIÓN o, si un fallo ha impedido que algún emplazamiento la realice, todos los demás emplazamientos han revertido la INSERCIÓN. (Si se produce un fallo durante el proceso de confirmación, toda la unidad de trabajo puede quedar en duda)

Nombres de tres partes y varios servidores

Recomendación: Utilice siempre un asterisco (*) para el nombre de la ubicación en una lista de paquetes. Nunca utilice el nombre explícito de la ubicación a menos que esté seguro de que no se puede acceder a ninguna otra ubicación.

Se recomiendan los siguientes pasos:

1. Envuelva el DBRM en un paquete en el centro de distribución local (Db2).

2. Enlazar copia del paquete en el primer sitio de destino del alias.
3. Encuadernar copia del paquete en el sitio de destino.

Conceptos relacionados

[Consideraciones para vincular paquetes en una ubicación remota](#)

Cuando vincula paquetes en una ubicación remota, debe entender cómo el comportamiento de los paquetes remotos es diferente del comportamiento de los paquetes locales.

[Alias \(Db2 SQL\)](#)

[Sinónimos \(en desuso\) \(Db2 SQL\)](#)

Tareas relacionadas

[Inclusión de SQL dinámico en el programa](#)

El SQL dinámico se prepara y ejecuta mientras el programa está en ejecución.

Referencia relacionada

[Tabla de proyectos \(DSN8C10 .PROJ \) \(Introducción a Db2 for z/OS \)](#)

Acceso a mesas temporales declaradas remotas mediante el uso de nombres de mesa de tres partes

Puede acceder a una mesa temporal declarada remota utilizando un nombre de tres partes. Sin embargo, si combina sentencias CONNECT explícitas y nombres de tres partes en su aplicación, una referencia a una tabla temporal declarada remota debe ser una referencia directa.

En una sentencia CREATE GLOBAL TEMPORARY TABLE o DECLARE GLOBAL TEMPORARY TABLE, no se puede especificar un alias que se resuelva en un objeto de nombre de tres partes en una ubicación remota. Tampoco puede especificar un objeto de nombre de tres partes, incluso si la ubicación del nombre de tres partes se refiere a la ubicación donde se está creando o declarando el objeto.

Ejemplo

Puede realizar la siguiente serie de acciones, que incluye una referencia directa a una tabla temporal declarada:

```
EXEC SQL CONNECT TO CHICAGO;          /* Connect to the remote site */
EXEC SQL
  DECLARE GLOBAL TEMPORARY TABLE T1 /* Define the temporary table */
    (CHARCOL CHAR(6) NOT NULL)      /* at the remote site */
    ON COMMIT DROP TABLE;
EXEC SQL CONNECT RESET;              /* Connect back to local site */
EXEC SQL INSERT INTO CHICAGO.SESSION.T1
  (VALUES 'ABCDEF');                /* Access the temporary table*/
                                         /* at the remote site (forward reference) */
```

Sin embargo, no puede realizar la siguiente serie de acciones, que incluye una referencia hacia atrás a la tabla temporal declarada:

```
EXEC SQL
  DECLARE GLOBAL TEMPORARY TABLE T1 /* Define the temporary table */
    (CHARCOL CHAR(6) NOT NULL)      /* at the local site (ATLANTA)*/
    ON COMMIT DROP TABLE;
EXEC SQL CONNECT TO CHICAGO;          /* Connect to the remote site */
EXEC SQL INSERT INTO ATLANTA.SESSION.T1
  (VALUES 'ABCDEF');                /* Cannot access temp table */
                                         /* from the remote site (backward reference)*/
```

Ejemplo de uso de un alias

Puede realizar la siguiente serie de acciones, que incluye una referencia directa a una tabla temporal declarada mediante un alias. Primero debe declarar el alias al solicitante. El nombre que le dé al alias debe resolverse para que coincida con el nombre real.

```
CREATE APPLT1 FOR CHICAGO.SESSION.T1
```

Las sentencias CONNECT y DECLARE hacen referencia a la tabla real de temperatura declarada.

```
EXEC SQL CONNECT TO CHICAGO;
EXEC SQL DECLARE GLOBAL TEMPORARY TABLE T1
  (CHARCOL CHAR(6) NOT NULL)
  ON COMMIT DROP TABLE;
EXEC SQL CONNECT RESET;
EXEC SQL INSERT INTO APPLT1 VALUES ('ABCDEF');
```

Acceso a datos distribuidos utilizando sentencias CONNECT explícitas

Cuando utiliza sentencias CONNECT explícitas para acceder a datos distribuidos, el programa de aplicación se conecta explícitamente a cada nuevo servidor.

Acerca de esta tarea

Debe vincular los DBRM para que las sentencias SQL se ejecuten en el servidor a los paquetes que residen en ese servidor.

El siguiente ejemplo asume que todos los sistemas involucrados implementan el compromiso en dos fases. Este ejemplo sugiere actualizar varios sistemas en un bucle y finalizar la unidad de trabajo confirmando solo cuando el bucle se haya completado. Las actualizaciones se coordinan en todo el conjunto de sistemas.

En este ejemplo, la aplicación de Spiffy ejecuta CONNECT para cada servidor por turno, y el servidor ejecuta INSERT. En este caso, las tablas que se van a actualizar tienen el mismo nombre, aunque cada tabla se define en un servidor diferente. La aplicación ejecuta las sentencias en un bucle, con una iteración para cada servidor.

La aplicación se conecta a cada nuevo servidor mediante una variable de host en la instrucción CONNECT. CONNECT cambia el registro especial CURRENT SERVER para mostrar la ubicación del nuevo servidor. Los valores que se introducen en la tabla se transmiten a una ubicación como variables de host de entrada.

La siguiente descripción general muestra cómo la aplicación utiliza CONNECT explícitos:

```
Read input values
Do for all locations
  Read location name
  Connect to location
  Execute insert statement
End loop
Commit
Release all
```

Por ejemplo, la aplicación inserta un nuevo nombre de ubicación en la variable LOCATION_NAME y ejecuta las siguientes instrucciones:

```
EXEC SQL
  CONNECT TO :LOCATION_NAME;
EXEC SQL
  INSERT INTO DSN8C10.PROJ VALUES (:PROJNO, :PROJNAME, :DEPTNO, :RESPEMP,
                                    :PRSTAFF, :PRSTDATE, :PRENDATE, :MAJPROJ);
```

Para mantener la coherencia de los datos en todas las ubicaciones, la aplicación solo confirma el trabajo cuando el bucle se ha ejecutado para todas las ubicaciones. O bien todos los emplazamientos han realizado la INSERCIÓN o, si un fallo ha impedido que algún emplazamiento la realice, todos los demás emplazamientos han revertido la INSERCIÓN. (Si se produce un fallo durante el proceso de confirmación, toda la unidad de trabajo puede quedar en duda)

Las variables de host para la tabla de proyectos de Spiffy coinciden con la declaración de la tabla de proyectos de muestra. LOCATION_NAME es una variable de cadena de caracteres de longitud 16.

Referencia relacionada

[Tabla de proyectos \(DSNC10 .PROJ \) \(Introducción a Db2 for z/OS \)](#)

Especificación de un nombre de alias de ubicación para varios sitios

Puede sustituir el nombre de ubicación que utiliza una aplicación para acceder a un servidor.

Acerca de esta tarea

Db2 utiliza el valor DBALIAS en la tabla SYSIBM.LOCATIONS para anular el nombre de ubicación que una aplicación utiliza para acceder a un servidor.

Por ejemplo, supongamos que una base de datos de empleados se despliega en dos sitios y que ambos sitios se dan a conocer como EMPLEADO de nombre de ubicación. Para acceder a cada sitio, inserte una fila para cada sitio en SYSIBM.LOCATIONS con los nombres de ubicación SVL_EMPLOYEE y SJ_EMPLOYEE. Ambas filas contienen EMPLOYEE como valor de DBALIAS. Cuando una aplicación emite una instrucción CONNECT TO SVL_EMPLOYEE, Db2 busca en la tabla SYSIBM.LOCATIONS para recuperar la ubicación y los atributos de red del servidor de la base de datos. Debido a que el valor de DBALIAS no está en blanco, Db2 utiliza el alias EMPLOYEE, y no el nombre de la ubicación, para acceder a la base de datos.

Si la aplicación utiliza nombres de objetos totalmente calificados en sus sentencias SQL, Db2 envía las sentencias al servidor remoto sin modificación. Por ejemplo, supongamos que la aplicación emite la sentencia SELECT * FROM SVL_EMPLOYEE. *authid.table* con el nombre de objeto completo. Sin embargo, Db2 accede al servidor remoto utilizando el alias EMPLOYEE. El servidor remoto debe identificarse como SVL_EMPLOYEE y EMPLOYEE; de lo contrario, rechaza la instrucción SQL con un mensaje que indica que no se encuentra la base de datos. Si el servidor remoto es Db2, la ubicación SVL_EMPLOYEE podría definirse como un alias de ubicación para EMPLOYEE. Db2z/OS los servidores se definen con este alias mediante la instrucción DDF ALIAS de la utilidad de inventario de registro de cambios de DSNJU003. Db2 ejecuta localmente cualquier instrucción SQL que contenga nombres de objetos totalmente calificados si el calificador de alto nivel es el nombre de la ubicación o cualquiera de sus alias.

Referencia relacionada

[Tabla de catálogo LOCATIONS \(Db2 SQL\)](#)

[DSNJU003 \(cambiar inventario de registro\) \(Db2 Utilities\)](#)

Liberar conexiones

Cuando se conecta a ubicaciones remotas de forma explícita, también debe finalizar esas conexiones de forma explícita.

Acerca de esta tarea

Para romper las conexiones, puede utilizar la instrucción RELEASE. La instrucción RELEASE difiere de la instrucción CONNECT en los siguientes aspectos:

- Mientras que la sentencia CONNECT establece una conexión inmediata, la sentencia RELEASE **no interrumpe** inmediatamente una conexión. La instrucción RELEASE etiqueta las conexiones para su liberación en el siguiente punto de confirmación. Una conexión que ha sido etiquetada para su liberación se encuentra en *estado de liberación pendiente* y aún puede utilizarse antes del siguiente punto de confirmación.
- Mientras que la sentencia CONNECT se conecta exactamente a un sistema remoto, puede utilizar la sentencia RELEASE para especificar una sola conexión o un conjunto de conexiones para liberarlas en el siguiente punto de confirmación.

Ejemplo

Al utilizar la instrucción RELEASE, puede poner cualquiera de las siguientes conexiones en estado de liberación pendiente:

- Una conexión específica que la siguiente unidad de trabajo no utiliza:

```
EXEC SQL RELEASE SPIFFY1;
```

- La conexión SQL actual, sea cual sea su nombre de ubicación:

```
EXEC SQL RELEASE CURRENT;
```

- Todas las conexiones excepto la conexión local:

```
EXEC SQL RELEASE ALL;
```

Transmisión de datos mixtos

Los datos mixtos son datos que contienen tanto caracteres como gráficos.

Acerca de esta tarea

Si transmite datos mixtos entre su sistema local y un sistema remoto, ponga los datos en cadenas de caracteres de longitud variable en lugar de cadenas de caracteres de longitud fija.

Conversión de datos mixtos: Cuando los datos ASCII MIXED o Unicode MIXED se convierten a EBCDIC MIXED, la cadena convertida es más larga que la cadena de origen. Se produce un error si esa conversión se realiza en una variable de host de entrada de longitud fija. La solución es utilizar una variable de cadena de longitud variable con una longitud máxima suficiente para contener la expansión.

Identificación del servidor en tiempo de ejecución

Puede solicitar el nombre de la ubicación del sistema al que está conectado.

Acerca de esta tarea

El registro especial SERVIDOR ACTUAL contiene el nombre de la ubicación del sistema al que está conectado. Puede asignar ese nombre a una variable de host con una instrucción como esta:

```
EXEC SQL SET :CS = CURRENT SERVER;
```

Limitaciones de SQL en servidores diferentes

Cuando ejecutas instrucciones SQL en un servidor remoto que ejecuta otro producto de la familia de productos de Oracle (Db2), existen ciertas limitaciones. Por lo general, un programa que utiliza acceso DRDA puede utilizar sentencias y cláusulas SQL que son compatibles con un servidor remoto, incluso si no son compatibles con el servidor local.

Los siguientes ejemplos sugieren qué esperar de servidores diferentes:

- Admiten SELECT, INSERT, UPDATE, DELETE, DECLARE CURSOR y FETCH, pero los detalles varían.

Ejemplo : Db2 for Linux, UNIX, and Windows y Db2 for i admiten una forma de INSERT que permite varias filas de datos de entrada. En este caso, la cláusula VALUES va seguida de varias listas entre paréntesis. Cada lista representa los valores que se deben insertar para una fila de datos. Db2 for z/OS no admite este formulario de INSERT.

- Las declaraciones de definición de datos varían más ampliamente.

Ejemplo : Db2 for z/OS admite columnas ROWID; Db2 for Linux, UNIX, and Windows no admite columnas ROWID. Las declaraciones de definición de datos que utilizan columnas ROWID no pueden ejecutarse en todas las plataformas.

- Las declaraciones pueden tener diferentes límites.

Ejemplo : Una consulta en Db2 for z/OS puede tener 750 columnas; para otros sistemas, el máximo es mayor. Pero una consulta que utilice 750 columnas o menos podría ejecutarse en todos los sistemas.

- Algunas declaraciones no se envían al servidor, sino que son procesadas completamente por el solicitante. No puede utilizar esas sentencias en un paquete remoto aunque el servidor las admita.

- En general, si una instrucción que se va a ejecutar en un servidor remoto contiene variables de host, un solicitante de Db2 asume que son variables de host de entrada, a menos que admita la sintaxis de la instrucción y pueda determinar lo contrario. Si la suposición no es válida, el servidor rechaza la declaración.

Referencia relacionada

[Acciones permitidas en instrucciones SQL \(Db2 SQL\)](#)

Soporte para la ejecución de sentencias de SQL largas en un entorno distribuido

Una aplicación distribuida puede enviar sentencias SQL preparadas cuyo tamaño sea superior a 32 KB. Si las sentencias superan los 32 KB de tamaño, es necesario que el servidor admita estas sentencias largas.

Si una aplicación distribuida asigna una instrucción SQL a una variable DBCLOB (UTF-16) y envía la instrucción preparada a un servidor remoto, el servidor de Db2 remoto la convierte a UTF-8. Si el servidor remoto no admite UTF-8, el solicitante convierte la instrucción al sistema EBCDIC CCSID antes de enviarla al servidor remoto.

Consultas distribuidas en tablas ASCII o Unicode

Cuando realiza una consulta distribuida, el servidor determina el esquema de codificación de la tabla de resultados.

Cuando una consulta distribuida contra una tabla ASCII o Unicode llega al servidor Db2 for z/OS , el servidor indica en el mensaje de respuesta que las columnas de la tabla de resultados contienen datos ASCII o Unicode, en lugar de datos EBCDIC. El mensaje de respuesta también incluye los CCSID de los datos que se van a devolver. El CCSID de los datos de una columna es el CCSID que estaba en vigor cuando se definió la columna.

El esquema de codificación en el que Db2 devuelve datos depende de dos factores:

- El esquema de codificación del sistema solicitante.

Si el solicitante es ASCII o Unicode, los datos devueltos son ASCII o Unicode. Si el solicitante es EBCDIC, los datos devueltos son EBCDIC, aunque se almacenen en el servidor como ASCII o Unicode. Sin embargo, si la instrucción SELECT que se utiliza para recuperar los datos contiene una cláusula ORDER BY, los datos se muestran en orden ASCII o Unicode.

- Si el programa de aplicación anula el CCSID para los datos devueltos. Las formas de hacerlo son las siguientes:

- Para SQL estático

Puede vincular un plan o paquete con la opción de vinculación ENCODING para controlar los CCSID de todos los datos estáticos de ese plan o paquete. Por ejemplo, si especifica ENCODING(UNICODE) al vincular un paquete en un sistema de e Db2 for z/OS encia remoto, los datos que se devuelven en las variables del host desde el sistema remoto se codifican en el CCSID Unicode predeterminado para ese sistema.

- Para SQL estático o dinámico

Un programa de aplicación puede especificar CCSID de anulación para variables de host individuales en sentencias DECLARE VARIABLE.

Un programa de aplicación que utiliza un SQLDA puede especificar un CCSID prioritario para los datos devueltos en el SQLDA. Cuando el programa de aplicación ejecuta una instrucción FETCH, usted recibe los datos en el CCSID que se especifica en el SQLDA.

Tareas relacionadas

[Establecimiento del CCSID para variables host](#)

Todos los datos de serie Db2, distintos a los datos binarios, tienen un esquema de codificación y un ID del conjunto de caracteres codificado (CCSID) asociado a ellos. Se puede asociar un esquema de

codificación y un CCSID a variables de host individuales. A continuación, los datos de esas variables de host se asocian a ese esquema de codificación y CCSID.

Referencia relacionada

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2\)](#)

Restricciones al utilizar cursos desplazables para acceder a datos distribuidos

Las restricciones que existen para los cursos desplazables dependen de lo que admitan el solicitante y el servidor.

Si un servidor Db2 for z/OS procesa una instrucción de cursor OPEN para un cursor desplazable, y la instrucción de cursor OPEN proviene de un solicitante que no admite cursos desplazables, el servidor Db2 for z/OS devuelve un error SQL. Sin embargo, si un procedimiento almacenado en el servidor utiliza un cursor desplazable para devolver un conjunto de resultados, el solicitante de nivel inferior puede acceder a los datos a través de ese cursor. El servidor Db2 for z/OS convierte el cursor del conjunto de resultados desplazable en un cursor no desplazable. El solicitante puede recuperar los datos utilizando sentencias FETCH secuenciales.

Restricciones al utilizar cursos posicionados en filas para acceder a datos distribuidos

Las restricciones que existen para los cursos posicionados en fila dependen de lo que admitan el solicitante y el servidor.

Si un servidor Db2 for z/OS procesa una instrucción de cursor OPEN para un cursor posicionado en un conjunto de filas, y la instrucción de cursor OPEN proviene de un solicitante que no admite cursos posicionados en conjuntos de filas, el servidor Db2 for z/OS devuelve un error SQL. Sin embargo, si un procedimiento almacenado en el servidor utiliza un cursor posicionado en fila para devolver un conjunto de resultados, el solicitante de nivel inferior puede acceder a los datos a través de ese cursor utilizando sentencias FETCH posicionadas en fila.

IBM MQ con Db2

IBM MQ es un sistema de gestión de mensajes que permite a las aplicaciones comunicarse en un entorno distribuido a través de diferentes sistemas operativos y redes.

IBM MQ gestiona la comunicación de un programa a otro mediante interfaces de programación de aplicaciones (API). Puede utilizar cualquiera de las siguientes API para interactuar con el sistema de gestión de mensajes de IBM MQ :

- Interfaz de colas de mensajes (MQI)
- Clases IBM MQ para Java
- IBM MQ clases para Java Message Service (JMS)

Db2 proporciona su propia interfaz de programación de aplicaciones al WebSphere MQ sistema de gestión de mensajes a través de un conjunto de funciones externas definidas por el usuario, denominadas funciones de interfaz de programación de aplicaciones (Db2 , API). MQ. Puede utilizar estas funciones en instrucciones SQL para combinar el acceso a bases de datos e Db2 es con el manejo de mensajes e IBM MQ es. Las funciones MQ de Db2 utilizan el MQI.

Referencia relacionada

[Descripción general de la interfaz de cola de mensajes \(Message Queue Interface, MQI\)](#)

Mensajes de IBM MQ

IBM MQ utiliza mensajes para transmitir información entre aplicaciones.

Los mensajes constan de las partes siguientes:

- Los atributos del mensaje, que identifican el mensaje y sus propiedades.
- Los datos del mensaje, que son los datos de aplicación transportados en el mensaje.

Conceptos relacionados

[Funciones de Db2 MQ y procedimientos almacenados XML de Db2 MQ](#)

Se pueden utilizar funciones y procedimientos almacenados de MQ de Db2 para enviar mensajes a una cola de mensajes o recibir mensajes desde una cola de mensajes.

IBM MQ manejo de mensajes

Conceptualmente, el sistema de gestión de mensajes de correo electrónico (IBM MQ) toma un fragmento de información (el mensaje) y lo envía a su destino. MQ garantiza la entrega, aunque se produzca cualquier interrupción en la red.

En IBM MQ, un destino se denomina cola de mensajes, y una cola reside en un gestor de colas. Las aplicaciones pueden poner mensajes en colas o recibirlas de ellas.

Db2 se comunica con el WebSphere sistema de gestión de mensajes a través de un conjunto de funciones externas definidas por el usuario, denominadas funciones de gestión de mensajes (Db2 , MQ). Estas funciones utilizan el MQI.

Cuando envíe un mensaje, debe especificar los tres componentes siguientes:

datos del mensaje

Define lo que se envía de un programa a otro.

Servicio

Define de dónde va a venir o a dónde va a ir el mensaje. Los parámetros para gestionar una cola se definen en el servicio, que normalmente lo define un administrador del sistema. La complejidad de los parámetros del servicio está oculta para el programa de aplicación.

política

Define cómo se gestiona el mensaje. Las políticas controlan elementos como por ejemplo:

- Los atributos del mensaje, por ejemplo, la prioridad.
- Opciones para enviar y recibir operaciones, por ejemplo, si una operación forma parte de una unidad de trabajo.

El servicio y la política predeterminados se establecen como parte de la definición de la WebSphere MQ configuración de una instalación concreta de Db2. (Esta acción la suele realizar un administrador del sistema) Db2 proporciona el servicio predeterminado Db2.DEFAULT.SERVICE y la política predeterminada Db2.DEFAULT.POLICY.

Tareas relacionadas

[Pasos adicionales para habilitar las funciones definidas por el usuario de IBM MQ \(Db2 Installation and Migration\)](#)

Referencia relacionada

[Inicio de IBM MQ](#)

IBM MQ gestión de mensajes con el MQI

Una forma de enviar y recibir mensajes de IBM MQ desde aplicaciones Db2 es utilizar las funciones Db2 MQ que utilizan MQI.

Estas funciones basadas en MQI utilizan los servicios y políticas que se definen en dos tablas de Db2 , SYSIBM.MQSERVICE_TABLE y SYSIBM.MQPOLICY_TABLE. Estas tablas son gestionadas por el usuario y normalmente las crea y mantiene un administrador del sistema. Cada tabla contiene una fila para el servicio y la política predeterminados que proporciona Db2.

El programa de aplicación no necesita conocer los detalles de los servicios y políticas que se definen en estas tablas. La aplicación solo necesita especificar qué servicio y política utilizar para cada mensaje que envía y recibe. La aplicación especifica esta información cuando llama a una función Db2 MQ.

Conceptos relacionados

[Funciones de Db2 MQ y procedimientos almacenados XML de Db2 MQ](#)

Se pueden utilizar funciones y procedimientos almacenados de MQ de Db2 para enviar mensajes a una cola de mensajes o recibir mensajes desde una cola de mensajes.

Referencia relacionada

[Tablas de Db2 MQ](#)

Las tablas MQ de Db2 contienen definiciones de política y servicio utilizadas por la Interfaz de cola de mensajes (MQI) basadas en funciones MQ de Db2. Debe llenar las tablas MQ de Db2 antes de poder utilizar las funciones basadas en MQI.

Db2 Servicios MQI

Un servicio describe un destino al que una aplicación envía mensajes o del que una aplicación recibe mensajes. Db2 Los servicios de la interfaz de cola de mensajes (MQI) se definen en la tabla Db2 SYSIBM.MQSERVICE_TABLE.

Las funciones de MQ basadas en MQI (Db2) utilizan los servicios que se definen en la tabla Db2 SYSIBM.MQSERVICE_TABLE. Esta tabla es gestionada por el usuario y normalmente la crea y mantiene un administrador del sistema. Esta tabla contiene una fila para cada servicio definido, incluidos sus servicios personalizados y el servicio predeterminado que proporciona Db2.

El programa de aplicación no necesita conocer los detalles de los servicios definidos. Cuando un programa de aplicación llama a una función MQI-based Db2 MQ, el programa selecciona un servicio de SYSIBM.MQSERVICE_TABLE especificándolo como parámetro.

Conceptos relacionados

[Funciones de Db2 MQ y procedimientos almacenados XML de Db2 MQ](#)

Se pueden utilizar funciones y procedimientos almacenados de MQ de Db2 para enviar mensajes a una cola de mensajes o recibir mensajes desde una cola de mensajes.

[Manejo de mensajes de IBM MQ](#)

Conceptualmente, el sistema de gestión de mensajes de correo electrónico (IBM MQ) toma un fragmento de información (el mensaje) y lo envía a su destino. MQ garantiza la entrega, aunque se produzca cualquier interrupción en la red.

Referencia relacionada

[Tablas de Db2 MQ](#)

Las tablas MQ de Db2 contienen definiciones de política y servicio utilizadas por la Interfaz de cola de mensajes (MQI) basadas en funciones MQ de Db2. Debe llenar las tablas MQ de Db2 antes de poder utilizar las funciones basadas en MQI.

Db2 Políticas de MQI

Una política controla cómo se manejan los mensajes de " MQ ". Db2 Las políticas de la interfaz de cola de mensajes (MQI) se definen en la tabla de configuración de la interfaz de cola de mensajes (Db2) SYSIBM.MQPOLICY_TABLE.

Las funciones de MQ basadas en MQI (Db2) utilizan las políticas que se definen en la tabla Db2 SYSIBM.MQPOLICY_TABLE. Esta tabla es gestionada por el usuario y normalmente la crea y mantiene un administrador del sistema. Esta tabla contiene una fila para cada política definida, incluidas sus políticas personalizadas y la política predeterminada que proporciona Db2.

El programa de aplicación no necesita conocer los detalles de las políticas definidas. Cuando un programa de aplicación llama a una función MQI-based Db2 MQ, el programa selecciona una política de SYSIBM.MQPOLICY_TABLE especificándola como parámetro.

Conceptos relacionados

[Funciones de Db2 MQ y procedimientos almacenados XML de Db2 MQ](#)

Se pueden utilizar funciones y procedimientos almacenados de MQ de Db2 para enviar mensajes a una cola de mensajes o recibir mensajes desde una cola de mensajes.

[Manejo de mensajes de IBM MQ](#)

Conceptualmente, el sistema de gestión de mensajes de correo electrónico (IBM MQ) toma un fragmento de información (el mensaje) y lo envía a su destino. MQ garantiza la entrega, aunque se produzca cualquier interrupción en la red.

Referencia relacionada

Tablas de Db2 MQ

Las tablas MQ de Db2 contienen definiciones de política y servicio utilizadas por la Interfaz de cola de mensajes (MQI) basadas en funciones MQ de Db2. Debe llenar las tablas MQ de Db2 antes de poder utilizar las funciones basadas en MQI.

Funciones de MQ de Db2 y procedimientos almacenados XML y MQ de Db2

Se pueden utilizar funciones y procedimientos almacenados de MQ de Db2 para enviar mensajes a una cola de mensajes o recibir mensajes desde una cola de mensajes.

Las funciones de MQ Db2 admiten los siguientes tipos de operaciones:

- Enviar y olvidar, donde no se necesita respuesta.
- Leer o recibir, donde uno o todos los mensajes se leen sin eliminarlos de la cola, o se reciben y eliminan de la cola.
- Solicitud y respuesta, cuando una aplicación de envío necesita una respuesta a una solicitud.
- Publicar y suscribirse, donde los mensajes se asignan a servicios de publicación específicos y se envían a colas. Las aplicaciones que se suscriben al servicio de suscriptor correspondiente pueden supervisar mensajes específicos.

Se pueden utilizar funciones y procedimientos almacenados de MQ de Db2 para enviar mensajes a una cola de mensajes o recibir mensajes desde una cola de mensajes. Puede enviar una solicitud a una cola de mensajes y recibir una respuesta, y también puede publicar mensajes en el editor IBM MQ y suscribirse a mensajes que se han publicado con temas específicos. Las funciones XML y los procedimientos almacenados de Db2 MQ le permiten consultar documentos XML y, a continuación, publicar los resultados en una cola de mensajes.

Las funciones MQ de Db2 incluyen funciones escalares, funciones de tabla y funciones específicas de XML. Para cada una de estas funciones, puede llamar a una versión que utilice el MQI. Las firmas de función son las mismas. Sin embargo, los nombres de esquema que cumplen los requisitos son diferentes. Para llamar a una función basada en MQI, especifique el nombre del esquema DB2MQ.

Requisito: Antes de poder llamar a la versión de estas funciones que utiliza MQI, es necesario rellenar las tablas Db2 MQ.

La siguiente tabla describe las funciones escalares MQ de Db2 .

Tabla 123. Db2 MQ funciones escalares

Función escalar	Descripción
MQREAD (<i>recibir-servicio, servicio-política</i>)	MQREAD devuelve un mensaje en una variable VARCHAR desde la ubicación de la base de datos (MQ) especificada por <i>receive-service</i> , utilizando la política definida en <i>service-policy</i> . Esta operación no elimina el mensaje de la cabecera de la cola sino que en su lugar lo devuelve. Si no hay disponibles mensajes para devolver, se devuelve un valor nulo.
MQREADCLOB (<i>recibir-servicio, servicio-política</i>)	MQREADCLOB devuelve un mensaje en una variable CLOB de la ubicación de almacenamiento (MQ) especificada por <i>receive-service</i> , utilizando la política definida en <i>service-policy</i> . Esta operación no elimina el mensaje de la cabecera de la cola sino que en su lugar lo devuelve. Si no hay disponibles mensajes para devolver, se devuelve un valor nulo.

Tabla 123. Db2 MQ funciones escalares (continuación)

Función escalar	Descripción
MQRECEIVE (<i>recibir-servicio, servicio-política, correlación-id</i>)	MQRECEIVE devuelve un mensaje en una variable VARCHAR desde la ubicación de la base de datos de mensajes (MQ) especificada por <i>receive-service</i> , utilizando la política definida en <i>service-policy</i> . Esta operación devuelve el mensaje de la cola. Si se especifica <i>el identificador de correlación</i> , se devuelve el primer mensaje con un identificador de correlación coincidente; si no se especifica <i>el identificador de correlación</i> , se devuelve el mensaje que se encuentra al principio de la cola. Si no hay disponibles mensajes para devolver, se devuelve un valor nulo.
MQRECEIVECLOB (<i>receive-service, service-policy, correlation-id</i>)	MQRECEIVECLOB devuelve un mensaje en una variable CLOB desde la ubicación de MQ, especificada por <i>receive-service</i> , utilizando la política definida en <i>service-policy</i> . Esta operación devuelve el mensaje de la cola. Si está especificado <i>correlation-id</i> , se devuelve el primer mensaje con un identificador de correlación coincidente; si no está especificado <i>correlation-id</i> , se devuelve el mensaje en la cabecera de la cola. Si no hay disponibles mensajes para devolver, se devuelve un valor nulo.
MQSEND (<i>servicio de envío, política de servicio, datos de mensaje, id de correlación</i>)	MQSEND envía los datos en <i>una variable msg-data</i> VARCHAR o CLOB a la ubicación MQ especificada por <i>send-service</i> , utilizando la política definida en <i>service-policy</i> . <i>correlation-id</i> puede especificar un identificador de correlación de mensaje definido por el usuario opcional. El valor de retorno es 1 si es satisfactorio o 0 si no es satisfactorio.

Notas:

1. Puede enviar o recibir mensajes en variables VARCHAR o variables CLOB. La longitud máxima de un mensaje en una variable VARCHAR es de 32 KB. La longitud máxima de un mensaje en una variable CLOB es de 2 MB.

La siguiente tabla describe las funciones de la tabla de MQ que puede utilizar Db2 .

Tabla 124. Db2 MQ funciones de la tabla

Función de tabla	Descripción
MQREADALL (<i>recibir-servicio, servicio-política, num-filas</i>)	MQREADALL devuelve una tabla que contiene los mensajes y los metadatos de los mensajes en variables VARCHAR desde la ubicación de MQ es especificada por <i>receive-service</i> , utilizando la política definida en <i>service-policy</i> . Esta operación no elimina los mensajes de la cola. Si está especificado <i>num-rows</i> , se devuelve un número máximo de mensajes <i>num-rows</i> ; si no está especificado <i>num-rows</i> , se devuelven todos los mensajes disponibles.
MQREADALLCLOB (<i>receive-service, service-policy, num-rows</i>)	MQREADALLCLOB devuelve una tabla que contiene los mensajes y los metadatos de los mensajes en variables CLOB de la ubicación de MQ es especificada por <i>receive-service</i> , utilizando la política definida en <i>service-policy</i> . Esta operación no elimina los mensajes de la cola. Si está especificado <i>num-rows</i> , se devuelve un número máximo de mensajes <i>num-rows</i> ; si no está especificado <i>num-rows</i> , se devuelven todos los mensajes disponibles.

Tabla 124. Db2 MQ funciones de la tabla (continuación)

Función de tabla	Descripción
MQRECEIVEALL (<i>receive-service</i> , <i>service-policy</i> , <i>correlation-id</i> , <i>num-rows</i>)	MQRECEIVEALL devuelve una tabla que contiene los mensajes y los metadatos de los mensajes en variables VARCHAR de la ubicación de recepción (MQ) especificada por <i>receive-service</i> , utilizando la política definida en <i>service-policy</i> . Esta operación elimina los mensajes de la cola. Si está especificado <i>correlation-id</i> , sólo se devuelven los mensajes con un identificador de correlación coincidente; si no está especificado <i>correlation-id</i> , se devuelven todos los mensajes disponibles. Si está especificado <i>num-rows</i> , se devuelve un número máximo de mensajes <i>num-rows</i> ; si no está especificado <i>num-rows</i> , se devuelven todos los mensajes disponibles.
MQRECEIVEALLCLOB (<i>receive-service</i> , <i>service-policy</i> , <i>correlation-id</i> , <i>num-rows</i>)	MQRECEIVEALLCLOB devuelve una tabla que contiene los mensajes y los metadatos de los mensajes en variables CLOB de la ubicación de recepción (MQ) especificada por <i>receive-service</i> , utilizando la política definida en <i>service-policy</i> . Esta operación elimina los mensajes de la cola. Si está especificado <i>correlation-id</i> , sólo se devuelven los mensajes con un identificador de correlación coincidente; si no está especificado <i>correlation-id</i> , se devuelven todos los mensajes disponibles. Si está especificado <i>num-rows</i> , se devuelve un número máximo de mensajes <i>num-rows</i> ; si no está especificado <i>num-rows</i> , se devuelven todos los mensajes disponibles.

Notas:

1. Puede enviar o recibir mensajes en variables VARCHAR o variables CLOB. La longitud máxima de un mensaje en una variable VARCHAR es de 32 KB. La longitud máxima de un mensaje en una variable CLOB es de 2 MB.
2. La primera columna de la tabla de resultados de una función de tabla Db2 MQ contiene el mensaje.

Tareas relacionadas

Pasos adicionales para habilitar las funciones definidas por el usuario de IBM MQ (Db2 Installation and Migration)

Referencia relacionada

[Procedimientos que se suministran con Db2 \(Db2 SQL\)](#)

[MQREADALL función de tabla \(Db2 SQL \)](#)

[MQREADALLCLOB función de tabla \(Db2 SQL \)](#)

[MQRECEIVEALL función de tabla \(Db2 SQL \)](#)

[MQRECEIVEALLCLOB función de tabla \(Db2 SQL \)](#)

[Descripción general de la interfaz de cola de mensajes \(Message Queue Interface, MQI\)](#)

Generación de documentos XML a partir de tablas existentes y posterior envío a una cola de mensajes de MQ

Puede enviar datos desde una tabla Db2 a la cola de mensajes MQ. Para ello, se deben poner los datos en el documento XML y enviar dicho documento a la cola de mensajes.

Procedimiento

Para generar documentos XML a partir de tablas existentes y enviarlos a una cola de mensajes de MQ :

1. Redactar un documento XML utilizando las funciones de publicación XML de Db2 .
2. Convertir el documento XML al tipo VARCHAR o CLOB.
3. Enviar el documento a una cola de mensajes de correo electrónico (MQ) utilizando la función adecuada (Db2 MQ).

Conceptos relacionados

[Funciones de Db2 MQ y procedimientos almacenados XML de Db2 MQ](#)

Se pueden utilizar funciones y procedimientos almacenados de MQ de Db2 para enviar mensajes a una cola de mensajes o recibir mensajes desde una cola de mensajes.

[Funciones para la construcción de valores XML \(Db2 Programming for XML\)](#)

Fragmentación de documentos XML desde una cola de mensajes de MQ

Cuando recuperas datos XML de una cola de mensajes de MQ, puedes triturar esos datos en tablas de Db2 es para facilitar su recuperación.

Acerca de esta tarea

Procedimiento

Para destruir documentos XML de una cola de mensajes de MQ :

1. Recuperar el documento XML de una cola de mensajes de MQ mediante la función MQ adecuada.
2. Destruir el mensaje recuperado en tablas de Db2 es mediante el procedimiento almacenado de descomposición XML (XDBDECOMPXML).

Conceptos relacionados

[Funciones de Db2 MQ y procedimientos almacenados XML de Db2 MQ](#)

Se pueden utilizar funciones y procedimientos almacenados de MQ de Db2 para enviar mensajes a una cola de mensajes o recibir mensajes desde una cola de mensajes.

Tablas MQ de Db2

Las tablas MQ de Db2 contienen definiciones de política y servicio utilizadas por la Interfaz de cola de mensajes (MQI) basadas en funciones MQ de Db2. Debe llenar las tablas MQ de Db2 antes de poder utilizar las funciones basadas en MQI.

Las tablas MQ Db2 son SYSIBM.MQSERVICE_TABLE y SYSIBM.MQPOLICY_TABLE. Estas tablas son gestionadas por el usuario. Debe crearlos durante el proceso de instalación o migración. El trabajo de instalación DSNTIJRT crea estas tablas con una fila predeterminada en cada una.

Db2 Si anteriormente utilizó las funciones MQ de AMI basadas en AMI, utilizó archivos de configuración de AMI en lugar de estas tablas. Para utilizar las funciones de MQ basadas en MQI (Db2), debe mover los datos de esos archivos de configuración a las tablas de Db2 SYSIBM.MQSERVICE_TABLE y SYSIBM.MQPOLICY_TABLE.

La siguiente tabla describe las columnas de SYSIBM.MQSERVICE_TABLE.

Tabla 125. SYSIBM.MQSERVICE_TABLE descripciones de las columnas

Nombre de columna	Descripción
SERVICENAME	Esta columna contiene el nombre del servicio, que es un parámetro de entrada opcional de las funciones de MQ.
	Esta columna es la clave principal de la tabla "SYSIBM.MQSERVICE_TABLE".
QueueManager	Esta columna contiene el nombre del administrador de la cola con el que las funciones de MQ deben establecer una conexión.
COLA DE ENTRADA	Esta columna contiene el nombre de la cola desde la que las funciones de MQ van a enviar y recuperar mensajes.

Tabla 125. SYSIBM.MQSERVICE_TABLE descripciones de las columnas (continuación)

Nombre de columna	Descripción
CodedCharSetID	<p>Esta columna contiene el identificador del conjunto de caracteres para los datos de caracteres en los mensajes que se envían y reciben mediante las funciones de MQ.</p> <p>Esta columna corresponde al campo "CodedCharsetId" (mensaje de correo electrónico) en la estructura del descriptor de mensajes (MQMD). MQ las funciones utilizan el valor de esta columna para establecer el campo " CodedCharsetId " (valor de la propiedad).</p> <p>CodedCharsetId El valor predeterminado para esta columna es 0, que establece el campo MQCCSI_Q_MGR del MQMD en el valor MQCCSI_Q_MGR.</p>
ENCODING	<p>Esta columna contiene el valor de codificación de los datos numéricos de los mensajes enviados y recibidos por las funciones de MQ.</p> <p>Esta columna corresponde al campo de codificación en la estructura del descriptor de mensajes (MQMD). MQ utilizan el valor de esta columna para establecer el campo Codificación.</p> <p>El valor predeterminado para esta columna es 0, que establece el campo Codificación en el MQMD en el valor MQENC_NATIVE.</p>
DESCRIPCIÓN	Esta columna contiene la descripción del servicio.

La siguiente tabla describe las columnas de SYSIBM.MQPOLICY_TABLE.

Tabla 126. SYSIBM.MQPOLICY_TABLE descripciones de las columnas

Nombre de columna	Descripción
POLICYNAME	<p>Esta columna contiene el nombre de la política, que es un parámetro de entrada opcional de las funciones de MQ.</p> <p>Esta columna es la clave principal de la tabla "SYSIBM.MQPOLICY_TABLE".</p>
SEND_PRIORITY	<p>Esta columna contiene la prioridad del mensaje.</p> <p>Esta columna se corresponde con el campo Prioridad en la estructura del descriptor de mensajes (MQMD). MQ utilizan el valor de esta columna para establecer el campo Prioridad.</p> <p>El valor predeterminado para esta columna es -1, que establece el campo Prioridad en el MQMD con el valor MQQPRI_PRIORITY_AS_Q_DEF.</p>

Tabla 126. SYSIBM.MQPOLICY_TABLE descripciones de las columnas (continuación)

Nombre de columna	Descripción
SEND_PERSISTENCE	<p>Esta columna indica si el mensaje persiste a pesar de cualquier fallo del sistema o de los casos de reinicio del gestor de colas.</p> <p>Esta columna corresponde al campo Persistence en la estructura del descriptor de mensajes (MQMD). MQ utilizan el valor de esta columna para establecer el campo Persistencia.</p> <p>Esta columna puede tener los siguientes valores:</p> <p>Q Establece el campo Persistencia en el MQMD con el valor MQPER_PERSISTENCE_AS_Q_DEF. Este valor es el valor por omisión.</p> <p>Y Establece el campo Persistencia en el MQMD con el valor MQPER_PERSISTENT.</p> <p>N Establece el campo Persistencia en el MQMD con el valor MQPER_NOT_PERSISTENT.</p>
SEND_EXPIRY	<p>Esta columna contiene el tiempo de expiración del mensaje, en décimas de segundo.</p> <p>Esta columna corresponde al campo Expiry (Vencimiento) en la estructura del descriptor de mensajes (MQMD). MQ utilizan el valor de esta columna para establecer el campo Caducidad.</p> <p>El valor predeterminado es -1, que establece el campo Expiry en el valor MQEI_UNLIMITED.</p>
SEND_RETRY_COUNT	<p>Esta columna contiene el número de veces que la función " MQ " intentará enviar un mensaje si el procedimiento falla.</p> <p>El valor predeterminado es 5.</p>
SEND_RETRY_INTERVAL	<p>Esta columna contiene el intervalo, en milisegundos, entre cada intento de enviar un mensaje.</p> <p>El valor predeterminado es 1000.</p>

Tabla 126. SYSIBM.MQPOLICY_TABLE descripciones de las columnas (continuación)

Nombre de columna	Descripción
ENVIAR_NUEVO_CORRECID	<p>Esta columna especifica cómo se debe establecer el identificador de correlación si no se pasa un identificador de correlación como parámetro de entrada en la función " MQ ". El identificador de correlación se establece en el campo " CorrelId " en la estructura del descriptor de mensajes (MQMD).</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <p>N Establece el campo " CorrelId " en el MQMD en ceros binarios. Este valor es el valor por omisión.</p> <p>Y Especifica que el gestor de colas debe generar un nuevo identificador de correlación y establecer el campo " CorrelId " en el MQMD con ese valor. Este valor «Y» equivale a establecer la opción MQPMO_NEW_CORREL_ID en el campo Opciones de la estructura de opciones de mensajes de envío (MQPMO).</p>
SEND_RESPONSE_MSGID	<p>Esta columna especifica cómo se debe configurar el campo " MsgId " en la estructura del descriptor de mensajes (MQMD) para los mensajes de informe y respuesta.</p> <p>Esta columna corresponde al campo Informe en el MQMD. MQ utilizan el valor de esta columna para establecer el campo Informe.</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <p>N Establece la opción MQRO_NEW_MSG_ID en el campo Report del MQMD. Este valor es el valor por omisión.</p> <p>P Establece la opción MQRO_PASS_MSG_ID en el campo Report del MQMD.</p>

Tabla 126. SYSIBM.MQPOLICY_TABLE descripciones de las columnas (continuación)

Nombre de columna	Descripción
SEND_RESPONSE_CORRELID	<p>Esta columna especifica cómo se debe configurar el campo " CorrelID " en la estructura del descriptor de mensajes (MQMD) para los mensajes de informe y respuesta.</p> <p>Esta columna corresponde al campo Informe en el MQMD. MQ utilizan el valor de esta columna para establecer el campo Informe.</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <p>C Establece la opción MQRO_COPY_MSG_ID_TO_CORREL_ID en el campo Report del MQMD. Este valor es el valor por omisión.</p> <p>P Establece la opción MQRO_PASS_CORREL_ID en el campo Report del MQMD.</p>
SEND_EXCEPTION_ACTION	<p>Esta columna especifica qué hacer con el mensaje original cuando no se puede entregar a la cola de destino.</p> <p>Esta columna corresponde al campo Report (Informe) en la estructura descriptora de mensajes (MQMD). MQ utilizan el valor de esta columna para establecer el campo Informe.</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <p>Q Establece la opción MQRO_DEAD_LETTER_Q en el campo Report del MQMD. Este valor es el valor por omisión.</p> <p>D Establece la opción MQRO_DISCARD_MSG en el campo Report del MQMD.</p> <p>P Establece la opción MQRO_PASS_DISCARD_AND_EXPIRY en el campo Report del MQMD.</p>

Tabla 126. SYSIBM.MQPOLICY_TABLE descripciones de las columnas (continuación)

Nombre de columna	Descripción
SEND_REPORT_EXCEPTION	<p>Esta columna especifica si se debe generar un mensaje de informe de excepción cuando no se pueda entregar un mensaje a la cola de destino especificada y, en caso afirmativo, qué debe contener ese mensaje de informe.</p> <p>Esta columna corresponde al campo Report (Informe) en la estructura descriptora de mensajes (MQMD). MQ utilizan el valor de esta columna para establecer el campo Informe.</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <p>N Especifica que no se generará un mensaje de informe de excepción. No hay opciones establecidas en el campo Informe. Este valor es el valor por omisión.</p> <p>E Establece la opción MQRO_EXCEPTION en el campo Report del MQMD.</p> <p>D Establece la opción MQRO_EXCEPTION_WITH_DATA en el campo Report del MQMD.</p> <p>F Establece la opción MQRO_EXCEPTION_WITH_FULL_DATA en el campo Report del MQMD.</p>

Tabla 126. SYSIBM.MQPOLICY_TABLE descripciones de las columnas (continuación)

Nombre de columna	Descripción
ENVIAR_INFORME_COA	<p>Esta columna especifica si el administrador de la cola debe enviar un mensaje de informe de confirmación de llegada (COA) cuando el mensaje se coloca en la cola de destino y, en caso afirmativo, qué debe contener ese mensaje COA.</p> <p>Esta columna corresponde al campo Report (Informe) en la estructura descriptora de mensajes (MQMD). MQ utilizan el valor de esta columna para establecer el campo Informe.</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <p>N Especifica que no se debe enviar un mensaje COA. No hay opciones establecidas en el campo Informe. Este valor es el predeterminado</p> <p>C Establece la opción MQRO_COA en el campo Informe en el MQMD</p> <p>D Establece la opción MQRO_COA_WITH_DATA en el campo Report del MQMD.</p> <p>F Establece la opción MQRO_COA_WITH_FULL_DATA en el campo Report del MQMD.</p>

Tabla 126. SYSIBM.MQPOLICY_TABLE descripciones de las columnas (continuación)

Nombre de columna	Descripción
SEND_REPORT_COD	<p>Esta columna especifica si el administrador de la cola debe enviar un mensaje de informe de confirmación de entrega (COD) cuando una aplicación recupera y elimina un mensaje de la cola de destino y, en caso afirmativo, qué debe contener ese mensaje COD.</p> <p>Esta columna corresponde al campo Report (Informe) en la estructura descriptora de mensajes (MQMD). MQ utilizan el valor de esta columna para establecer el campo Informe.</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <p>N Especifica que no se debe enviar un mensaje de COD. No hay opciones establecidas en el campo Informe. Este valor es el valor por omisión.</p> <p>C Establece la opción MQRO_COD en el campo Report del MQMD.</p> <p>D Establece la opción MQRO_COD_WITH_DATA en el campo Report del MQMD.</p> <p>F Establece la opción MQRO_COD_WITH_FULL_DATA en el campo Report del MQMD.</p>

Tabla 126. SYSIBM.MQPOLICY_TABLE descripciones de las columnas (continuación)

Nombre de columna	Descripción
SEND_REPORT_EXPIRY	<p>Esta columna especifica si el gestor de colas debe enviar un mensaje de informe de caducidad si un mensaje se descarta antes de ser entregado a una aplicación y, en caso afirmativo, qué debe contener ese mensaje.</p> <p>Esta columna corresponde al campo Report (Informe) en la estructura descriptora de mensajes (MQMD). MQ utilizan el valor de esta columna para establecer el campo Informe.</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <p>N Especifica que no se debe enviar un mensaje de informe de caducidad. No hay opciones en el campo Informe, el valor predeterminado es "set.This".</p> <p>C Establece la opción MQRO_EXPIRATION en el campo Report del MQMD.</p> <p>D Establece la opción MQRO_EXPIRATION_WITH_DATA en el campo Report del MQMD.</p> <p>F Establece la opción MQRO_EXPIRATION_WITH_FULL_DATA en el campo Report del MQMD.</p>

Tabla 126. SYSIBM.MQPOLICY_TABLE descripciones de las columnas (continuación)

Nombre de columna	Descripción
SEND_REPORT_ACTION	<p>Esta columna especifica si la aplicación receptora envía una notificación de acción positiva (PAN), una notificación de acción negativa (NAN) o ambas.</p> <p>Esta columna corresponde al campo Report (Informe) en la estructura descriptora de mensajes (MQMD). MQ utilizan el valor de esta columna para establecer el campo Informe.</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <ul style="list-style-type: none"> N Especifica que no se debe enviar ninguna notificación. No hay opciones establecidas en el campo Informe. Este valor es el valor por omisión. P Establece la opción MQRO_PAN en el campo Report del MQMD. T Establece la opción MQRO_NAN en el campo Report del MQMD. B Establece las opciones MQRO_PAN y MQRO_NAN en el campo Report del MQMD.
SEND_MSG_TYPE	<p>Esta columna contiene el tipo de mensaje.</p> <p>Esta columna corresponde al campo " MsqType " (mensaje de correo electrónico) en la estructura del descriptor de mensajes (MQMD). MQ las funciones utilizan el valor de esta columna para establecer el campo " MsqType ".</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <ul style="list-style-type: none"> DTG Establece el campo MsgType en el MQMD a MQMT_DATAGRAM. Este valor es el valor por omisión. REQ Establece el campo MsgType en el MQMD a MQMT_REQUEST. RLY Establece el campo MsgType en el MQMD a MQMT_REPLY. rpt Establece el campo " MsgType " en el MQMD en MQMT_REPORT.

Tabla 126. SYSIBM.MQPOLICY_TABLE descripciones de las columnas (continuación)

Nombre de columna	Descripción
RESPONDER_A_P	<p>Esta columna contiene el nombre de la cola de mensajes a la que la aplicación que emitió la llamada MQGET debe enviar mensajes de respuesta e informe.</p> <p>Esta columna corresponde al campo " ReplyToQ " en la estructura del descriptor de mensajes (MQMD). MQ las funciones utilizan el valor de esta columna para establecer el campo " ReplyToQ ".</p> <p>El valor predeterminado para esta columna es SAME AS INPUT_Q, que establece el nombre en el nombre de la cola que se define en el servicio que se utilizó para enviar el mensaje. Si no se especificó ningún servicio, el nombre se establece en DB2MQ_DEFAULT_Q, que es el nombre de la cola de entrada para el servicio predeterminado.</p>
RESPONDER_AL_GERENTE	<p>Esta columna contiene el nombre del gestor de colas al que se deben enviar los mensajes de respuesta e informe.</p> <p>Esta columna corresponde al campo " ReplyToQMGr " en la estructura del descriptor de mensajes (MQMD). MQ las funciones utilizan el valor de esta columna para establecer el campo " ReplyToQMGr ".</p> <p>El valor predeterminado para esta columna es SAME AS INPUT_QMGR, que establece el nombre en el nombre del administrador de colas definido en el servicio que se utilizó para enviar el mensaje. Si no se especificó ningún servicio, el nombre se establece en el nombre del administrador de la cola para el servicio predeterminado.</p>
RCV_WAIT_INTERVAL	<p>Esta columna contiene el tiempo, en milisegundos, que Db2 debe esperar a que lleguen los mensajes a la cola.</p> <p>Esta columna corresponde al campo " WaitInterval " en la estructura de opciones de mensajes de obtención (MQGMO). MQ las funciones utilizan el valor de esta columna para establecer el campo " WaitInterval ".</p> <p>El valor predeterminado es 10.</p>

Tabla 126. SYSIBM.MQPOLICY_TABLE descripciones de las columnas (continuación)

Nombre de columna	Descripción
rcv_convert	<p>Esta columna indica si se deben convertir los datos de la aplicación en el mensaje para que se ajusten a los valores de CodedCharSetId y Encoding del servicio MQ especificado.</p> <p>Esta columna corresponde al campo Opciones en la estructura de opciones de mensajes get (MQGMO). MQ utilizan el valor de esta columna para establecer el campo Opciones.</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <p>Y Establece la opción MQGMO_CONVERT en el campo Opciones en el MQGMO. Este valor es el valor por omisión.</p> <p>N Especifica que no se debe convertir ningún dato.</p>
RCV_ACCEPT_TRUNC_MSG	<p>Esta columna especifica el comportamiento de la función " MQ " cuando se recuperan mensajes de gran tamaño.</p> <p>Esta columna corresponde al campo Opciones en la estructura de opciones de mensajes get (MQGMO). MQ utilizan el valor de esta columna para establecer el campo Opciones.</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <p>Y Establece la opción MQGMO_ACCEPT_TRUNCATED_MSG en el campo Opciones del MQGMO. Este valor es el valor por omisión.</p> <p>N Especifica que no se deben truncar los mensajes. Si el mensaje es demasiado grande para caber en el búfer, la función MQ finaliza con un error.</p> <p>Recomendación: Establezca esta columna en S. En este caso, si el búfer de mensajes es demasiado pequeño para contener el mensaje completo, la función MQ puede llenar el búfer con la mayor cantidad de mensaje que el búfer pueda contener.</p>

Tabla 126. SYSIBM.MQPOLICY_TABLE descripciones de las columnas (continuación)

Nombre de columna	Descripción
REV_OPEN_SHARED	<p>Esta columna especifica el modo de cola de entrada cuando se recuperan los mensajes.</p> <p>Esta columna corresponde al parámetro Options para una llamada MQOPEN. MQ utilizan el valor de esta columna para establecer el parámetro Opciones.</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <p>S Establece la opción MQOO_INPUT_SHARED. Este valor es el valor por omisión.</p> <p>E Establece la opción MQOO_INPUT_EXCLUSIVE de la opción MQ.</p> <p>D Establece la opción MQOO_INPUT_AS_Q_DEF de la opción MQ.</p>
SYNCPOINT	<p>Esta columna indica si la función de MQ o debe operar dentro del protocolo para una unidad de trabajo normal.</p> <p>Esta columna puede tener uno de los valores siguientes:</p> <p>Y Especifica que la función de la MQ es operar dentro del protocolo para una unidad de trabajo normal. Utilice este valor para entornos de compromiso de dos fases. Este valor es el valor por omisión.</p> <p>N Especifica que la función de la MQ es operar fuera del protocolo para una unidad de trabajo normal. Utilice este valor para entornos de compromiso de una fase.</p>
DESC	Esta columna contiene la descripción de la política.

Referencia relacionada

[Entornos de WLM principales para rutinas suministradas por Db2\(Db2 Installation and Migration\)](#)

[Guía de consulta para el desarrollo de aplicaciones](#)

Mensajes básicos con IBM MQ

La forma más básica de mensajería con las funciones Db2 MQ se produce cuando todas las aplicaciones de base de datos se conectan al mismo servidor de base de datos Db2 . Los clientes pueden ser locales respecto al servidor de bases de datos o distribuidos en un entorno de red.

En un escenario simple, el cliente A invoca la función MQSEND para enviar una serie definida por el usuario a la ubicación definida por el servicio predeterminado. Db2 ejecuta las funciones de MQ que realizan esta operación en el servidor de base de datos. Más tarde, el cliente B invoca la función MQRECEIVE para eliminar el mensaje al principio de la cola definida por el servicio predeterminado y

devolverlo al cliente. Db2 ejecuta las funciones de MQ que realizan esta operación en el servidor de la base de datos.

Los clientes de la base de datos pueden utilizar la mensajería simple de varias maneras:

- Recopilación de datos

La información se recibe en forma de mensajes de una o más fuentes. Una fuente de información puede ser cualquier información. Los datos se reciben de las colas y se almacenan en tablas de bases de datos para su procesamiento adicional.

- Distribución de la carga de trabajo

Las solicitudes de trabajo se publican en una cola que comparten varias instancias de la misma aplicación. Cuando una instancia de la aplicación está lista para realizar parte del trabajo, esta recibe un mensaje con una petición de trabajo de la parte superior de la cola. Varias instancias de la aplicación pueden compartir la carga de trabajo representada por una única cola de peticiones agrupadas.

- Señalización de aplicaciones

En una situación en la que colaboran varios procesos, los mensajes se utilizan a menudo para coordinar sus esfuerzos. Estos mensajes pueden contener órdenes o solicitudes de trabajo que se deben realizar. Para más información sobre esta técnica, consulte [“Conectividad de aplicación a aplicación con IBM MQ” en la página 845.](#)

El siguiente escenario amplía los mensajes básicos para incorporar mensajes remotos. Supongamos que la máquina A envía un mensaje a la máquina B.

1. El cliente de MQ Db2 ejecuta una llamada de función MQSEND, especificando un servicio de destino que se ha definido como una cola remota en la máquina B.
2. Las funciones de MQ realizan el trabajo de enviar el mensaje. El WebSphere MQ servidor de la máquina A acepta el mensaje y garantiza que lo entregará al destino definido por el servicio y la configuración actual de MQ de la máquina A. El servidor determina que el destino es una cola en la máquina B. El servidor intenta entonces entregar el mensaje al WebSphere MQ servidor de la máquina B, y vuelve a intentarlo si es necesario.
3. El servidor IBM MQ de la máquina B acepta el mensaje del servidor de la máquina A y lo coloca en la cola de destino de la máquina B.
4. Un cliente de IBM MQ en la máquina B solicita el mensaje que está al principio de la cola.

Envío de mensajes con IBM MQ

Cuando envías mensajes con IBM MQ, tú eliges qué datos enviar, dónde enviarlos y cuándo enviarlos. Este tipo de mensajería se denomina «*enviar y olvidar*»: el remitente envía un mensaje y confía en IBM MQ para asegurarse de que el mensaje llegue a su destino.

Procedimiento

Para enviar mensajes con IBM MQ, utilice MQSEND.

El contenido del mensaje puede ser cualquier combinación de sentencias de SQL, expresiones, funciones y datos especificados por el usuario. Debido a que esta función MQSEND utiliza una confirmación en dos fases, la instrucción COMMIT garantiza que el mensaje se añada a la cola de mensajes de espera (MQ).

Ejemplos

Si envía más de una columna de información, separe las columnas con los caracteres || ' ' ||.

MQSEND (LASTNAME || ' ' || FIRSTNAME)

Los siguientes ejemplos utilizan el esquema DB2MQ para la confirmación en dos fases, con el servicio predeterminado Db2. DEFAULT.SERVICE y la política predeterminada Db2. DEFAULT.POLICY.

La siguiente instrucción SQL SELECT envía un mensaje que consiste en la cadena "Mensaje de prueba":

```
SELECT DB2MQ.MQSEND ('Testing msg')
  FROM SYSIBM.SYSDUMMY1;
  COMMIT;
```

La función MQSEND se invoca una vez porque SYSIBM.SYSDUMMY1 solo tiene una fila. Debido a que esta función MQSEND utiliza una confirmación en dos fases, la instrucción COMMIT garantiza que el mensaje se añada a la cola.

Cuando se utiliza el commit monofásico, no es necesario utilizar una sentencia COMMIT. Por ejemplo:

```
SELECT DB2MQ.MQSEND ('Testing msg')
  FROM SYSIBM.SYSDUMMY1;
```

La operación MQ hace que el mensaje se añada a la cola.

Supongamos que tiene una tabla EMPLOYEE, con las columnas VARCHAR LASTNAME, FIRSTNAME y DEPARTMENT. Para enviar un mensaje que contiene esta información para cada empleado en DEPARTMENT 5LGA, emita la siguiente sentencia SELECT de SQL:

```
SELECT DB2MQ.MQSEND (LASTNAME || ' ' || FIRSTNAME || ' ' || DEPARTMENT)
  FROM EMPLOYEE WHERE DEPARTMENT = '5LGA';
  COMMIT;
```

Referencia relacionada

[MQSEND función escalar \(Db2 SQL\)](#)

Recuperación de mensajes con IBM MQ

Con IBM MQ, los programas pueden leer o recibir mensajes. Tanto la lectura como la recepción devuelven el mensaje al principio de la cola. Sin embargo, la operación de lectura no elimina el mensaje de la cola, mientras que la operación de recepción sí lo hace.

Acerca de esta tarea

Un mensaje que se recupera mediante una operación de recepción solo puede recuperarse una vez, mientras que un mensaje que se recupera mediante una operación de lectura permite recuperar el mismo mensaje muchas veces.

Ejemplos

Los siguientes ejemplos utilizan el esquema DB2MQ2N para la confirmación en dos fases, con el servicio predeterminado Db2.DEFAULT.SERVICE y la política predeterminada Db2.DEFAULT.POLICY.

Ejemplo

La siguiente instrucción SQL SELECT lee el mensaje al principio de la cola que se especifica por el servicio y la política predeterminados:

```
SELECT DB2MQ2N.MQREAD()
  FROM SYSIBM.SYSDUMMY1;
```

La función MQREAD se invoca una vez porque SYSIBM.SYSDUMMY1 solo tiene una fila. La sentencia SELECT devuelve una cadena VARCHAR(4000). Si no hay mensajes disponibles para leer, se devuelve un valor nulo. Como MQREAD no cambia la cola, no es necesario utilizar una instrucción COMMIT.

Ejemplo

La siguiente instrucción SQL SELECT hace que el contenido de una cola se materialice como una tabla de tipo "Db2":

```
SELECT T.*
  FROM TABLE(DB2MQ2N.MQREADALL()) T;
```

La tabla de resultados T de la función de tabla consta de todos los mensajes de la cola, que está definida por el servicio predeterminado, y los metadatos sobre esos mensajes. La primera columna de la tabla de resultados materializados es el mensaje en sí, y las columnas restantes contienen los metadatos. La sentencia SELECT devuelve tanto los mensajes como el metadata. To devuelve solo los mensajes, emita la siguiente sentencia:

```
SELECT T.MSG  
  FROM TABLE(DB2MQ2N.MQREADALL()) T;
```

La tabla de resultados T de la función de tabla consta de todos los mensajes de la cola, que está definida por el servicio predeterminado, y los metadatos sobre esos mensajes. Esta instrucción SELECT devuelve solo los mensajes.

Ejemplo

La siguiente instrucción SQL SELECT recibe (elimina) el mensaje al principio de la cola:

```
SELECT DB2MQ2N.MQRECEIVE()  
  FROM SYSIBM.SYSDUMMY1;  
 COMMIT;
```

La función MQRECEIVE se invoca una vez porque SYSIBM.SYSDUMMY1 solo tiene una fila. La sentencia SELECT devuelve una cadena VARCHAR(4000). Debido a que esta función MQRECEIVE utiliza una confirmación de dos fases, la instrucción COMMIT garantiza que el mensaje se elimine de la cola. Si no hay mensajes disponibles para recuperar, se devuelve un valor nulo y la cola no cambia.

Ejemplo

Supongamos que tiene una tabla MESSAGES con una sola columna VARCHAR(2000). La siguiente instrucción SQL INSERT inserta todos los mensajes de la cola de servicio predeterminada en la tabla MESSAGES de su base de datos Db2 :

```
INSERT INTO MESSAGES  
SELECT T.MSG  
  FROM TABLE(DB2MQ2N.MQRECEIVEALL()) T;  
 COMMIT;
```

La tabla de resultados T de la función de tabla consta de todos los mensajes de la cola de servicio predeterminada y los metadatos sobre esos mensajes. La sentencia SELECT devuelve solo los mensajes. La instrucción INSERT guarda los mensajes en una tabla de su base de datos.

Conectividad de aplicación a aplicación con IBM MQ

La conectividad de aplicación a aplicación se utiliza normalmente cuando se reúne un conjunto diverso de subsistemas de aplicaciones. Para facilitar la integración de aplicaciones, WebSphere MQ proporciona los medios para interconectar aplicaciones.

El método *de solicitud y respuesta* es muy común cuando se interconectan aplicaciones.

Método de comunicación de solicitud y respuesta

El método de petición y respuesta permite a una aplicación solicitar los servicios de otra aplicación. Una forma de hacerlo es que el peticionario envíe un mensaje al proveedor de servicio para solicitar que se realice algún trabajo. Cuando el trabajo haya finalizado, el proveedor puede decidir enviar los resultados, o simplemente una confirmación de finalización, al solicitante. A menos que el solicitante espere una respuesta antes de continuar, IBM MQ debe proporcionar una forma de asociar la respuesta con su solicitud.

IBM MQ proporciona un identificador de correlación para correlacionar mensajes en un intercambio entre un solicitante y un proveedor. El peticionario marca un mensaje con un identificador de correlación conocido. El proveedor marca su respuesta con el mismo identificador de correlación. Para recuperar la respuesta asociada, el peticionario proporciona el identificador de correlación al recibir mensajes de la cola. El primer mensaje con un identificador de correlación coincidente se devuelve al solicitante.

Los siguientes ejemplos utilizan el esquema DB2MQ para la confirmación en una sola fase.

Ejemplos de comunicación de solicitud y respuesta

Ejemplo

La siguiente instrucción SQL SELECT envía un mensaje que consiste en la cadena "Msg with corr id" al servicio MYSERVICE, utilizando la política MYPOLICY con el identificador de correlación CORRID1:

```
SELECT DB2MQ.MQSEND ('MYSERVICE', 'MYPOLICY', 'Msg with corr id', 'CORRID1')
  FROM SYSIBM.SYSDUMMY1;
```

La función MQSEND se invoca una vez porque SYSIBM.SYSDUMMY1 solo tiene una fila. Debido a que este MQSEND utiliza confirmación monofásica, el procedimiento IBM MQ añade el mensaje a la cola y no es necesario utilizar una instrucción COMMIT.

Ejemplo

La siguiente instrucción SQL SELECT recibe el primer mensaje que coincide con el identificador CORRID1 de la cola especificada por el servicio MYSERVICE, utilizando la política MYPOLICY:

```
SELECT DB2MQ.MQRECEIVE ('MYSERVICE', 'MYPOLICY', 'CORRID1')
  FROM SYSIBM.SYSDUMMY1;
```

La sentencia SELECT devuelve una cadena VARCHAR(4000). Si no hay mensajes disponibles con este identificador de correlación, se devuelve un valor nulo y la cola no cambia.

Mensajería asíncrona en Db2 for z/OS

Los programas pueden comunicarse entre sí enviando datos en mensajes en lugar de utilizar construcciones como llamadas a procedimientos remotos síncronos. Con la mensajería asíncrona, el programa que envía el mensaje continúa con su procesamiento después de enviar el mensaje, sin esperar una respuesta.

Si el programa necesita información de la respuesta, el programa suspende el proceso y espera un mensaje de respuesta. Si los programas de mensajes utilizan una cola intermedia que contiene mensajes, no es necesario que el programa solicitante y el programa emisor se ejecuten al mismo tiempo. El programa solicitante coloca un mensaje de petición en una cola y, a continuación, sale. El programa receptor recupera la petición de la cola y la procesa.

Las operaciones asíncronas requieren que el proveedor de servicio sea capaz de aceptar peticiones de clientes sin aviso previo. Un oyente asíncrono es un programa que supervisa los transportadores de mensajes, como WebSphere MQ, y realiza acciones basadas en el tipo de mensaje. Un oyente asíncrono puede utilizar WebSphere MQ para recibir todos los mensajes que se envían a un punto final. Un oyente asíncrono también puede registrar una suscripción con una infraestructura de publicación o suscripción para restringir los mensajes recibidos a los mensajes que cumplan restricciones especificadas.

Ejemplos : Los siguientes ejemplos muestran algunos usos comunes de la mensajería asíncrona:

Acumulador de mensajes

Puede acumular los mensajes que se envían de forma asíncrona de forma que el oyente comprueba los mensajes y almacena estos mensajes automáticamente en una base de datos. Esta base de datos, que actúa como un acumulador de mensajes, puede guardar todos los mensajes para un punto final determinado, tal como un seguimiento de auditoría. El oyente asíncrono puede suscribirse a un subconjunto de mensajes como, por ejemplo, *guardar sólo las operaciones bursátiles de gran valor*. El acumulador de mensajes almacena mensajes completos y no permite la selección, transformación o asignación de contenidos de mensajes a estructuras de bases de datos. El acumulador de mensajes no responde a los mensajes.

Manejador de sucesos de mensajes

El controlador de eventos asíncrono escucha los mensajes e invoca el controlador adecuado (como un procedimiento almacenado) para el punto final del mensaje. Puede llamar a cualquier procedimiento almacenado arbitrario. El oyente asíncrono le permite seleccionar, correlacionar o volver a formatear contenido del mensaje para la inserción en una o más estructuras de datos.

La mensajería asíncrona tiene las siguientes ventajas:

- No es necesario que el cliente y la base de datos estén disponibles simultáneamente. Si el cliente está disponible de forma intermitente, o si el cliente falla entre el momento en que se emite la solicitud y se envía la respuesta, aún es posible que el cliente reciba la respuesta. O bien, si el cliente está en un sistema portátil y se desconecta de la base de datos, y si se envía una respuesta, el cliente puede seguir recibiendo la respuesta.
- El contenido de los mensajes en la base de datos contiene información sobre cuándo procesar determinadas peticiones. Los mensajes de la base de datos utilizan prioridades y el contenido de la petición para determinar cómo planificar las peticiones.
- Un oyente de mensajes asíncronos puede delegar una petición a un nodo distinto. Puede reenviar la petición a un segundo sistema para completar el proceso. Cuando se completa la petición, el segundo sistema devuelve una respuesta directamente al punto final especificado en el mensaje.
- Un oyente asíncrono puede responder a un mensaje de un cliente suministrado o de una aplicación definida por el usuario. El número de entornos que pueden actuar como cliente de base de datos se ha ampliado considerablemente. Clientes como equipos de automatización de fábricas, dispositivos ubicuos o controladores integrados pueden comunicarse con Db2 , ya sea directamente a través de IBM MQ o a través de alguna pasarela que admita WebSphere MQ.

MQListener en Db2 for z/OS

Db2 for z/OS proporciona un oyente asíncrono, MQListener. MQListener es un marco para tareas que leen de colas de peticiones (IBM MQ) y llaman a procedimientos almacenados (Db2) con mensajes a medida que llegan esos mensajes.

MQListener combina la mensajería con las operaciones de base de datos. Puede configurar el demonio MQListener para que escuche las colas de mensajes IBM MQ que especifique en una base de datos de configuración. MQListener lee los mensajes que llegan de la cola y llama a procedimientos almacenados de T Db2 , utilizando los mensajes como parámetros de entrada. Si el mensaje requiere una respuesta, MQListener crea una respuesta a partir del resultado generado por el procedimiento almacenado. El orden de recuperación de mensajes se fija primero en la prioridad más alta, y luego dentro de cada prioridad el primer mensaje recibido es el primero en atenderse.

MQListener se ejecuta como un único proceso multiproceso en z/OS UNIX System Services. Cada hebra o tarea establece una conexión con su cola de mensajes configurada para la entrada. Cada tarea también se conecta a una base de datos de procedimientos almacenados (Db2) en la que se ejecuta el procedimiento almacenado. La información sobre la cola y el procedimiento almacenado se almacenan en una tabla en la base de datos de configuración. La combinación de la cola y del procedimiento almacenado es una tarea.

MQListener las tareas se agrupan en configuraciones con nombre. Por omisión, el nombre de configuración está vacío. Si no especifica el nombre de una configuración para una tarea, MQListener utiliza la configuración con un nombre vacío.

Soporte de transacciones en MQListener

Hay soporte para entornos de confirmación tanto monofásicos como bifásicos. Un entorno de compromiso de una fase es aquel en el que las interacciones de la base de datos y las interacciones de la aplicación (MQ) son independientes. Un entorno de compromiso de dos fases es aquel en el que las interacciones de la base de datos y las interacciones de la aplicación (MQ) se combinan en una sola unidad de trabajo.

« db2mqln1 » es el nombre del ejecutable para una fase y « db2mqln2 » es el nombre del ejecutable para dos fases.

Orden lógico de los mensajes en MQListener

La versión de confirmación en dos fases del procedimiento almacenado " MQListener " procesa los mensajes que están en un grupo en orden lógico. La versión de compromiso monofásico del procedimiento almacenado de MQListener procesa los mensajes que están en un grupo en orden físico.

MQListener interfaces de procedimientos almacenados

La interfaz de programación de aplicaciones (MQListener , API) admite dos formatos de procedimiento almacenado: con dos o tres parámetros.

El tipo de datos y la longitud de los parámetros del procedimiento almacenado se determinan cuando se inicia el procedimiento almacenado (MQListener). Si cambia el tipo de datos o la longitud de los parámetros, el cambio surtirá efecto cuando reinicie el navegador (MQListener). Puede utilizar los siguientes comandos para reiniciar el navegador (MQListener):

```
mqlistener-command mqlistener-command admin  
-adminQueue adminqueue-name  
-adminQMgr adminqueueumanager-name  
-adminCommand restart
```

MQListener interfaz de procedimiento almacenado con dos parámetros

Esta interfaz de procedimiento almacenado para MQListener toma el mensaje entrante como entrada y devuelve la respuesta, que podría ser NULL, como salida. Por ejemplo:

```
CREATE schema.proc(  
  IN INMSG inMsgType,  
  OUT OUTMSG outMsgType)...
```

El tipo de datos para INMSG y el tipo de datos para OUTMSG pueden ser VARCHAR, VARBINARY, CLOB o BLOB, de cualquier longitud, y se determinan al inicio. El tipo de datos de entrada y el tipo de datos de salida pueden ser tipos de datos distintos. Si un mensaje entrante es una solicitud o un datagrama y tiene una cola de respuesta especificada, el mensaje en OUTMSG se envía a la cola especificada. El mensaje entrante puede ser uno de los siguientes tipos de mensaje:

- Datagrama
- Se ha solicitado un datagrama con informe
- Solicitar mensaje con respuesta
- Solicitar mensaje con respuesta y reporte solicitado

MQListener interfaz de procedimiento almacenado con tres parámetros

Esta interfaz de procedimiento almacenado para MQListener tiene parámetros con la siguiente información:

- Un mensaje entrante como entrada
- Una respuesta, que podría ser NULL, como salida
- Un encabezado de mensaje, que puede ser de entrada o de salida

Por ejemplo:

```
CREATE schema.proc(  
  IN INMSG inMsgType,  
  OUT OUTMSG outMsgType,  
  INOUT MSGHEADER msgHeaderType)...
```

El tipo de datos para INMSG y el tipo de datos para OUTMSG pueden ser VARCHAR, VARBINARY, CLOB o BLOB, de cualquier longitud, y se determinan al inicio. El tipo de datos de entrada y el tipo de datos de salida pueden ser tipos de datos distintos. Si un mensaje entrante es una solicitud o un datagrama y tiene una cola de respuesta especificada, el mensaje en OUTMSG se envía a la cola especificada. El mensaje entrante puede ser uno de los siguientes tipos de mensaje:

- Datagrama
- Se ha solicitado un datagrama con informe
- Solicitar mensaje con respuesta

- Solicitar mensaje con respuesta y reporte solicitado

El tipo de datos para MSGHEADER puede ser VARBINARY o BLOB. La longitud mínima de MSGHEADER es 324, que es el tamaño de la estructura del descriptor de mensajes de la cola de mensajes (MQMD) para mensajes de tipo IBM MQ .

MQListener pasa el encabezado del mensaje al procedimiento almacenado en el parámetro MSGHEADER. El procedimiento almacenado puede obtener las propiedades del descriptor de mensajes del parámetro MSGHEADER. Si el mensaje es un mensaje de solicitud, el procedimiento almacenado puede especificar las propiedades de la cola de respuesta y del administrador de la cola de respuesta en el parámetro MSGHEADER. El mensaje de salida en OUTMSG se envía a esa cola especificada.

Configuración de MQListener en Db2 for z/OS

Antes de poder utilizar MQListener, debe configurar su entorno de base de datos para que sus aplicaciones puedan utilizar la mensajería con operaciones de base de datos. También debe configurar IBM MQ para MQListener.

Antes de empezar

Asegúrese de que la persona que realiza el trabajo de instalación tiene la autoridad necesaria para crear la tabla de configuración y vincular los DBRM.

Acerca de esta tarea

Los siguientes trabajos de muestra para MQListener se encuentran en el conjunto de datos de *prefijo.SDSNSAMP* .

DSNTIJML

Un trabajo de muestra para extraer archivos de la biblioteca y configurar MQListener en z/OS UNIX System Services

DSNTEJML

Un trabajo de muestra que ejecuta scripts para configurar MQListener.

DSNTEJSP

Un trabajo de ejemplo para extraer archivos tar de la biblioteca en z/OS UNIX System Services al aplicar PTF para MQListener.

Procedimiento

Para configurar el entorno para MQListener y desarrollar una aplicación sencilla que reciba un mensaje, inserte el mensaje en una tabla y cree un mensaje de respuesta sencillo, siga estos pasos:

1. Configure MQListener para que se ejecute en el entorno de Db2 , de modo que sus aplicaciones puedan utilizar la mensajería con operaciones de base de datos, completando los siguientes pasos:
 - a) En z/OS UNIX System Services, la ruta predeterminada para MQListener es /usr/lpp/db2c10/mql. El archivo tar de mqlsn.tar.Z para MQListener se encuentra en esta ruta. Si MQListener no está instalado en la ruta predeterminada, sustituya todas las apariciones de la ruta predeterminada en los ejemplos DSNTEJML, DSNTEJSP y DSNTIJML por el nombre de la ruta donde está instalado MQListener antes de ejecutar DSNTIJML.
 - b) Personalizar y ejecutar el trabajo de instalación DSNTIJML. Completa las siguientes acciones:
 - i) Extrae los archivos y bibliotecas necesarios a z/OS UNIX System Services en la ruta donde está instalado MQListener .
 - ii) Crea la tabla de configuración de MQListener (SYSML.LISTENERS) en la base de datos predeterminada DSNDB04.
 - iii) Vincula los DBRM al plan DB2MQSLN.

Aplicación de PTF:

Al aplicar los PTF de APAR para MQListener, extraiga el archivo tar y vuelva a enlazar los archivos de la biblioteca MQListener en z/OS UNIX System Services ejecutando los siguientes pasos del trabajo DSNTIJML: COPYHFS, UNTARLN, BINDBRM. Si la tabla SYSMQL.LISTENERS ya está definida, debe omitir el paso CREATBL.

- c) Siga las instrucciones del archivo README en la ruta de instalación de MQListener para completar el proceso de configuración.
2. Configure IBM MQ para MQListener.

Puede ejecutar una aplicación simple de MQListener con una configuración simple de IBM MQ .

Es posible que aplicaciones más complejas necesiten una configuración más compleja. Configure al menos dos tipos de entidades de e IBM MQ : el gestor de colas y algunas colas locales. Configure estas entidades para su uso en casos como la gestión de transacciones, la cola de cartas muertas, la cola de retrocesos y el umbral de reintentos de retroceso.

- a) Crear IBM MQ QueueManager. Defina el subsistema IBM MQ en z/OS y, a continuación, emita el siguiente comando desde una z/OS consola para iniciar el gestor de colas, donde *el prefijo de comando* es el prefijo de comando para el subsistema IBM MQ .

```
command-prefix START QMGR
```

- b) Crear colas en IBM MQ QueueManager:

Por ejemplo, en una aplicación sencilla de MQListener , normalmente se utilizan las siguientes colas de IBM MQ :

Cola de mensajes no entregados

La cola de mensajes no entregados en IBM MQ contiene mensajes que no se pueden procesar. MQListener utiliza esta cola para retener las respuestas que no se pueden entregar, por ejemplo, porque la cola a la que se deben enviar las respuestas está llena. Una cola de mensajes no entregados es útil en cualquier instalación de MQ especialmente para recuperar mensajes que no están enviados.

Cola de restitución

Para las tareas de e MQListener , que utilizan el compromiso en dos fases, la cola de retroceso tiene una finalidad similar a la de la cola de mensajes no entregados. MQListener coloca la solicitud original en la cola de retroceso después de que la solicitud se revierta un número específico de veces (llamado umbral de retroceso).

Cola de administración

La cola de administración se utiliza para enrutar mensajes de control, como el apagado y el reinicio, a MQListener. Si no proporciona una cola de administración, la única forma de cerrar MQListener es emitir un comando kill.

Colas de entrada y salida de la aplicación

La aplicación utiliza colas de entrada y colas de salida. La aplicación recibe mensajes de la cola de entrada y envía respuestas y excepciones a la cola de salida.

Cree sus colas locales utilizando la utilidad CSQUTIL o utilizando operaciones y paneles de control de IBM MQ desde ISPF (csqorexx). El siguiente es un ejemplo del JCL que se utiliza para crear sus colas locales. En este ejemplo, MQND es el nombre del administrador de colas:

```
/*
/* ADMIN_Q      : Admin queue
/* BACKOUT_Q    : Backout queue
/* IN_Q         : Input queue having a backout queue with threshold=3
/* REPLY_Q      : output queue or reply queue
/* DEADLETTER_Q: Dead letter queue
*/
//DSNTECU EXEC PGM=CSQUTIL,PARM='MQND'
//STEPLIB  DD DSN=MQS.SCSQANLE,DISP=SHR
//          DD DSN=MQS.SCSQAUTH,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
      COMMAND   DDNAME(CREATEQ)
/*
//CREATEQ DD *
      DEFINE QLOCAL('ADMIN_Q') REPLACE +
      DESCR('INPUT-OUTPUT') +
```

```

PUT(ENABLED)      +
DEFPRTY(0)       +
DEFPSIST(NO)     +
SHARE            +
DEFSOFT(SHARED) +
GET(ENABLED)

DEFINE QLOCAL('BACKOUT_Q') REPLACE +
DESCR('INPUT-OUTPUT')   +
PUT(ENABLED)      +
DEFPRTY(0)       +
DEFPSIST(NO)     +
SHARE            +
DEFSOFT(SHARED) +
GET(ENABLED)

DEFINE QLOCAL('REPLY_Q') REPLACE +
DESCR('INPUT-OUTPUT')   +
PUT(ENABLED)      +
DEFPRTY(0)       +
DEFPSIST(NO)     +
SHARE            +
DEFSOFT(SHARED) +
GET(ENABLED)

DEFINE QLOCAL('IN_Q') REPLACE +
DESCR('INPUT-OUTPUT')   +
PUT(ENABLED)      +
DEFPRTY(0)       +
DEFPSIST(NO)     +
SHARE            +
DEFSOFT(SHARED) +
GET(ENABLED)
BOQNAME('BACKOUT_Q') +
BOTHRESH(3)

DEFINE QLOCAL('DEADLETTER_Q') REPLACE +
DESCR('INPUT-OUTPUT')   +
PUT(ENABLED)      +
DEFPRTY(0)       +
DEFPSIST(NO)     +
SHARE            +
DEFSOFT(SHARED) +
GET(ENABLED)

/* ALTER QMGR DEADQ ('DEADLETTER_Q') REPLACE

```

3. Configurar tareas de MQListener . Para obtener más información, consulte “Configuración de tareas de MQListener ” en la página 853.
4. Crear un procedimiento almacenado que MQListener utiliza para almacenar mensajes en una tabla. Consulte “Creación de un procedimiento almacenado de muestra para usar con MQListener” en la página 855 para obtener detalles.
5. Ejecute una aplicación sencilla de MQListener .

Variables de entorno para el registro y el seguimiento MQListener

Varias variables de entorno controlan el registro y el seguimiento de MQListener. Estas variables se definen en el archivo .profile.

MQLSNTRC

Cuando esta variable de entorno se establece en 1, MQListener escribe la entrada de la función, los datos y los puntos de salida en un archivo HFS o zFS único. Se genera un archivo de seguimiento único cada vez que se ejecuta cualquiera de los comandos MQListener . IBM Support utiliza este archivo de seguimiento para la depuración. No defina esta variable a menos que IBM Support se lo solicite.

Si ha habilitado el seguimiento, cuando el demonio MQListener se está ejecutando, escribe en el archivo de seguimiento. Por lo tanto, si abre el archivo de seguimiento mientras el demonio MQListener se está ejecutando, debe abrirlo solo en modo de lectura.

MQLSNLOG

El archivo de registro contiene información de diagnóstico sobre los principales eventos. Esta variable de entorno se establece con el nombre del archivo donde se escribe toda la información de registro. Todas las instancias del demonio MQListener que ejecutan una o más tareas comparten el mismo archivo. Para supervisar el demonio MQListener , esta variable debe estar siempre configurada.

Cuando el demonio MQListener se está ejecutando, escribe en el archivo de registro. Por lo tanto, si abre el archivo de registro mientras el demonio MQListener se está ejecutando, debe abrirlo solo en modo de lectura.

MQLSNLWR

Cuando MQLSNLOG especifica un archivo de registro HFS, MQLSNLWR ofrece la posibilidad de limitar el tamaño del archivo de registro HFS. La sintaxis de un comando de exportación MQLSNLWR es:

```
export MQLSNLWR=file-size,file-name
```

Los significados de las variables son:

file-size

El tamaño máximo del archivo de registro de MQListener , en megabytes.

nombre-archivo

El nombre del archivo HFS en el que MQListener guarda una copia del archivo de registro de MQListener .

Cuando se especifica MQLSNLWR y el archivo de registro HFS de MQListener alcanza *el tamaño de archivo*, MQListener guarda una copia del archivo de registro con *el nombre nombre-de-archivo* y reinicializa el archivo de registro HFS.

Importante: La ID con la que se ejecuta MQListener debe tener acceso de escritura al archivo de registro HFS y a la copia del archivo de registro HFS que se especifica por *nombre de archivo*. Si MQListener no puede abrir o escribir en la copia del archivo de registro HFS, MQListener reinicializa el archivo de registro HFS, pero no crea una copia.

Consulte el archivo README para obtener más detalles sobre estas variables.

Tabla de configuración: SYSMQL.LISTENERS

Si utiliza MQListener, debe crear la tabla de configuración MQListener SYSMQL.LISTENERS ejecutando el trabajo de instalación DSNTIJML.

La tabla SYSMQL.LISTENERS contiene una fila para cada configuración que cree al emitir los comandos de configuración MQListener db2mqln1 o db2mqln2.

La siguiente tabla describe cada una de las columnas de la tabla de configuración SYSMQL.LISTENERS.

Tabla 127. Descripción de las columnas de la tabla "SYSMQL.LISTENERS" (Información de la cuenta)

Nombre de columna	Descripción
configurationName	El nombre de la configuración. El nombre de la configuración le permite agrupar varias tareas en la misma configuración. Una sola instancia de MQListener puede ejecutar todas las tareas que se definen dentro de un nombre de configuración.
QueueManager	El nombre del subsistema de IBM MQ que contiene las colas que se van a utilizar.
COLA DE ENTRADA	El nombre de la cola en el WebSphere MQ subsistema que se va a supervisar para los mensajes entrantes. La combinación de la cola de entrada y el gestor de colas son únicos dentro de una configuración
PROCNODE	Actualmente sin usar
PROCESO	El nombre de esquema del procedimiento almacenado que será llamado por MQListener
PROCNAME	El nombre del procedimiento almacenado que será llamado por MQListener
PROYECTO	Actualmente sin usar

Tabla 127. Descripción de las columnas de la tabla "SYSMQL.LISTENERS" (Información de la cuenta) (continuación)

Nombre de columna	Descripción
NUMINSTANCES	El número de instancias duplicadas de una sola tarea que se ejecutarán en esta configuración
WAITMILLIS	El tiempo que espera (en milisegundos) el servidor de correo electrónico (MQListener) después de procesar el mensaje actual antes de buscar el siguiente mensaje
MINQUEUEDEPTH	Actualmente sin usar

Configuración de tareas de MQListener

Como parte de la configuración de MQListener en Db2 for z/OS, debe configurar al menos una tarea de MQListener .

Acerca de esta tarea

Utilice el comando MQListener **db2mq1n1** o **db2mq1n2** para configurar las tareas de MQListener . Emitir el comando desde la línea de comandos de z/OS UNIX System Services en cualquier directorio. De forma alternativa, puede poner el comando en un archivo, conceder permiso de ejecución en el archivo y utilizar la utilidad BPXBATCH para invocar el comando utilizando JCL. Se proporcionan archivos de script de muestra que se encuentran en el directorio */mqlistener-install-path/mqlsn/listener/script* en z/OS UNIX System Services. También se proporciona una muestra de JCL en el miembro DSNTEJML del conjunto de datos *prefix.SDSNSAMP*. Cuando ejecutas comandos MQListener , la información de configuración se almacena en la tabla Db2 SYSMQL.LISTENERS.

Los parámetros de comando son:

-adminQueue

La cola a la que MQListener escucha comandos de administración. Si no se especifica **-adminQueue** , las aplicaciones no reciben ningún comando de administración a través de la cola de mensajes.

-adminQMgr

El nombre del subsistema de IBM MQ , que contiene las colas que se utilizarán para tareas administrativas. Si no se especifica **-adminQMgr** , se utiliza el gestor de colas predeterminado configurado.

-config

Nombre que identifica un grupo de tareas que se ejecutan juntas. Si no se especifica -config, se ejecuta la configuración predeterminada.

-queueManager

El nombre del subsistema de IBM MQ que contiene las colas que se van a utilizar. Si no se especifica **-queueManager** , se utiliza el gestor de colas predeterminado.

-inputQueue

El nombre de la cola en el subsistema de correo electrónico (IBM MQ) que se va a supervisar para los mensajes entrantes. La combinación del valor " -inputQueue " y el valor " **-queueManager** " debe ser única dentro de una configuración.

-numInstances

El número de instancias duplicadas de una sola tarea que se van a ejecutar en una configuración.

-numMessagesCommit

El número de mensajes que se reciben antes de que MQListener emita un COMMIT. El valor predeterminado es 1. Esta opción solo es compatible con **db2mq1n2**.

-procName

El nombre del procedimiento almacenado al que llama MQListener cuando detecta que se recibe un mensaje.

-procSchema

El nombre de esquema del procedimiento almacenado al que llama MQListener cuando detecta que se recibe un mensaje.

-ssID

El subsistema donde se ejecuta el demonio MQListener . La información de configuración se almacena en este subsistema.

-timeRestart

Si un procedimiento almacenado especificado por **-procSchema** y **-procName** falla en el momento de inicio de MQListener , el número de segundos que los subprocessos que se ejecutan con ese procedimiento almacenado se suspenden antes de repetir el proceso de configuración. MQListener continúa el inicio para los hilos que no utilizan ese procedimiento almacenado. Este valor debe ser un número entero en el rango de 0 a 7200. El valor predeterminado es 0.

-restartDB2

Si MQListener se vuelve a conectar automáticamente y reanuda el procesamiento después de que se detenga y reinicie Db2 .

«Y»

MQListener se vuelve a conectar automáticamente y reanuda el procesamiento después de que se detenga y reinicie Db2 .

«N»

MQListener no se vuelve a conectar automáticamente después de detener y reiniciar Db2 . «N» es el valor predeterminado.

A continuación se muestra la sintaxis de los comandos. En la sintaxis del comando, *mqlistener-command* es db2mqln1 o db2mqln2.

- Para agregar una configuración de MQListener , ejecute el comando *mqlistener- add* :

```
mqlistener-command add  
-ssID subsystem-name  
-config configuration-name  
-queueManager queueManager-name  
-inputQueue inputQueue-name  
-procName stored-procedure-name  
-procSchema stored-procedure-schema name  
-numInstances number-of-instances
```

- Para mostrar información sobre la configuración, emita el siguiente comando *mqlistener show* dominio:

```
mqlistener-command show  
-ssID subsystem-name  
-config configuration-name
```

Para mostrar información sobre todas las configuraciones, ejecute el comando *mqlistener - show* :

```
mqlistener-command show  
-ssID subsystem-name  
-config all
```

- Para eliminar las tareas de mensajería, ejecute el comando *m qlistener- remove* :

```
mqlistener-command remove  
-ssID subsystem-name  
-config configuration-name  
-queueManager queueManager-name  
-inputQueue inputQueue-name
```

- Para ejecutar la tarea " MQListener ", ejecute el comando *mqlistener- run* :

```
mqlistener-command run  
-ssID subsystem-name  
-config configuration-name  
-adminQueue adminQueue-name  
-adminQMGr adminQueueManager-name  
-numMessagesCommit number-of-messages-before-commit
```

```
-timeRestart number-of-seconds-to-suspend-before-restart  
-restartDB2 y-or-n
```

- Para cerrar el demonio MQListener , ejecute el comando `mqlistener- admin` :

```
mqlistener-command admin  
-adminQueue adminqueue-name  
-adminQMgr adminqueuemanager-name  
-adminCommand shutdown
```

- Para reiniciar el demonio MQListener , ejecute el siguiente comando:

```
mqlistener-command mqlistener-command admin  
-adminQueue adminqueue-name  
-adminQMgr adminqueuemanager-name  
-adminCommand restart
```

- Para obtener ayuda con el comando y los parámetros válidos, ejecute el comando `mqlistener- help` :

```
mqlistener-command help
```

- Para obtener ayuda sobre un parámetro concreto, ejecute el comando `mqlistener- help` , donde `comando` es un parámetro específico:

```
mqlistener-command help command
```

Restricción:

- Utilice el mismo gestor de colas para la cola de peticiones y la cola de respuestas.
- MQListener no admite mensajes lógicos compuestos por múltiples mensajes físicos. MQListener procesa mensajes físicos de forma independiente.

Creación de un procedimiento almacenado de muestra para usar con MQListener

Puede crear un procedimiento almacenado de muestra, APROC, que puede ser utilizado por MQListener para almacenar un mensaje en una tabla. El procedimiento almacenado devuelve la serie OK si el mensaje se ha insertado satisfactoriamente en la tabla.

Acerca de esta tarea

Este ejemplo asume la siguiente información sobre el entorno:

- MQListener está instalado y configurado para el subsistema DB2A.
- El subsistema de IBM MQ que se define se denomina CSQ1.
- El gestor de colas se está ejecutando y las siguientes colas locales están definidas en el subsistema de DB2A :
 - ADMIN_Q: La cola de administración
 - BACKOUT_Q: La cola de espera
 - DB2MQ_DEFAULT_Q : La cola de entrada, que tiene una cola de retroceso con un umbral de 3
 - REPLY_Q: La cola de salida o la cola de respuesta
 - DEADLETTER_Q: La cola de cartas muertas
- El usuario que ejecuta el demonio MQListener tiene el privilegio EXECUTE en el plan DB2MQLSN.
- MQListener pasa el encabezado del mensaje (estructura MQMD) a la interfaz de procedimiento almacenado para MQListener.

Procedimiento

Los siguientes pasos crean objetos d Db2 , que puede utilizar con aplicaciones d MQListener :

1. Cree una tabla utilizando SPUFI, DSNTEP2 o el Db2 command line processor en el subsistema donde desee ejecutar MQListener:

```
CREATE TABLE PROCTABLE (MSG VARCHAR(25) CHECK (MSG NOT LIKE 'FAIL%'));
```

La tabla contiene una restricción de verificación para que los mensajes que comiencen con los caracteres FAIL no puedan insertarse en la tabla. La restricción de verificación se utiliza para demostrar el comportamiento de MQListener cuando falla el procedimiento almacenado.

2. Cree el siguiente procedimiento SQL y defínalo en el mismo subsistema de Db2 .

```
CREATE PROCEDURE TEST.APROC (
    IN PIN VARCHAR(25),
    OUT POUT VARCHAR(2),
    INOUT PMSGHEADER VARBINARY(500))
VERSION V1
LANGUAGE SQL
PROCEDURE1: BEGIN
    DECLARE REPLYQ VARBINARY(48);
    DECLARE REPLYQM VARBINARY(48);
    SET REPLYQ = VARBINARY(CONCAT('NEWREPLYQUEUE',X'00'));
    SET REPLYQM = VARBINARY(CONCAT('CSQ1',X'00'));
    SET PMSGHEADER = INSERT(PMSGHEADER,101,LENGTH(REPLYQ),REPLYQ);
    SET PMSGHEADER = INSERT(PMSGHEADER,149,LENGTH(REPLYQM),REPLYQM);
    INSERT INTO SYSADM.PROCTABLE VALUES (PIN);
    SET POUT = 'OK';
END PROCEDURE1
```

3. Añada la siguiente configuración, llamada ACFG, a la tabla de configuración ejecutando este comando:

```
db2mqln2 add
-ssID DB2A
-config ACFG
-queueManager CSQ1
-inputQueue DB2MQ_DEFAULT_Q
-procName APROC
-procSchema TEST
```

4. Ejecutar el demonio MQListener para la confirmación en dos fases para la configuración ACFG. Para ejecutar MQListener con todas las tareas especificadas en la configuración, ejecute el siguiente comando:

```
db2mqln2 run
-ssID DB2A
-config ACFG
-adminQueue ADMIN_Q
-adminQMgr MQND
```

5. Enviar una solicitud a la cola de entrada, "DB2MQ_DEFAULT_Q", con el mensaje 'otro mensaje de muestra'.
6. Consulte la tabla PROCTABLE para verificar que se insertó el mensaje de muestra:

```
SELECT * FROM PROCTABLE;
```

7. Mostrar el número de mensajes que quedan en la cola de entrada, para verificar que el mensaje ha sido eliminado. Para hacerlo, ejecute el siguiente comando desde una consola de z/OS :

```
/-CSQ1 display queue('DB2MQ_DEFAULT_Q') curdepth
```

8. Mira el nombre de la e ReplytoQ a que especificaste, para verificar que el procedimiento almacenado genera la cadena 'OK'.

MQListener Error durante el procesamiento

MQListener lee desde colas de mensajes IBM MQ y llama a procedimientos almacenados Db2 con esos mensajes. Si se produce algún error durante este proceso y el mensaje debe enviarse a la cola de mensajes no entregados, MQListener devuelve un código de motivo a la cola de mensajes no entregados.

MQListener , en concreto, lleva a cabo las siguientes acciones:

- antepone al mensaje una estructura de encabezado de carta muerta de mensajes (MQ, MQDLH)
- establece el campo de motivo en la estructura MQDLH con el código de motivo adecuado
- envía el mensaje a la cola de correo no distribuido

La siguiente tabla describe los códigos de motivo que devuelve el demonio MQListener .

Tabla 128. Códigos de motivo que devuelve MQListener	
Código de razón	Explicación
900	<p>La llamada a un procedimiento almacenado se realizó correctamente, pero se produjo un error durante el proceso de confirmación de la transacción (Db2) y se cumplió alguna de las siguientes condiciones:</p> <ul style="list-style-type: none"> • No se solicitó ningún informe de excepción.¹ • Se solicitó un informe de excepción, pero no se pudo entregar. <p>Este código de motivo solo se aplica a entornos de confirmación de una fase.</p>
901	<p>La llamada al procedimiento almacenado especificado ha fallado y la disposición del mensaje de error (MQ) es que se genere un informe de excepción y que el mensaje original se envíe a la cola de mensajes no entregados.</p>
902	<p>Se han dado todas las condiciones siguientes:</p> <ul style="list-style-type: none"> • La disposición del mensaje de error (MQ) es que no se genere un informe de excepción.¹ • El procedimiento almacenado se llamó sin éxito el número de veces que se especifica como el umbral de retroceso. • El nombre de la cola de devolución es el mismo que el de la cola de mensajes no entregados. <p>Este código de motivo solo se aplica a entornos de confirmación en dos fases.</p>
MQRC_TRUNCATED_MSG FAILED	<p>El tamaño del mensaje de error " MQ " es mayor que el parámetro de entrada del procedimiento almacenado que se va a invocar. En entornos de envío en una sola fase, este mensaje de gran tamaño se envía a la cola de correo no distribuido. En entornos de envío en dos fases, este mensaje de gran tamaño se envía a la cola de mensajes no entregados solo cuando el mensaje no se puede entregar a la cola de mensajes devueltos.</p>

Nota:

1. Para especificar que la aplicación receptora genere informes de excepción si se producen errores, establezca el campo de informe en la estructura MQMD que se utilizó al enviar el mensaje a uno de los siguientes valores:
 - MQRO_EXCEPTION
 - MQRO_EXCEPTION_WITH_DATA
 - MQRO_EXCEPTION_WITH_FULL_DATA

Referencia relacionada

[Inicio de IBM MQ](#)

MQListener Ejemplos

La aplicación recibe un mensaje, lo inserta en una tabla y genera un mensaje de respuesta simple.

Para simular una anomalía de proceso, la aplicación incluye una restricción de comprobación en la tabla que contiene el mensaje. La restricción impide que se inserte en la tabla cualquier cadena que comience con los caracteres «fail». Si intenta insertar un mensaje que infrinja la restricción de verificación, la aplicación de ejemplo devuelve un mensaje de error y vuelve a poner el mensaje fallido en la cola de retroceso.

En este ejemplo, se hacen las siguientes suposiciones:

- MQListener está instalado y configurado para el subsistema DB7A.

- MQND es el nombre de un subsistema de IBM MQ . El gestor de colas se está ejecutando y las siguientes colas locales están definidas en el subsistema de DB7A :

ADMIN_Q: Cola de administración
 BACKOUT_Q: Cola de espera
 IN_Q: Cola de entrada que tiene una cola de retroceso con umbral = 3
 REPLY_Q: Cola de salida o cola de respuesta
 DEADLETTER_Q: Cola de correos no entregados

- La persona que está ejecutando el demonio MQListener tiene permiso de ejecución en el plan DB2MQLSN.

Antes de ejecutar el demonio MQListener , agregue la siguiente configuración, llamada ACFG, a la tabla de configuración ejecutando el siguiente comando:

```
db2mqln2 add
-ssID DB7A
-config ACFG
-queueManager MQND
-inputQueue IN_Q
-procName APROC
-procSchema TEST
```

Ejecute el demonio MQListener para la confirmación en dos fases de la configuración ACFG mediante el siguiente comando:

```
db2mqln2 run
-ssID DB7A
-config ACFG
-adminQueue ADMIN_Q
-adminQMgr MQND
-numMessagesCommit 1
-timeRestart 60
```

Los siguientes ejemplos muestran cómo utilizar MQListener para enviar un mensaje sencillo y, a continuación, inspeccionar los resultados del mensaje en el gestor de colas de IBM MQ y en la base de datos. Los ejemplos incluyen consultas para determinar si la cola de entrada contiene un mensaje o para determinar si un registro es colocado en la tabla por el procedimiento almacenado.

MQListener ejemplo 1: Ejecutar una aplicación sencilla:

1. Comience con una tabla de base de datos limpia emitiendo la siguiente instrucción SQL:

```
delete from PROCTABLE
```

2. Enviar un datagrama a la cola de entrada, 'IN_Q', con el mensaje como 'mensaje de muestra'. Consulte WebSphere MQ CSQ4BCK1 de muestra para enviar un mensaje a la cola. Especifique la opción de MsgType para 'Message Descriptor' como 'MQMT_DATAGRAM'.
3. Consulte la tabla utilizando la siguiente instrucción para verificar que el mensaje de muestra se ha insertado:

```
select * from PROCTABLE
```

4. Mostrar el número de mensajes que quedan en la cola de entrada para verificar que el mensaje ha sido eliminado. Emitir el siguiente comando desde una z/OS consola:

```
/-MQND display queue('In_Q') curdepth
```

MQListener ejemplo 2: Envío de solicitudes a la cola de entrada e inspección de la respuesta:

1. Comience con una tabla de base de datos limpia emitiendo la siguiente instrucción SQL:

```
delete from PROCTABLE
```

2. Enviar una solicitud a la cola de entrada, 'IN_Q', con el mensaje como 'otro mensaje de muestra'. Consulte IBM MQ sample CSQ4BCK1 para enviar un mensaje a la cola. Especifique la opción "

MsgType " para "Message Descriptor" como "MQMT_REQUEST" y el nombre de la cola para la opción "ReplytoQ".

3. Consulte la tabla utilizando la siguiente instrucción para verificar que el mensaje de muestra se ha insertado:

```
select * from PROCTABLE
```

4. Mostrar el número de mensajes que quedan en la cola de entrada para verificar que el mensaje ha sido eliminado. Emitir el siguiente comando desde una z/OS consola:

```
/-MQND display queue('In_Q') curdepth
```

5. Busque el nombre de ReplytoQ que especificó cuando envió el mensaje de solicitud para la respuesta utilizando el programa de muestra IBM MQ CSQ4BCJ1. Verifique que el procedimiento almacenado genere la cadena «OK».

MQListener ejemplo 3: Prueba de una operación de inserción fallida: Si envía un mensaje que comienza con la cadena «fail», se infringe la restricción en la definición de la tabla y el procedimiento almacenado falla.

1. Comience con una tabla de base de datos limpia emitiendo la siguiente instrucción SQL:

```
delete from PROCTABLE
```

2. Enviar una solicitud a la cola de entrada, 'IN_Q', con el mensaje como 'mensaje de muestra erróneo'. Consulte IBM MQ sample CSQ4BCK1 para enviar un mensaje a la cola. Especifique la opción "MsgType" para "Message Descriptor" como "MQMT_REQUEST" y el nombre de la cola para la opción "ReplytoQ".

3. Consulte la tabla utilizando la siguiente instrucción para verificar que el mensaje de muestra no está insertado:

```
select * from PROCTABLE
```

4. Mostrar el número de mensajes que quedan en la cola de entrada para verificar que el mensaje ha sido eliminado. Emitir el siguiente comando desde una z/OS consola:

```
/-MQND display queue('In_Q') curdepth
```

5. Mira la cola de Backout y encuentra el mensaje original utilizando el WebSphere MQ programa de ejemplo CSQ4BCJ1.

Nota : En este ejemplo, si se envía un mensaje de solicitud con opciones añadidas para el «informe de excepciones» (la opción Informe se especifica para «Descriptor de mensaje»), se envía un informe de excepciones a la cola de respuesta y el mensaje original se envía a la cola de mensajes descartados.

Capítulo 7. Db2 como consumidor y proveedor de servicios web

Los servicios web son un conjunto de recursos y componentes que las aplicaciones pueden utilizar a través de HTTP. Puede utilizar Db2 como proveedor de servicios web y como consumidor de servicios web.

Db2 como consumidor de servicios web

Db2 puede actuar como cliente de servicios web, lo que le permite ser consumidor de servicios web en sus aplicaciones de Db2 .

Servicios web SOAP El Protocolo simple de acceso a objetos (SOAP) es un protocolo XML que consta de las siguientes características:

- Un sobre que define una infraestructura para describir el contenido de un mensaje y cómo procesar el mensaje
- Un conjunto de reglas de codificación para expresar instancias de tipos de datos definidos a nivel de aplicación
- Un convenio de representación de peticiones y respuestas SOAP

Db2 proporciona un conjunto de funciones SOAP que se instala y configura al instalar o migrar Db2.

Servicios web REST El protocolo Representational State Transfer (REST) proporciona acceso a contenido basado en web directamente desde sentencias SQL a través de solicitudes de tipo "HTTP ". Db2 permite instalar un conjunto de funciones REST básicas de muestra definidas por el usuario. Estas funciones proporcionan acceso a contenido basado en la web a través de los métodos GET, POST, PUT y DELETE de la especificación HTTP.

Db2 como proveedor de servicios web

Puede habilitar sus datos y aplicaciones de Db2 como servicios web a través del Web Services Object Runtime Framework (WORF). Puede definir un servicio web en Db2 utilizando una extensión de definición de acceso a documentos (DADX). En el archivo DADX, puede definir servicios web basados en sentencias SQL y procedimientos almacenados. En función de sus definiciones en el archivo DADX, WORF realiza las siguientes acciones:

- Gestiona la conexión con Db2 y la ejecución del SQL y la llamada al procedimiento almacenado
- Convierte el resultado en un servicio web
- Gestiona la generación de cualquier información de Lenguaje de Definición de Servicios Web (WSDL) y UDDI (Descripción, Descubrimiento e Integración Universales) que necesite la aplicación cliente

Conceptos relacionados

[Funciones REST definidas por el usuario de ejemplo \(Db2 Installation and Migration\)](#)

En desuso: Funciones definidas por el usuario SOAPHTTPV y SOAPHTTPC

Db2 proporciona funciones definidas por el usuario que le permiten trabajar con SOAP y consumir servicios web en instrucciones SQL. Las funciones definidas por el usuario son dos variedades de SOAPHTTPV para datos VARCHAR y dos variedades de SOAPHTTPC para datos CLOB.

Restricción: Las funciones definidas por el usuario SOAPHTTPV y SOAPHTTPC han quedado obsoletas. Utilice en su lugar las funciones definidas por el usuario SOAPHTTPNV y SOAPHTTPNC.

Las funciones definidas por el usuario realizan las siguientes acciones:

1. Redactar una solicitud SOAP
2. Publicar la solicitud en el punto final del servicio
3. Recibir la respuesta SOAP
4. Devuelve el contenido del cuerpo SOAP

Cuando un consumidor recibe el resultado de una solicitud de servicios web, se elimina el sobre SOAP y se devuelve el documento XML. Un programa de aplicación puede procesar los datos de resultados y realizar diversas operaciones, como insertar o actualizar una tabla con los datos de resultados.

SOAPHTTPV y SOAPHTTPC son funciones definidas por el usuario que permiten a Db2 trabajar con SOAP y consumir servicios web en sentencias SQL. Estas funciones son funciones sobrecargadas que se utilizan para datos VARCHAR o CLOB de diferentes tamaños, dependiendo del cuerpo SOAP. Los servicios web pueden invocarse de una de cuatro maneras, dependiendo del tamaño de los datos de entrada y de los datos de resultado. SOAPHTTPV devuelve datos VARCHAR(32672) y SOAPHTTPC devuelve datos CLOB(1M). Ambas funciones aceptan VARCHAR(32672) o CLOB(1M) como cuerpo de entrada.

Ejemplo : El siguiente ejemplo muestra un encabezado de publicación de tipo " HTTP " que publica un sobre de solicitud SOAP a un host. El cuerpo del sobre de SOAP muestra una solicitud de temperatura para Barcelona.

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host: services.xmethods.net
Connection: Keep-Alive User-Agent: DB2SOAP/1.0
Content-Type: text/xml; charset="UTF-8"
SOAPAction: ""
Content-Length: 410

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAP-ENV:Body>
        <ns:getTemp xmlns:ns="urn:xmethods-Temperature">
            <city>Barcelona</city>
        </ns:getTemp>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Ejemplo : El siguiente ejemplo es el resultado del ejemplo anterior. Este ejemplo muestra el encabezado de respuesta SOAP (HTTP) con el sobre de respuesta SOAP. El resultado muestra que la temperatura es de 85 grados Fahrenheit en Barcelona.

```
HTTP/1.1 200 OK
Date: Wed, 31 Jul 2002 22:06:41 GMT
Server: Enhydra-MultiServer/3.5.2
Status: 200
Content-Type: text/xml; charset=utf-8
Servlet-Engine: Lutris Enhydra Application Server/3.5.2
  (JSP 1.1; Servlet 2.2; Java 1.3.1_04;
   Linux 2.4.7-10smp i386; java.vendor=Sun Microsystems Inc.)
Content-Length: 467
Set-Cookie:JSESSIONID=JLEcR34rBc2GTIkn-0F51ZDk;Path=/soap
X-Cache: MISS from www.xmethods.net
Keep-Alive: timeout=15, max=10
Connection: Keep-Alive

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAP-ENV:Body>
        <ns1:getTempResponse xmlns:ns1="urn:xmethods-Temperature">
            <SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
                <return xsi:type="xsd:float">85</return>
            </ns1:getTempResponse>
        </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
```

Ejemplo : El siguiente ejemplo muestra cómo insertar el resultado de un servicio web en una tabla

```
INSERT INTO MYTABLE(XMLCOL) VALUES (DB2XML.SOAPHTTPC(
  'http://www.myserver.com/services/db2sample/list.dadx/SOAP',
  'http://tempuri.org/db2sample/list.dadx'
  '<listDepartments xmlns="http://tempuri.org/db2sample/listdadx">
    <deptno>A00</deptno>
  </ListDepartments>'))
```

Funciones definidas por el usuario SOAPHTTPNV y SOAPHTTPNC

Db2 proporciona las funciones definidas por el usuario SOAPHTTPNV y SOAPHTTPNC, las cuales permiten trabajar con SOAP y consumir servicios web en sentencias de SQL. Las funciones definidas por el usuario son dos variedades de SOAPHTTPNV para datos VARCHAR y dos variedades de SOAPHTTPNC para datos CLOB.

Las funciones definidas por el usuario realizan las siguientes acciones:

1. Publicar la solicitud SOAP de entrada en el punto final del servicio
2. Recibir y devolver la respuesta SOAP

SOAPHTTPNV y SOAPHTTPNC le permiten especificar un mensaje SOAP completo como entrada y devolver mensajes SOAP completos desde el servicio web especificado como una representación CLOB o VARCHAR de los datos XML devueltos.. SOAPHTTPNV devuelve datos VARCHAR(32672) y SOAPHTTPNC devuelve datos CLOB(1M). Ambas funciones aceptan VARCHAR(32672) o CLOB(1M) como cuerpo de entrada.

Las funciones definidas por el usuario SOAPHTTPNV y SOAPHTTPNC pueden admitir SOAP 1.1 o SOAP 1.2. Consulte con el administrador del sistema para determinar qué niveles de SOAP son compatibles con las funciones definidas por el usuario en su entorno.

Ejemplo

El siguiente ejemplo muestra cómo insertar el resultado completo de un servicio web en una tabla utilizando SOAPHTTPNC.

```
INSERT INTO EMPLOYEE(XMLCOL)
  VALUES (DB2XML.SOAPHTTPNC(
    'http://www.myserver.com/services/db2sample/list.dadx/SOAP',
    'http://tempuri.org/db2sample/list.dadx',
    '<?xml version="1.0" encoding="UTF-8" ?>' ||
    '<SOAP-ENV:Envelope' ||
    'xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" ' ||
    'xmlns:xsd="http://www.w3.org/2001/XMLSchema" ' ||
    'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">' ||
    '<SOAP-ENV:Body' ||
    '<listDepartments xmlns="http://tempuri.org/db2sample/list.dadx">
      <deptNo>A00</deptNo>
    </listDepartments>' ||
    '</SOAP-ENV:Body>' ||
    '</SOAP-ENV:Envelope>'))
```

Tareas relacionadas

[Pasos adicionales para habilitar funciones definidas por el usuario de servicios web \(Db2 Installation and Migration\)](#)

SQLSTATEs para Db2 como consumidor de servicios web

Db2 devuelve valores SQLSTATE para condiciones de error relacionadas con el uso de Db2 como consumidor de servicios web.

Las siguientes tablas muestran los posibles valores de SQLSTATE.

Tabla 129. Valores SQLSTATE para funciones definidas por el usuario SOAPHTTPV y SOAPHTTPC

SQLSTATE	Descripción
38301	Se ha pasado un valor NULL inesperado como entrada a la función.
38302	La función no pudo asignar espacio.
38304	Se especificó un protocolo desconocido en el punto final URL.
38305	Se especificó un URL no válido en el punto final URL.
38306	Se ha producido un error al intentar crear un socket TCP/IP.
38307	Se ha producido un error al intentar vincular un socket TCP/IP.
38308	La función no pudo resolver el nombre de host especificado.
38309	Se ha producido un error al intentar conectarse al servidor especificado.
38310	Se ha producido un error al intentar recuperar información del protocolo.
38311	Se ha producido un error al intentar establecer las opciones de socket.
38312	La función recibió datos inesperados devueltos para el servicio web.
38313	El servicio web no devolvió datos del tipo de contenido adecuado.
38314	Se ha producido un error al inicializar el analizador XML.
38315	Se ha producido un error al crear el analizador XML.
38316	Se ha producido un error al establecer un controlador para el analizador XML.
38317	El analizador XML encontró un error al analizar los datos resultantes.
38318	El analizador XML no pudo convertir los datos resultantes a la página de códigos de la base de datos.
38319	La función no pudo asignar memoria al crear un socket TCP/IP.
38320	Se ha producido un error al intentar enviar la solicitud al servidor especificado.
38321	La función no pudo enviar la solicitud completa al servidor especificado.
38322	Se ha producido un error al intentar leer los datos de resultados del servidor especificado.
38323	Se ha producido un error mientras se esperaba a que el servidor especificado devolviera los datos.
38324	La función encontró un error interno al intentar formatear el mensaje de entrada.
38325	La función encontró un error interno al intentar agregar información de espacio de nombres al mensaje de entrada.
38327	El analizador XML no pudo eliminar el sobre SOAP del mensaje de resultado.
38328	Se ha producido un error al procesar una conexión SSL.

Tabla 130. Valores SQLSTATE para funciones definidas por el usuario SOAPHTTPNV y SOAPHTTPNC

SQLSTATE	Descripción
38350	Se especificó un valor NULL inesperado para el punto final, la acción o la entrada SOAP.
38351	Un error dinámico de asignación de memoria.

Tabla 130. Valores SQLSTATE para funciones definidas por el usuario SOAPHTTPNV y SOAPHTTPNC (continuación)

SQLSTATE	Descripción
38352	Un protocolo de transporte desconocido o no compatible.
38353	Se ha especificado una dirección de correo electrónico no válida (URL).
38354	Se ha producido un error al resolver el nombre de host.
38355	Una excepción de memoria para el socket.
38356	Se ha producido un error durante la conexión del socket.
38357	Se ha producido un error al configurar las opciones de socket.
38358	Se ha producido un error durante el control de entrada/salida (ioctl) para verificar la habilitación de HTTPS.
38359	Se ha producido un error al leer desde el socket.
38360	Se ha producido un error debido a un tiempo de espera de socket agotado.
38361	No hay respuesta del host especificado.
38362	Se ha producido un error debido a un retorno o tipo de contenido inesperado de HTTP
38363	La pila TCP/IP no estaba habilitada para HTTPS.

Tareas relacionadas

Pasos adicionales para habilitar funciones definidas por el usuario de servicios web (Db2 Installation and Migration)

Capítulo 8. Niveles de compatibilidad de aplicaciones en Db2 12

El nivel de compatibilidad de sus aplicaciones controla la adopción y el uso de nuevas capacidades y mejoras, y a veces reduce el impacto de los cambios incompatibles. La ventaja es que puede completar el proceso de migración de Db2 12 sin necesidad de actualizar sus aplicaciones inmediatamente.

Después de activar nivel de función 500 o superior, puede continuar ejecutando aplicaciones con las características y el comportamiento de versiones anteriores o niveles de función específicos de Db2 12.

Puede cambiar el nivel de compatibilidad de aplicación para cada aplicación cuando esté preparado para que se ejecute con las características y el comportamiento de una versión de Db2 o nivel de función superior. El nivel de compatibilidad de la aplicación se aplica a la mayoría de las sentencias SQL, incluidas las sentencias de definición de datos (como las sentencias CREATE y ALTER) y las sentencias de control de datos (como las sentencias GRANT y REVOKE).

La compatibilidad de la aplicación de un paquete se establece inicialmente cuando vinculas un paquete, en función de los siguientes valores:

1. El valor de la opción de enlace APPLCOMPAT, si se especifica.
2. Si se omite la opción bind, el parámetro del subsistema APPLCOMPAT.

Para las sentencias SQL estáticas, la columna APPLCOMPAT de la tabla de catálogo de SYSIBM.SYSPACKAGE almacena la configuración de compatibilidad de la aplicación. Esta configuración cambia por las siguientes razones:

- Usted emite un comando REBIND para el paquete y especifica un valor diferente para la opción APPLCOMPAT. Si omite esta opción, se utilizará el valor anterior del paquete. Si no hay ningún valor anterior disponible (como en el caso de los paquetes vinculados por última vez antes de la introducción de la compatibilidad de aplicaciones), se utiliza el valor del parámetro del subsistema APPLCOMPAT.
- Se produce un enlace automático del paquete. La compatibilidad de la aplicación se establece en el valor anterior. Si no hay ningún valor anterior disponible, se utiliza el valor del parámetro del subsistema APPLCOMPAT.

Para las sentencias SQL dinámicas, el registro especial CURRENT APPLICATION COMPATIBILITY almacena la configuración de compatibilidad de la aplicación. Esta configuración cambia por las siguientes razones:

- El registro especial se inicializa con la compatibilidad de la aplicación del paquete, como se ha descrito anteriormente.
- Durante la ejecución del paquete, las sentencias SET CURRENT APPLICATION COMPATIBILITY pueden cambiar el registro especial. El valor debe ser equivalente a la opción de enlace APPLCOMPAT para el paquete o inferior, si el valor es de V12R1M500 o superior.

Para nuevas instalaciones, el valor predeterminado del parámetro del subsistema APPLCOMPAT es V12R1M500. Sin embargo, puede especificar un valor más alto. Para entornos migrados, el valor predeterminado es el valor del miembro de entrada de migración.

Consejo: Cuando migre a Db2 12 o active cualquier nivel de función superior, cambie el valor del parámetro del subsistema APPLCOMPAT solo después de que todas las aplicaciones puedan utilizar las capacidades SQL de Db2 12 o el nivel de función superior. Para obtener detalles, consulte “[Habilitar la compatibilidad de aplicaciones predeterminada con nivel de función 500 o superior](#)” en la página 889.

Niveles de compatibilidad de aplicaciones compatibles en Db2 12

Db2 12 admite los siguientes niveles de compatibilidad de aplicaciones en la mayoría de los contextos.

Consejo: Para obtener los mejores resultados, configure su entorno de desarrollo para que utilice el nivel de compatibilidad de aplicación más bajo en el que se ejecutará la aplicación en el entorno de

producción. Para SQL dinámico, recuerde tener en cuenta los niveles de compatibilidad de la aplicación de los paquetes cliente y NULLID. Si desarrolla y prueba aplicaciones en un nivel de compatibilidad de aplicaciones más alto e intenta ejecutarlas en un nivel más bajo en producción, es probable que encuentre el código SQL -4743 y otros errores cuando implemente las aplicaciones en producción.

VvvRrMmmm

Compatibilidad con el comportamiento del nivel de función Db2 identificado. Por ejemplo, V12R1M510 especifica la compatibilidad con el nivel de función de Db2 12 más alto disponible. Debe activarse el nivel de función equivalente o superior.

Para conocer las nuevas capacidades que están disponibles en cada nivel de compatibilidad de la aplicación, consulte:

- [Cambios en SQL en Db2 13 niveles de compatibilidad de aplicaciones](#)
- [Cambios en SQL en Db2 12 niveles de compatibilidad de aplicaciones](#)

Consejo: Es posible que se requieran pasos adicionales de preparación del programa para aumentar el nivel de compatibilidad de las aplicaciones que utilizan clientes o controladores de servidores de datos para acceder a Db2 for z/OS. Para obtener más información, consulte [“Establecimiento de niveles de compatibilidad de aplicaciones para clientes y controladores de servidores de datos” en la página 870](#).

V12R1

Compatibilidad con el comportamiento del nivel de función 500 Db2 12. Este valor tiene el mismo resultado que especificar V12R1M500.

V11R1

Compatibilidad con el comportamiento de la modalidad de nueva función de Db2 11. Después de la migración a Db2 12, este valor tiene el mismo resultado que especificar V12R1M100. Para más información, consulte [“Nivel de compatibilidad de aplicaciones V11R1” en la página 878](#)

V10R1

Compatibilidad con el comportamiento de la modalidad de nueva función de DB2 10. Para obtener más información, consulte [“Nivel de compatibilidad de aplicaciones V10R1” en la página 884](#).

Ejemplo: compatibilidad de la aplicación V10R1

El siguiente ejemplo muestra los resultados de utilizar una capacidad que se introduce en el nivel de compatibilidad de la aplicación V11R1, con el nivel de compatibilidad de la aplicación establecido en V10R1. Supongamos que el valor del parámetro del subsistema APPLCOMPAT es V10R1. La instrucción CREATE PROCEDURE de ejemplo no especifica la palabra clave APPLCOMPAT. En este ejemplo, la instrucción CREATE TYPE se ejecuta correctamente, pero la instrucción CREATE PROCEDURE da como resultado el código SQL -4743.

```
CREATE TYPE PHONENUMBERS AS VARCHAR(12) ARRAY ??(1000000???)
```

```
DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
```

```
CREATE PROCEDURE FIND_CUSTOMERS(
  IN NUMBERS_IN KRAMSC01.PHONENUMBERS,
  IN AREA_CODE CHAR(3),
  OUT NUMBERS_OUT KRAMSC01.PHONENUMBERS)
BEGIN
  SET NUMBERS_OUT =
    (SELECT ARRAY_AGG(T.NUM)
     FROM UNNEST(NUMBERS_IN) AS T(NUM)
     WHERE SUBSTR(T.NUM, 1, 3) = AREA_CODE);
END
```

```
DSNT408I SQLCODE = -4743, ERROR: ATTEMPT TO USE A FUNCTION WHEN THE
APPLICATION COMPATIBILITY SETTING IS SET FOR A PREVIOUS LEVEL
```

El valor de la opción APPLCOMPAT bind para la instrucción CREATE PROCEDURE se establece entonces en V11R1 o superior y el resultado de la instrucción es entonces satisfactorio.

```
CREATE PROCEDURE FIND_CUSTOMERS(
    IN NUMBERS_IN KRAMSC01.PHONENUMBERS,
    IN AREA_CODE CHAR(3),
    OUT NUMBERS_OUT KRAMSC01.PHONENUMBERS)
APPLCOMPAT V11R1
BEGIN
    SET NUMBERS_OUT =
        (SELECT ARRAY_AGG(T.NUM)
        FROM UNNEST(NUMBERS_IN) AS T(NUM)
        WHERE SUBSTR(T.NUM, 1, 3) = AREA_CODE);
END
```

```
DSNT400I SQLCODE = 000, SUCCESSFUL EXECUTION
```

Conceptos relacionados

[Niveles de función y niveles relacionados en Db2 12 \(Db2 for z/OS ¿Qué hay de nuevo?\)](#)

Tareas relacionadas

[Controlar el nivel de compatibilidad de la aplicación Db2 \(Db2 for z/OS ¿Qué hay de nuevo?\)](#)

Referencia relacionada

[APPLCOMPAT opción bind \(comandos Db2 \)](#)

[COMPATIBILIDAD CON LA APLICACIÓN ACTUAL registro especial \(Db2 SQL\)](#)

[APPL COMPAT LEVEL \(parámetro del subsistemaAPPLCOMPAT \) \(Db2 Instalación y migración\)](#)

[Tabla de catálogo SYSPACKAGE \(Db2 SQL\)](#)

[SET CURRENT APPLICATION COMPATIBILITY declaración \(Db2 SQL\)](#)

[-ACTIVATE comando \(Db2\) \(Db2 Comandos\)](#)

Niveles de compatibilidad de aplicaciones V12R1Mnnn

En Db2 12, puede utilizar el nivel de compatibilidad de aplicaciones para controlar la adopción de nuevas funciones de SQL y mejoras de determinados niveles de función.

Puede utilizar el nivel de *compatibilidad de aplicación* de aplicaciones, y objetos como, por ejemplo, rutinas o desencadenantes, para controlar la adopción y el uso de las funciones de SQL nuevas y modificadas que se introducen en los niveles de función. Generalmente, las aplicaciones y rutinas o desencadenantes no pueden utilizar prestaciones SQL nuevas o modificadas a menos que el nivel de compatibilidad de aplicación efectivo sea equivalente o superior al nivel de función que ha introducido los cambios. El nivel de compatibilidad de la aplicación se aplica a la mayoría de las sentencias SQL, incluidas las sentencias de definición de datos (como las sentencias CREATE y ALTER) y las sentencias de control de datos (como las sentencias GRANT y REVOKE).

Debe activarse el nivel de función correspondiente o superior cuando enlaza paquetes en un nivel de compatibilidad de aplicaciones. Sin embargo, si activa un nivel de función inferior (o un nivel de función *), las aplicaciones pueden seguir ejecutándose con el nivel de compatibilidad de aplicaciones superior. Para evitar el uso continuado de las prestaciones de SQL introducidas en el nivel de función superior, también debe modificar la aplicación y cambiar el nivel de compatibilidad de aplicación efectivo al nivel inferior.

Consejo: Es posible que se requieran pasos adicionales de preparación del programa para aumentar el nivel de compatibilidad de las aplicaciones que utilizan clientes o controladores de servidores de datos para acceder a Db2 for z/OS. Para obtener más información, consulte “[Establecimiento de niveles de compatibilidad de aplicaciones para clientes y controladores de servidores de datos](#)” en la página 870.

Consejo: No aumente el nivel de compatibilidad de aplicación predeterminado del subsistema Db2 inmediatamente después de migrar o activar un nuevo nivel de función. En su lugar, espere hasta que se haya verificado que las aplicaciones funcionan correctamente en el nivel de función superior y se hayan resuelto las incompatibilidades. Para obtener detalles, consulte “[Habilitar la compatibilidad de aplicaciones predeterminada con nivel de función 500 o superior](#)” en la página 889.

Los niveles de compatibilidad de aplicación se especifican en los mandatos y la salida de mensajes mediante series de nueve caracteres que corresponden a la versión, release y valor de modificación de Db2 del nivel de función correspondiente. Consulte los detalles de activación para cada nivel de función para obtener un resumen de las nuevas características controladas por el nivel de compatibilidad de aplicación correspondiente.

Por ejemplo, V12R1M510 especifica la compatibilidad con el nivel de función de Db2 12 más alto disponible. Debe activarse el nivel de función equivalente o superior.

Referencia relacionada

[APPLCOMPAT opción bind \(comandos Db2 \)](#)

[APPL COMPAT LEVEL \(parámetro del subsistemaAPPLCOMPAT \) \(Db2 Instalación y migración\)](#)

[COMPATIBILIDAD CON LA APLICACIÓN ACTUAL registro especial \(Db2 SQL\)](#)

Establecimiento de niveles de compatibilidad de aplicaciones para clientes y controladores de servidores de datos

IBM los clientes y controladores de servidores de datos que utilizan capacidades de Db2 for z/OS con un requisito de nivel de función superior a V12R1M500 requieren pasos adicionales de preparación del programa.

Antes de empezar

Tome estas medidas con los clientes que se conecten a su sistema Db2 for z/OS :

- Determine si necesita actualizar los clientes o controladores del servidor de datos para que sean compatibles con la aplicación de V12R1M501 :
 - Si sus aplicaciones incluyen una función que requiere un nivel mínimo de compatibilidad de aplicación de V12R1M501, debe actualizar a Db2 Connect Versión 11.1 Modification 2 Fix Pack 2 o posterior.
 - Si sus aplicaciones incluyen una función que requiere un nivel mínimo de compatibilidad de la aplicación de V12R1M500 o anterior, puede utilizar cualquier versión de Db2 Connect .

Los niveles mínimos de cliente o controlador de servidor de datos para la explotación de la compatibilidad de aplicaciones de V12R1M501 o posterior son:

- IBM Data Server Driver for JDBC and SQLJ: Versiones 3.72 y 4.22, o posteriores. Para obtener información sobre las versiones de controlador que se entregan con cada versión de Db2 Connect , consulte [IBM Data Server Driver for JDBC and SQLJ versiones y Db2 o Db2 Niveles de conexión \(Db2 Application Programming for Java\)](#).
- Otros IBM clientes y controladores de servidores de datos: Db2 for Linux, UNIX, and Windows, versión 11.1, Modification 2, Fix Pack 2 o posterior.
- Ejecute la utilidad db2connectactivate para activar el archivo de certificado de licencia de la versión 11.1 para Db2 Connect Unlimited Edition en Db2 for z/OS. Especifique las opciones para vincular los paquetes de controlador y cliente en la colección NULLID, con APPLCOMPAT V12R1M500. Por ejemplo:

```
db2connectactivate.sh -host sys1.sv1.ibm.com -port 5021 -database STLEC1 -user dbadm  
-password dbadmpass -bindoptions "APPLCOMPAT V12R1M500" -collection NULLID
```

Para obtener más información, consulte:

[Activación de la clave de licencia de Db2 Connect Unlimited Edition \(IBM Z\)](#)

[db2connectactivate: programa de utilidad de activación de licencia de servidor](#)

Acerca de esta tarea

Este procedimiento establece la compatibilidad de la aplicación V12R1Mnnn para un cliente o controlador que necesita utilizar capacidades de servidor que requieren una compatibilidad de aplicación

superior a V12R1M500. Si el cliente o el controlador no utilizan las capacidades de un nuevo nivel de función, no es necesario que actualice su configuración de compatibilidad de aplicaciones.

Procedimiento

1. Establecer el nivel de compatibilidad de aplicaciones de servidor (Db2 for z/OS , APPLCOMPAT).
 - a) Para las aplicaciones cliente que contienen sentencias SQL estáticas, vuelva a vincular los paquetes de aplicaciones estáticas con el nuevo valor APPLCOMPAT, en el cliente y en el servidor.
 - b) Para las aplicaciones cliente que solo contienen sentencias SQL dinámicas, vincule o vuelva a vincular los paquetes cliente o controlador con el nuevo valor APPLCOMPAT, en el servidor.

Consejo: La vinculación de copias de paquetes y la conservación de los paquetes originales del controlador le permiten acceder a nuevas capacidades para las aplicaciones que las necesitan, al tiempo que garantiza la estabilidad de las aplicaciones que no deben estar expuestas a incompatibilidades.

Solo para controladores, puede utilizar trabajos que se proporcionan con Db2 for z/OS, en el conjunto de datos *prefix.SDSNSAMP*, para vincular o revincular los paquetes de controladores en el servidor. Para ejecutar esos trabajos, siga estos pasos:

 - i) Personalice los trabajos [DSNTIJLC](#) y [DSNTIJLR](#), siguiendo las instrucciones de los prólogos de los trabajos.
 - ii) Si existe la posibilidad de que aún necesite ejecutar aplicaciones con un controlador que se encuentra en el nivel de compatibilidad de aplicaciones antiguo, ejecute DSNTIJLC para vincular copias de los paquetes de controladores en el nuevo nivel de compatibilidad de aplicaciones, dejando los paquetes en la colección NULLID en el nivel de compatibilidad de aplicaciones antiguo. En la mayoría de los casos, debería hacerlo.
 - Si está seguro de que no necesita ejecutar aplicaciones con un controlador en el nivel de compatibilidad de aplicaciones antiguo, ejecute el trabajo DSNTIJLR para volver a vincular los paquetes de cliente o controlador en el nuevo nivel de compatibilidad de aplicaciones.
 - iii) Si vincula copias de los paquetes de controladores para el nuevo nivel de función, cambie los controladores al nuevo nivel de función modificando la propiedad que controla el conjunto de paquetes actual para que coincida con el ID de colección de las nuevas copias de paquetes.
 - Para los controladores CLI o ODBC, cambie el valor de la palabra clave de configuración CLI/ ODBC CurrentPackageSet.
 - Para el IBM Data Server Driver for JDBC and SQLJ, cambie el valor de la propiedad DB2DataSource.currentPackageSet Connection o DataSource .
 2. Establezca el valor de compatibilidad de la aplicación cliente para controlar las capacidades de las aplicaciones cliente cuando un cliente o controlador contenga cambios que habiliten nuevas capacidades del servidor. Si establece el nivel de compatibilidad de la aplicación cliente, su valor debe ser menor o igual que el nivel de compatibilidad de la aplicación servidor.
- Realice una de las siguientes acciones para establecer el valor de compatibilidad de la aplicación cliente.
- Para los controladores CLI o ODBC o IBMODBC Clientes de Data Server, cambie el valor de la palabra clave de configuración CLI/ ClientApplCompat.
- Para ello, añada una línea similar a este ejemplo a la sección <databases> o a la sección <dsn> del archivo db2dsdriver.cfg.
- ```
<parameter name="clientApplCompat" value="V12R1M501" />
```
- Para obtener más información, consulte:
- [Archivo de configuración de IBM Data Server Driver](#)
  - [Instalación del software IBM Data Server Driver Package en sistemas operativos Linux y UNIX](#) (incluye información sobre cómo crear y rellenar el archivo db2dsdriver.cfg )

- Palabra clave de configuración ClientApplCompat para IBM Data Server Driver
- Para el IBM Data Server Driver for JDBC and SQLJ, cambie el valor de la propiedad DB2BaseDataSource.clientApplcompat Connection o DataSource .

### Referencia relacionada

[IBM Data Server Driver for JDBC and SQLJ propiedades para Db2 for z/OS \( Db2 Application Programming for Java\)](#)

[-DISPLAY LOCATION comando \(Db2\) \( Db2 Comandos\)](#)

[-ACTIVATE comando \(Db2\) \( Db2 Comandos\)](#)

### Información relacionada

[-30025 \(Db2 Codes\)](#)

[DSNL200I \(Db2 Messages\)](#)

## DSNTIJLC

Migrar paquetes de Db2 Connect para admitir un nuevo nivel de funciones.

```
//*****
//** JOB NAME = DSNTIJLC
//**
//** DESCRIPTIVE NAME = INSTALLATION JOB STREAM
//**
//** Licensed Materials - Property of IBM
//** 5650-DB2
//** (C) COPYRIGHT 2016 IBM Corp. All Rights Reserved.
//**
//** STATUS = Version 12
//**
//** FUNCTION = Migrate DB2 Connect packages to support a new
//** function level.
//**
//** PSEUDOCODE =
//** DSNTIRU STEP Bind Copy the IBM JDBC and CLI standard
//** set of packages to a new collection in
//** order to override the APPLCOMPAT package
//** option.
//**
//** NOTES =
//** (1) This job includes an in-stream data set having
//** DB2 bind statements that contain substitution
//** symbols. For example:
//** BIND PACKAGE (&TGTCOLID) +
//** COPY(&SRCCOLID..SYSLH100) +
//** APPLCOMPAT(&APPLCMPT)
//** where
//** &TGTCOLID is the name of the collection-ID to
//** bind from copy. The DB2-supplied
//** setting is 'NULLID_V12R1M500'. Use the
//** SET TGTCOLID statement in job step
//** DSNTIRU to specify a different setting.
//** &SRCCOLID is the name of the collection-ID to copy
//** from. The DB2-supplied setting
//** is 'NULLID'. Use the SET SRCCOLID
//** statement in job step DSNTIRU to
//** specify a different setting.
//** &APPLCMPT is the DB2 application compatibility
//** level. The DB2-supplied setting is
//** 'V12R1M500'. Use the SET APPLCMPT
//** statement in job step DSNTIRU to
//** specify a different setting.
//**
//** Attention JES3 users: Symbolic substitution within
//** in-stream data sets in JES3 requires z/OS 2.2 or
//** above. In order to run this job on JES2 using z/OS
//** 2.1, you need to make the following manual changes:
//** (a) Remove the EXPORT SYMLIST and all SET statements
//** (b) Change all occurrences of &TGTCOLID to the name
//** of the collection-ID to bind from copy.
//** (c) Change all occurrences of &SRCCOLID to the name
//** of the collection-ID to copy from.
//** (d) Change all occurrences of &APPLCMPT to the DB2
//** application compatibility setting.
//**
//** (2) Before running this job, customize it as follows:
```

```

//* (a) Add a valid job card.
//* (b) Locate and change all occurrences of the
//* following strings as indicated:
//* - '!DSN!' to the name of the DB2 subsystem.
//* - 'DSN!!0' to the prefix of the DB2 target
//* libraries for the DB2 subsystem.
//* (c) Set TGTCOLID, SRCCOLID, and APPLCMPT as
//* described above.
//*
//* CHANGE LOG =
//* 11/08/2016 Job created S28617 PI74456
//*
//JOBLIB DD DISP=SHR,
// DSN=DSN!!0.SDSNLOAD
//*
//* Symbolic substitution requires z/OS 2.2, or z/OS 2.1 with JES2.
// EXPORT SYMLIST=(TGTCOLID,SRCCOLID,APPLCMPT)
// SET TGTCOLID='NULLID_V12R1M500'
// SET SRCCOLID='NULLID'
// SET APPLCMPT='V12R1M500'
//DSNTIRU EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *,SYMBOLS=JCLONLY
DSN SYSTEM(!DSN!)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLH100) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLH101) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLH102) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLH200) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLH201) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLH202) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLH300) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLH301) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLH302) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLH400) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLH401) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLH402) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLN100) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLN101) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLN102) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLN200) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLN201) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLN202) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLN300) +

```

```

APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLN301) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLN302) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLN400) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLN401) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSLN402) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSH100) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSH101) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSH102) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSH200) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSH201) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSH202) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSH300) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSH301) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSH302) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSH400) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSH401) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSH402) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSN100) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSN101) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSN102) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSN200) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSN201) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSN202) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSN300) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSN301) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSN302) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSN400) +
APPLCOMPAT(&APPLCMPT)

```

```

BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSN401) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSN402) +
APPLCOMPAT(&APPLCMPT)
BIND PACKAGE (&TGTCOLID) +
COPY(&SRCCOLID..SYSSTAT) +
APPLCOMPAT(&APPLCMPT)
/*
/*

```

## DSNTIJLR

Migrar paquetes de Db2 Connect para admitir un nuevo nivel de funciones.

```

//*****
//** JOB NAME = DSNTIJLR
//**
//** DESCRIPTIVE NAME = INSTALLATION JOB STREAM
//**
//** Licensed Materials - Property of IBM
//** 5650-DB2
//** (C) COPYRIGHT 2016 IBM Corp. All Rights Reserved.
//**
//** STATUS = Version 12
//**
//** FUNCTION = Migrate DB2 Connect packages to support a new
//** function level.
//**
//** PSEUDOCODE =
//** DSNTIRU STEP Rebind the IBM JDBC and CLI standard
//** set of packages to a new collection in
//** order to override the APPLCOMPAT package
//** option.
//**
//** NOTES =
//** (1) This job includes an in-stream data set having
//** DB2 bind statements that contain substitution
//** symbols. For example:
//** REBIND PACKAGE (&SRCCOLID..SYSLH100) +
//** APPLCOMPAT(&APPLCMPT)
//** where
//** &SRCCOLID is the name of the collection-ID
//** owning the package to be rebound.
//** The DB2-supplied setting is
//** 'NULLID V12R1M500'. Use the SET
//** SRCCOLID statement in job step
//** DSNTIRU to specify a different
//** setting.
//** &APPLCMPT is the DB2 application compatibility
//** level. The DB2-supplied setting is
//** 'V12R1M500'. Use the SET APPLCMPT
//** statement in job step DSNTIRU to
//** specify a different setting.
//**
//** Attention JES3 users: Symbolic substitution within
//** in-stream data sets in JES3 requires z/OS 2.2 or
//** above. In order to run this job on JES2 using z/OS
//** 2.1, you need to make the following manual changes:
//** (a) Remove the EXPORT SYMLIST and all SET statements
//** (b) Change all occurrences of &SRCCOLID to the name
//** of the collection-ID owning the package to be
//** rebound
//** (c) Change all occurrences of &APPLCMPT to the DB2
//** application compatibility setting.
//**
//** (2) Before running this job, customize it as follows:
//** (a) Add a valid job card.
//** (b) Locate and change all occurrences of the
//** following strings as indicated:
//** - '!DSN!' to the name of the DB2 subsystem.
//** - 'DSN!!0' to the prefix of the DB2 target
//** libraries for the DB2 subsystem.
//** (c) Set SRCCOLID and APPLCMPT as described above.
//**
//** CHANGE LOG =
//** 11/08/2016 Job created
//**

```

S28617 PI74456

```

//JOBLIB DD DISP=SHR,
// DSN=DSN!!0.SDSNLOAD
//*
//** Symbolic substitution requires z/OS 2.2, or z/OS 2.1 with JES2.
// EXPORT SYMLIST=(SRCCOLID,APPLCMPT)
// SET SRCCOLID='NULLID_V12R1M500'
// SET APPLCMPT='V12R1M500'
//DSNTIRU EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD * SYMBOLS=JCLONLY
 DSN SYSTEM(DB2A)
REBIND PACKAGE (&SRCCOLID..SYSLH100) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLH101) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLH102) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLH200) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLH201) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLH202) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLH300) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLH301) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLH302) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLH400) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLH401) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLH402) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLN100) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLN101) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLN102) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLN200) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLN201) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLN202) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLN300) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLN301) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLN302) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLN400) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLN401) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSLN402) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSH100) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSH101) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSH102) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSH200) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSH201) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSH202) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSH300) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSH301) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSH302) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSH400) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSH401) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSH402) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSN100) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSN101) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSN102) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSN200) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSN201) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSN202) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSN300) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSN301) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSN302) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSN400) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSN401) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSN402) APPLCOMPAT(&APPLCMPT)
REBIND PACKAGE (&SRCCOLID..SYSSTAT) APPLCOMPAT(&APPLCMPT)
*/
/*

```

## Usar tablas de perfiles para controlar qué niveles de compatibilidad de aplicaciones de Db2 for z/OS se usar para aplicaciones específicas de cliente de servidor de datos

Los perfiles se pueden utilizar para controlar qué aplicaciones cliente utilizan funciones que están asociadas con un nivel de compatibilidad de aplicaciones de Db2 for z/OS específico. Esta capacidad permite que las aplicaciones cliente que no necesitan utilizar nuevas funciones sigan conectándose a un servidor Db2 for z/OS con un nivel de compatibilidad de aplicación anterior.

## Acerca de esta tarea

En este procedimiento de ejemplo, la aplicación cliente ACCTG\_APP501 debe utilizar Db2 for z/OS capacidades que están disponibles en el nivel de compatibilidad de aplicaciones V12R1M501. Todas las demás aplicaciones de cliente deben utilizar capacidades que estén disponibles en el nivel de compatibilidad de aplicaciones V12R1M500 o inferior.

## Procedimiento

1. En el sistema operativo del cliente, vincule los paquetes de controladores del cliente en dos colecciones:
  - Una colección con el nombre de colección predeterminado NULLID y con la opción APPLCOMPAT establecida en V12R1M500. Si ya ha vinculado los paquetes de controladores cliente en la colección NULLID con APPLCOMPAT establecido en V12R1M500, no es necesario que los vincule de nuevo.
  - Otra colección con un nombre diferente, como NULLID\_NF, y con la opción APPLCOMPAT establecida en V12R1M501.
  - Para el IBM Data Server Driver for JDBC and SQLJ, siga estos pasos para vincular los paquetes de controladores:
    - Si aún no lo ha hecho, invoque la utilidad DB2Binder con una instrucción de control como ésta para crear una colección con el nombre predeterminado NULLID y con la compatibilidad de la aplicación establecida en V12R1M500:

```
java com.ibm.db2.jcc.DB2Binder -url jdbc:db2://sys1.svl.ibm.com:5021/STLEC1 \
-user user -password password \
-bindoptions "APPLCOMPAT V12R1M500" -action REPLACE
```
    - Invoque la utilidad DB2Binder con una declaración de control como ésta para crear una colección denominada NULLID\_NF, con la compatibilidad de la aplicación establecida en V12R1M501:

```
java com.ibm.db2.jcc.DB2Binder -url jdbc:db2://sys1.svl.ibm.com:5021/STLEC1 \
-collection NULLID_NF \
-user user -password password \
-bindoptions "APPLCOMPAT V12R1M501" -action REPLACE
```
- Para el IBM Data Server Driver for ODBC and CLI, siga estos pasos para vincular los paquetes de controladores:
  - Si CLI/ ODBC, palabra clave de configuración OnlyUseBigPackages=1, no es necesario vincular los paquetes de controladores.
  - Si la palabra clave de configuración CLI/ ODBC OnlyUseBigPackages=0, debe vincular paquetes pequeños con compatibilidad de aplicaciones establecida en V12R1M501:

```
db2 bind '%DB2PATH%\bnd\ddcsmvs.lst' blocking all sqlerror continue \
grant public action replace collection NULLID_NF \
generic \"APPLCOMPAT V12R1M501\"
```

2. En Db2 for z/OS, cree un perfil para la aplicación de cliente ACCTG\_APP501 insertando filas en las tablas SYSIBM.DSN\_PROFILE\_TABLE y SYSIBM.DSN\_PROFILE\_ATTRIBUTES. El perfil ordena a Db2 que utilice el controlador con paquetes en la colección NULLID\_NF cuando se ejecute la aplicación ACCTG\_APP501. Debido a que los paquetes de controladores en la colección NULLID\_NF están vinculados con la opción APPLCOMPAT V12R1M501, ACCTG\_APP501 puede usar capacidades que están disponibles en el nivel de compatibilidad de aplicaciones V12R1M501.

```
INSERT INTO SYSIBM.DSN_PROFILE_TABLE
(PROFILEID, CLIENT_APPLNAME, PROFILE_ENABLED)
VALUES (1002, 'ACCTG_APP501', 'Y');
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
(PROFILEID, KEYWORDS, ATTRIBUTE1)
VALUES (1002, 'SPECIAL_REGISTER', 'SET CURRENT PACKAGE PATH=NULLID_NF');
```

**Importante:** Aunque PKGNAME puede utilizarse como categoría de filtrado para las filas de la tabla de perfiles que utilizan el valor 'SPECIAL\_REGISTER' para KEYWORDS, cuando se utilizan controladores de cliente, no debe utilizarse PKGNAME solo o en combinación con COLLID.

3. En Db2 for z/OS, ejecute el comando -START PROFILE para cargar en la memoria las tablas de perfiles actualizadas.

## Resultados

ACCTG\_APP501, la aplicación, ahora puede conectarse correctamente a Db2 for z/OS y utilizar paquetes de controladores de servidor de datos en la colección NULLID\_NF. Todas las demás aplicaciones pueden conectarse a Db2 for z/OS y utilizar paquetes de controladores de servidor de datos en la colección NULLID.

Puede verificar el nivel de compatibilidad de la aplicación y la colección que se están utilizando para la aplicación de su cliente añadiendo código para ejecutar una consulta como esta en su aplicación.

```
SELECT CURRENT APPLICATION COMPATIBILITY,
GETVARIABLE('SYSIBM PACKAGE_SCHEMA')
FROM SYSIBM.SYSDUMMY1
```

Por ejemplo, puede añadir un código como este a una aplicación Java:

```
String currApplcompat, appCollection;
Connection con;
Statement stmt;
ResultSet rs;
...
stmt = con.createStatement(); // Create a Statement object
rs = stmt.executeQuery("SELECT CURRENT APPLICATION COMPATIBILITY," +
" GETVARIABLE('SYSIBM PACKAGE_SCHEMA') " +
" FROM SYSIBM.SYSDUMMY1"); // Get the result table from the query
while (rs.next()) { // Position the cursor
currApplcompat = rs.getString(1); // Retrieve the application compatibility
System.out.println("APPLCOMPAT = " + currApplcompat);
 // Print the application compatibility
appCollection = rs.getString(1); // Retrieve the collection name
System.out.println("COLLID = " + appCollection);
 // Print the collection name
```

Para la aplicación ACCTG\_APP501, la consulta debe devolver un valor de V12R1M501 para la compatibilidad de la aplicación actual y NULLID\_NF para el nombre de la colección.

### Conceptos relacionados

[Enlace de programas de utilidad de base de datos en Db2 Connect](#)

### Tareas relacionadas

[Establecimiento de niveles de compatibilidad de aplicaciones para clientes y controladores de servidores de datos](#)

IBM los clientes y controladores de servidores de datos que utilizan capacidades de Db2 for z/OS con un requisito de nivel de función superior a V12R1M500 requieren pasos adicionales de preparación del programa.

[Establecimiento de registros especiales utilizando tablas de perfiles \(Db2 Administration Guide\)](#)

### Referencia relacionada

[Programa de utilidad DB2Binder \(Db2 Application Programming for Java\)](#)

[Palabra clave de configuración del controlador de servidor de datos de IBM y CLI/ODBC OnlyUseBigPackages](#)

## Nivel de compatibilidad de aplicaciones V11R1

Cuando establece el nivel de compatibilidad de aplicaciones en V11R1, las aplicaciones que intentan utilizar las funciones y características que se introducen en Db2 12 o posteriores pueden comportarse de forma diferente o recibir un error.

Cuando se activa una nueva función en el entorno Db2 12, puede ejecutar aplicaciones individuales con algunas de las características y el comportamiento de Db2 11. Es decir, sus aplicaciones pueden seguir experimentando un comportamiento V11R1 e después de que se active la nueva función en Db2 12 o posterior. A continuación, puede migrar cada aplicación a un nuevo valor de compatibilidad de aplicación por separado hasta que se migren todos. Si el nivel de compatibilidad de la aplicación está configurado como "V11R1" e intentas utilizar las nuevas funciones de una versión posterior, SQL podría comportarse de manera diferente o dar lugar a códigos SQL negativos, como -4743.

Para ver ejemplos de las nuevas capacidades de SQL que no se pueden utilizar en la compatibilidad de aplicaciones V11R1, consulte los siguientes temas:

- [Cambios en SQL en Db2 13](#)
- [Cambios en SQL en Db2 12](#)

**Consejo:** Para obtener los mejores resultados, configure su entorno de desarrollo para que utilice el nivel de compatibilidad de aplicación más bajo en el que se ejecutará la aplicación en el entorno de producción. Para SQL dinámico, recuerde tener en cuenta los niveles de compatibilidad de la aplicación de los paquetes cliente y NULLID. Si desarrolla y prueba aplicaciones en un nivel de compatibilidad de aplicaciones más alto e intenta ejecutarlas en un nivel más bajo en producción, es probable que encuentre el código SQL -4743 y otros errores cuando implemente las aplicaciones en producción.



Puede ejecutar rastreos de contabilidad o supervisión de nivel de paquete con el IFCID 0239 y el campo de revisión QPACINCOMPAT, que indica un cambio incompatible con SQL. Si se inicia un rastreo para el IFCID 0376, y la compatibilidad de la aplicación se establece para una versión anterior, los detalles sobre las características y funciones que tienen un cambio en el comportamiento se escriben en el campo QW0376FN.



Un entorno Db2 12 migrado se comporta con compatibilidad de aplicaciones V11R1 hasta que se activa el nivel de función 500 o superior.

La siguiente tabla muestra algunas características y funciones que están controladas por la compatibilidad de la aplicación, y los resultados si especifica V11R1. Si se rastrea una diferencia de comportamiento, se muestra el código de función de rastreo de IFCID.

Tabla 131. Comportamiento de la compatibilidad de aplicaciones V11R1

Característica o función	Resultado con compatibilidad de aplicaciones V11R1	Código de función de rastreo IFCID 0376
La función incorporada POWER devuelve un resultado con el tipo de datos DOUBLE. El resultado está fuera de rango.	SQLCODE -802	1201
CURRENT_SERVER o CURRENT_TIMEZONE se utiliza como nombre de columna o nombre de variable.	SQLCODE -206	1204

## Cambios en el nivel de compatibilidad de la aplicación SQL V11R1

Las siguientes capacidades SQL están disponibles en el modo new-function de Db2 11, o posterior, para aplicaciones que se ejecutan en el nivel de compatibilidad de aplicaciones V11R1, o superior.

Cualquier intento de utilizar las capacidades de la siguiente tabla en un nivel de compatibilidad de aplicación inferior a V11R1 da lugar a una condición de error, como el código SQL -4743 u otros. Para más restricciones que se aplican a niveles de compatibilidad de aplicación más bajos, consulte ["Nivel de compatibilidad de aplicaciones V11R1"](#) en la página 878.

## Nuevas sentencias SQL en Db2 11

GUPI

Tabla 132. Nuevas sentencias SQL en Db2 11

Sentencia SQL	Descripción
<a href="#">CREATE</a>	La sentencia SQL CREATE TYPE (matriz) define un tipo de matriz en el servidor actual.
<a href="#">CREATE VARIABLE declaración ( Db2 SQL)</a>	La sentencia CREATE VARIABLE crea una variable global en el servidor actual.
<a href="#">SET CURRENT ACCELERATOR declaración ( Db2 SQL)</a>	SET CURRENT ACCELERATOR cambia el valor del registro especial CURRENT ACCELERATOR.
<a href="#">SET CURRENT APPLICATION COMPATIBILITY declaración ( Db2 SQL)</a>	La sentencia SET CURRENT APPLICATION COMPATIBILITY cambia el valor del registro especial CURRENT APPLICATION COMPATIBILITY.
<a href="#">SET CURRENT TEMPORAL BUSINESS_TIME declaración ( Db2 SQL)</a>	La sentencia SET CURRENT TEMPORAL BUSINESS_TIME cambia el valor del registro especial CURRENT TEMPORAL BUSINESS_TIME.
<a href="#">SET CURRENT TEMPORAL SYSTEM_TIME declaración ( Db2 SQL)</a>	La sentencia SET CURRENT TEMPORAL SYSTEM_TIME cambia el valor del registro especial CURRENT TEMPORAL SYSTEM_TIME.
<a href="#">SET assignment-statement declaración ( Db2 SQL)</a>	La sentencia SET <i>assignment-statement</i> es una reclasificación de la documentación de las sentencias SET <i>host-variable</i> y SET <i>transition-variable</i> en una única sentencia.

GUPI

## Cambios en la declaración SQL en Db2 11

La siguiente tabla muestra los cambios en las sentencias SQL existentes que las aplicaciones pueden utilizar en el nivel de compatibilidad de aplicaciones V11R1 o superior.

GUPI

Tabla 133. Cambios en las sentencias SQL existentes en Db2 11

Sentencia SQL	Descripción de mejoras y notas
<a href="#">ALTER FUNCTION (SQL escalar)</a>	<b>Cláusulas nuevas:</b> HORARIO DE OFICINA SENSIBLE SENSIBLE AL TIEMPO DEL SISTEMA ARCHIVO SENSIBLE APPLCOMPAT  <b>Cláusulas modificadas:</b> <i>data-type</i> , <i>data-type2</i> puede incluir <i>array-type-name</i> .

Tabla 133. Cambios en las sentencias SQL existentes en Db2 11 (continuación)

Sentencia SQL	Descripción de mejoras y notas
<u>ALTER PROCEDURE (SQL nativo)</u>	<p><b>Cláusulas nuevas:</b></p> <p>HORARIO DE OFICINA SENSIBLE SENSIBLE AL TIEMPO DEL SISTEMA ARCHIVO SENSIBLE APLCOMPAT</p> <p><b>Cláusulas modificadas:</b></p> <p><i>data-type</i> puede incluir <i>array-type-name</i>.</p>
<u>ALTER TABLE</u>	<p><b>Cláusulas nuevas:</b></p> <p>DROP COLUMN ACTIVAR ARCHIVO DESACTIVAR ARCHIVO</p> <p><b>Cláusulas modificadas:</b></p> <p>Las cláusulas ALTER PARTITION que cambian los valores clave de los límites ahora dan como resultado cambios de definición pendientes.</p>
<u>ALTER TABLESPACE</u>	<p><b>Cláusulas modificadas:</b></p> <p>PCTFREE ahora puede incluir FOR UPDATE <i>smallint</i>.</p>
<u>comentario</u>	<p><b>Cláusulas modificadas:</b></p> <p><i>data-type</i> puede incluir <i>array-type-name</i>.</p>
<u>CREATE FUNCTION (SQL escalar)</u>	<p><b>Cláusulas nuevas:</b></p> <p>HORARIO DE OFICINA SENSIBLE SENSIBLE AL TIEMPO DEL SISTEMA ARCHIVO SENSIBLE APLCOMPAT</p> <p><b>Cláusulas modificadas:</b></p> <p><i>data-type</i> puede incluir <i>array-type-name</i>.</p>
<u>Crear índice</u>	<p><b>Cláusulas nuevas:</b></p> <p>INCLUDE NULL KEYS EXCLUDE NULL KEYS</p>
<u>CREATE PROCEDURE (externo)</u>	<p><b>Cláusulas modificadas:</b></p> <p><i>data-type</i> puede incluir <i>array-type-name</i>.</p>
<u>CREATE PROCEDURE (SQL nativo)</u>	<p><b>Cláusulas nuevas:</b></p> <p>HORARIO DE OFICINA SENSIBLE SENSIBLE AL TIEMPO DEL SISTEMA ARCHIVO SENSIBLE APLCOMPAT</p> <p><b>Cláusulas modificadas:</b></p> <p><i>data-type</i> puede incluir <i>array-type-name</i>.</p>

Tabla 133. Cambios en las sentencias SQL existentes en Db2 11 (continuación)

<b>Sentencia SQL</b>	<b>Descripción de mejoras y notas</b>
<u>CREAR TABLESPACE</u>	<b>Cláusulas modificadas:</b> PCTFREE ahora puede incluir FOR UPDATE <i>smallint</i> .
<u>DECLARAR TABLA TEMPORAL GLOBAL</u>	<b>Cláusulas nuevas:</b> LOGGED NOT LOGGED
<u>DROP</u>	<b>Cláusulas modificadas:</b> <i>data-type</i> puede incluir <i>array-type-name</i> .
<u>EXECUTAR</u>	<b>Cláusulas modificadas:</b> El objeto de la cláusula USING puede ser una variable SQL, un parámetro SQL, una variable global o un variable host.
<u>FETCH</u>	<b>Cláusulas modificadas:</b> El objeto de la cláusula INTO puede ser una variable host, un parámetro SQL, una variable SQL, una variable de transición o un elemento de matriz.
<u>GRANT (privilegios de función o de procedimiento)</u>	<b>Cláusulas modificadas:</b> <i>data-type</i> puede incluir <i>array-type-name</i> .
<u>GRANT (privilegios de tipo o JAR)</u>	<b>Cláusulas modificadas:</b> El objeto de la cláusula TYPE puede ser un tipo diferenciado o un tipo de matriz.
<u>Abrir</u>	<b>Cláusulas modificadas:</b> El objeto de la cláusula USING puede ser una variable SQL, un parámetro SQL, una variable global o un variable host.
<u>REVOKE (privilegios de función o procedimiento)</u>	<b>Cláusulas modificadas:</b> <i>data-type</i> puede incluir <i>array-type-name</i> .
<u>REVOKE (privilegios de tipo o JAR)</u>	<b>Cláusulas modificadas:</b> El objeto de la cláusula TYPE puede ser un tipo diferenciado o un tipo de matriz.
<u>SELECCIONAR EN</u>	<b>Cláusulas modificadas:</b> El objeto de la cláusula INTO puede ser una variable de host, una variable global, un parámetro SQL, una variable SQL, una variable de transición o un elemento matriz.
<u>ESTABLECER RUTA</u>	<b>Cláusulas modificadas:</b> SYSTEM PATH ahora incluye los esquemas "SYSIBM", "SYSFUN", "SYSPROC", "SYSIBMADM".

Tabla 133. Cambios en las sentencias SQL existentes en Db2 11 (continuación)

Sentencia SQL	Descripción de mejoras y notas
Sentencia SQL con <u>subselect</u>	<p><b>Cláusulas modificadas:</b>  <i>collection-derived-table</i> se ha añadido a <i>table-reference</i> en la cláusula FROM de <i>subselect</i>.</p> <p><b>Otros cambios:</b>  Una función definida por el usuario que se define con MODIFIES SQL DATA puede invocarse en una subselección.</p>
<u>VALORES EN</u>	<p><b>Cláusulas modificadas:</b></p> <p>El objeto de la cláusula INTO puede ser una variable de host, una variable global, un parámetro SQL, una variable SQL, una variable de transición o un elemento matriz.</p>



## Nuevas funciones integradas en Db2 11

Db2 11 introduce nuevas funciones integradas que mejoran la potencia del lenguaje SQL. En la tabla siguiente se muestran las funciones incorporadas nuevas.



Tabla 134. Nuevas funciones integradas en Db2 11

Nombre de función	Descripción
Función agregada ARRAY_AGG (SQL Db2 )	La función ARRAY_AGG devuelve una matriz en la que cada valor del conjunto de entrada se asigna a un elemento de la matriz.
Función escalar ARRAY_DELETE (SQL Db2 )	La función ARRAY_DELETE elimina elementos de una matriz.
Función escalar ARRAY_FIRST (SQL Db2 )	La función ARRAY_FIRST devuelve el valor de índice de matriz mínimo de una matriz.
Función escalar ARRAY_LAST (SQL Db2 )	La función ARRAY_LAST devuelve el valor de índice de matriz máximo de una matriz.
Función escalar ARRAY_NEXT ( Db2 SQL)	La función ARRAY_NEXT devuelve el siguiente valor más alto de índice de matriz, en relación con un valor de índice de matriz especificado.
Función escalar ARRAY_PRIOR (SQL Db2 )	La función ARRAY_PRIOR devuelve el siguiente valor más bajo de índice de matriz, en relación con un valor de índice de matriz especificado.
Función de tabla BLOCKING_THREADS ( Db2 SQL)	La función BLOCKING_THREADS devuelve una tabla que contiene una fila por cada bloqueo o reclamación que los hilos mantienen contra bases de datos especificadas.
CARDINALITY función escalar ( Db2 SQL)	La función CARDINALITY devuelve el número de elementos de una matriz.

Tabla 134. Nuevas funciones integradas en Db2 11 (continuación)

Nombre de función	Descripción
<a href="#">CHAR9 función escalar ( Db2 SQL)</a>	<p>La función CHAR9 devuelve una representación de serie de caracteres de longitud fija del argumento. La función CHAR9 está prevista para que sea compatible con releases anteriores de Db2 for z/OS que dependen del formato del resultado devuelto para los valores de entrada decimales en la Versión 9 y anteriores.</p> <p><b>Importante:</b> Para aplicaciones portátiles que podrían ejecutarse en plataformas distintas de Db2 for z/OS, utilice en su lugar la función CHAR. Otros productos de la familia de e Db2 s no son compatibles con la función de e CHAR9.</p>
<a href="#">Función escalar MAX_CARDINALITY ( Db2 SQL)</a>	<p>La función MAX_CARDINALITY devuelve el número máximo de elementos que puede contener una matriz.</p>
<a href="#">MEDIAN</a>	<p>La función MEDIAN devuelve la mediana de un conjunto de números. Esta función sólo se puede ejecutar en un servidor de acelerador.</p>
<a href="#">Función escalar TRIM_ARRAY (SQL Db2 )</a>	<p>La función TRIM_ARRAY suprime los elementos del final de una matriz común.</p>
<a href="#">VARCHAR9 función escalar ( Db2 SQL)</a>	<p>La función VARCHAR9 función devuelve una representación del argumento en forma de cadena de caracteres de longitud fija. La función " VARCHAR9 " está pensada para ser compatible con versiones anteriores de " Db2 for z/OS " que dependen del formato de resultado que se devuelve para valores de entrada decimales en la versión 9 y anteriores.</p> <p><b>Importante:</b> Para aplicaciones portátiles que podrían ejecutarse en plataformas distintas de Db2 for z/OS, utilice la función VARCHAR en su lugar. Otros productos de la familia de e Db2 s no son compatibles con la función de e VARCHAR9.</p>

 GUPI

**Conceptos relacionados**

[Incompatibilidades de SQL y aplicaciones entre releases \(Novedades de DB2 para z/OS\)](#)

**Información relacionada**

[Referencia SQL \(Db2 11 for z/OS\)](#)

[Guía de programación de aplicaciones y SQL \(Db2 11 for z/OS\)](#)

## Nivel de compatibilidad de aplicaciones V10R1

Cuando establece el nivel de compatibilidad de aplicaciones en V10R1, las aplicaciones que intentan utilizar las funciones y características que se introducen en Db2 11 o posteriores pueden comportarse de forma diferente o recibir un error.

En Db2 12, puede continuar ejecutando aplicaciones individuales con algunas de las características y el comportamiento de DB2 10. Es decir, sus aplicaciones pueden seguir experimentando un comportamiento V10R1 e mientras están en Db2 12, independientemente de si se activa la nueva función. A continuación, puede migrar cada aplicación a un nuevo valor de compatibilidad de aplicación por separado hasta que se migren todos. Si la compatibilidad de la aplicación está configurada como " V10R1 " e intentas utilizar las nuevas funciones de una versión posterior, SQL podría comportarse de manera diferente o dar lugar a códigos SQL negativos, como -4743 y otros.

Puede ejecutar rastreos de contabilidad o supervisión de nivel de paquete con el IFCID 0239 y el campo de revisión QPACINCOMPAT, que indica un cambio incompatible con SQL. Si se inicia un rastreo para el IFCID 0376, y la compatibilidad de la aplicación se establece para una versión anterior, los detalles sobre las características y funciones que tienen un cambio en el comportamiento se escriben en el campo QW0376FN.

Un entorno de migración de Db2 12 se comporta con compatibilidad de aplicaciones de V11R1 hasta que se activa el nivel de función 500 o superior. Las incompatibilidades de aplicación y SQL se describen en la información de migración para cada versión.

La siguiente tabla muestra ejemplos de muchas de las nuevas capacidades de Db2 11 características y funciones que están controladas por la compatibilidad de la aplicación, y los resultados si especifica V10R1. Si se rastrea una diferencia de comportamiento, se muestra el código de función de rastreo de IFCID.

Además, las nuevas capacidades SQL de las versiones posteriores de Db2 no se pueden utilizar en el nivel de compatibilidad de aplicaciones V10R1. Para ver las listas de estas capacidades SQL, consulte:

- [Cambios en SQL en Db2 13](#)
- [Cambios en SQL en Db2 12](#)

**Consejo:** Para obtener los mejores resultados, configure su entorno de desarrollo para que utilice el nivel de compatibilidad de aplicación más bajo en el que se ejecutará la aplicación en el entorno de producción. Para SQL dinámico, recuerde tener en cuenta los niveles de compatibilidad de la aplicación de los paquetes cliente y NULLID. Si desarrolla y prueba aplicaciones en un nivel de compatibilidad de aplicaciones más alto e intenta ejecutarlas en un nivel más bajo en producción, es probable que encuentre el código SQL -4743 y otros errores cuando implemente las aplicaciones en producción.

Tabla 135. Comportamiento de compatibilidad de aplicación V10R1

Característica o función	Resultado con compatibilidad de aplicación V10R1	Código de función de rastreo IFCID 0376
Una sentencia SQL en una aplicación cliente incluye una conversión no soportada (de un tipo de serie a un tipo numérico o de un tipo numérico a un tipo de serie), y la conversión implícita está inhabilitada (DDF_COMPATIBILITY se establece en SP_PARMS_NJV o en DISABLE_IMPCAST_NJV).	SQLCODE -301	<a href="#">7“1” en la página 888</a>
Una aplicación cliente ejecuta una sentencia CALL de SQL para ejecutar un procedimiento almacenado Db2 for z/OS. El parámetro de subsistema DDF_COMPATIBILITY se establece en SP_PARMS_NJV para aplicaciones cliente que no sean aplicaciones Java o SP_PARMS_JV para aplicaciones Java.	Los tipos de datos de los datos que se devuelven de la sentencia CALL de SQL coinciden con los tipos de datos de los argumentos de la sentencia CALL. Este comportamiento es compatible con el comportamiento anterior a la Versión 10.	<a href="#">8“1” en la página 888</a>

Tabla 135. Comportamiento de compatibilidad de aplicación V10R1 (continuación)

Característica o función	Resultado con compatibilidad de aplicación V10R1	Código de función de rastreo IFCID 0376
Una aplicación cliente accede a Db2 11 desde un cliente IBM Data Server Driver for JDBC and SQLJ. El parámetro de subsistema DDF_COMPATIBILITY se establece en IGNORE_TZ para aplicaciones Java.	El servidor Db2 ignora la parte TIMEZONE, añadida por IBM Data Server Driver for JDBC and SQLJ, del valor de la entrada TIMESTAMP WITH TIMEZONE en un destino TIMESTAMP. Este comportamiento es compatible con el comportamiento anterior a DB2 10.	9
BIF_COMPATIBILITY se establece en V9_TRIM, y la entrada <i>expresión-serie</i> es datos mixtos EBCDIC para la función incorporada RTRIM, LTRIM o STRIP.	Se ejecuta la versión DB2 9 de SYSIBM.LTRIM( <i>expresión-serie</i> ), SYSIBM.RTRIM( <i>expresión-serie</i> ) o SYSIBM.STRIP( <i>expresión-serie</i> ).	10
Una inserción o actualización implícita de un nodo de documento XML	SQLCODE -20345	1101
Una expresión de predicado con una conversión explícita o una operación con un valor no válido que no afecta a los resultados del proceso de XPath	SQLCODE -20345	1102
Cómo el recurso de límite de recursos utiliza el valor de ASUTIME para rutinas anidadas	SQLCODE -905 sólo se emite cuando se encuentra el límite ASUTIME del paquete de llamada de nivel superior.	1103
Las longitudes de los valores que se devuelven del registro especial CURRENT CLIENT_USERID, CURRENT CLIENT_WRKSTNNAME, CURRENT CLIENT_APPLNAME o CURRENT CLIENT_ACCTNG son más largas que los límites de DB2 10.	Los valores de registro especial se truncan en las longitudes máximas de DB2 10 y se rellenan con espacios en blanco	1104, 1105, 1106, 1107
Una especificación CAST (serie como TIMESTAMP) con una serie de entrada de longitud de 8 o una serie de entrada de longitud 13	Una especificación de conversión explícita de serie como TIMESTAMP interpreta una serie de caracteres de 8 bytes como un valor de Reloj del almacén y una serie de 13 bytes como valor GENERATE_UNIQUE. El resultado de CAST puede ser incorrecto.	1109
Invocación de la función incorporada SPACE o VARCHAR cuando el resultado se define como VARCHAR(32765), VARCHAR(32766) o VARCHAR(32767)	Sin error	1110, 1111
El parámetro de subsistema XML_RESTRICT_EMPTY_TAG está establecido en YES y un elemento XML vacío se serializa como <emptyElement></emptyElement>	Sin error	1112
Especificación de la opción de enlace DBPROTOCOL(DRDACBF)	DSNT298I	

Tabla 135. Comportamiento de compatibilidad de aplicación V10R1 (continuación)

Característica o función	Resultado con compatibilidad de aplicación V10R1	Código de función de rastreo IFCID 0376
Especificación de periodo que sigue el nombre de una vista en la cláusula FROM de una consulta	SQLCODE -4743	
Cláusula de periodo que sigue el nombre de una vista de destino en una sentencia UPDATE o DELETE	SQLCODE -4743	
Una sentencia SET CURRENT TEMPORAL SYSTEM_TIME	SQLCODE -4743	
Una sentencia SET CURRENT TEMPORAL BUSINESS_TIME	SQLCODE -4743	
Una referencia a una variable global	SQLCODE -4743	
Uso de operaciones de matriz y funciones incorporadas como	SQLCODE -4743	
<ul style="list-style-type: none"> <li>• Uso de la tabla derivada de la colección UNNEST</li> <li>• Uso de las funciones incorporadas ARRAY_PRIMERO, ARRAY_LAST, ARRAY_NEXT, ARRAY_PRIOR, ARRAY_AGG, TRIM_ARRAY, CARDINALITY, MAX_CARDINALITY</li> <li>• Una sentencia de asignación SET de un elemento de matriz como tabla de destino</li> <li>• Una especificación CAST con un marcador de parámetro como origen y una matriz como tipo de datos</li> </ul>		
Función de agregación que contiene la palabra clave DISTINCT y hace referencia a una columna definida con una máscara de columna	SQLCODE -20478	
Una sentencia SQL contiene la cláusula GROUP BY y hace referencia a una columna definida con una máscara de columna	SQLCODE -20478	
Una sentencia SQL contiene el operador de conjunto UNION ALL o UNION DISTINCT y hace referencia a una columna definida con una máscara de columna	SQLCODE -20478	
Una referencia a un alias para un objeto de secuencia	SQLCODE -4743	
Una referencia a una secuencia no calificada que no está resuelta en un alias público	SQLCODE -204	
SELECT con una referencia de función de tabla que incluye una cláusula de correlación con tipo	SQLCODE -4743	

Tabla 135. Comportamiento de compatibilidad de aplicación V10R1 (continuación)

Característica o función	Resultado con compatibilidad de aplicación V10R1	Código de función de rastreo IFCID 0376
Una tabla de referencia, una tabla derivada de una colección o una expresión de tabla XML que no incluye una cláusula de correlación.	SQLCODE -4743	
Una sentencia CALL que especifica un procedimiento autónomo	SQLCODE -4743	
Las siguientes asignaciones de fecha y hora: <ul style="list-style-type: none"><li>• Una representación de serie válida de una indicación de fecha y hora en una columna de fecha</li><li>• Una representación de serie válida de una indicación de fecha y hora para una columna de hora</li><li>• Una representación de serie válida de una fecha en una columna de indicación de fecha y hora</li></ul>	SQLCODE -180	

**Notas:**

1.  Para encontrar detalles sobre los parámetros incompatibles, examine el contenido de los campos QW0376SC\_Var, QW0376PR\_Var y QW0376INC\_Var. Consulte el archivo DSNWMSG para obtener más información. 

**Conceptos relacionados**

[Incompatibilidades de SQL y aplicaciones entre releases \(Novedades de DB2 para z/OS\)](#)

[Nivel de compatibilidad de aplicaciones V11R1](#)

Cuando establece el nivel de compatibilidad de aplicaciones en V11R1, las aplicaciones que intentan utilizar las funciones y características que se introducen en Db2 12 o posteriores pueden comportarse de forma diferente o recibir un error.

**Información relacionada**

[Referencia SQL \(DB2 10 for z/OS\)](#)

[Guía de programación de aplicaciones y SQL \(DB2 10 for z/OS\)](#)

## Gestión de incompatibilidades de aplicaciones

Antes de mover una aplicación a un nuevo nivel de compatibilidad de aplicaciones, necesita encontrar incompatibilidades de aplicaciones, ajustar las aplicaciones a esas incompatibilidades y verificar que las incompatibilidades ya no existen.

### Procedimiento

1. Iniciar un rastreo que incluya IFCID 0239 para capturar la información del paquete.

Por ejemplo, emita el siguiente comando START TRACE:

```
-START TRACE(ACCTG) CLASS(7,8,10)
```



2. Examine el resultado de la traza.

**PSPI** El campo QPACFLGS de IFCID 0239 contiene un bit que está activado si un paquete contiene incompatibilidades. Si este bit está desactivado, no se han detectado incompatibilidades y puede omitir el resto de los pasos. Si este bit está activado, continúe con el paso 3. **PSPI**

3. Iniciar un rastreo para IFCID 0376 para informar de la incompatibilidad de la información sobre los paquetes.

Por ejemplo, emita el siguiente comando START TRACE:**GUPI**

```
-START TRACE(PERFM) CLASS(32) IFCID(376)
```

**GUPI**

4. Ejecute la aplicación.
5. Examine el resultado de la traza.

**PSPI** Los campos IFCID 0376 contienen información sobre las incompatibilidades. Db2 escribe un único registro de seguimiento para cada instrucción SQL que sea incompatible con el nivel de función de Db2 e posterior. Ver *prefijo* de archivo.SDSNIVPD(DSNWMSGS) para ver los listados de los registros de seguimiento IFCID 0239 y 0376. **PSPI**

6. Revise la aplicación para evitar cualquier incompatibilidad.
7. Preparar la solicitud para su ejecución. Cuando vincula los paquetes para la aplicación, utilice el valor APPLCOMPAT antiguo.
8. Ejecute la aplicación.
9. Examine de nuevo la salida de trazas para verificar que las incompatibilidades ya no existen.

### Qué hacer a continuación

Cuando la aplicación se ejecuta en el nivel anterior sin incompatibilidades, vuelva a vincular el paquete con el valor APPLCOMPAT para el nuevo nivel de función.

#### Conceptos relacionados

[Rastreo de rendimiento \(Db2 Performance\)](#)

#### Referencia relacionada

[-START TRACE comando \(Db2\) \( Db2 Comandos\)](#)

[Descripciones de campos de rastreo \(Db2 Performance\)](#)

## Habilitar la compatibilidad de aplicaciones predeterminada con nivel de función 500 o superior

El parámetro del subsistema APPLCOMPAT especifica el valor predeterminado de la opción de enlace APPLCOMPAT. Antes de activar nivel de función 500 o superior , el parámetro del subsistema APPLCOMPAT debe establecerse en V11R1 o V10R1. Estos valores garantizan que las aplicaciones SQL existentes están enlazadas por compatibilidad con el release anterior de forma predeterminada.

### Antes de empezar

1. Active nivel de función 500 o superior, como se describe en [Activación de Db2 12 nueva función en la migración \( Db2 Instalación y migración\)](#).
2. Para cualquier paquete que necesite seguir ejecutándose en un nivel inferior, vincúlelo o vuelva a vincularlo y especifique explícitamente la opción de vinculación APPLCOMPAT. Para obtener más información, consulte [Controlar el nivel de compatibilidad de la aplicación Db2 \( Db2 for z/OS ¿Qué hay de nuevo?\)](#).
3. Tome las siguientes precauciones para asegurarse de que las aplicaciones estén listas para ejecutarse en el nivel más alto de compatibilidad de aplicaciones de forma predeterminada.
  - Identificar y resolver todas las incompatibilidades de aplicación del nivel superior, como se describe en “[Gestión de incompatibilidades de aplicaciones](#)” en la página 888.

- Vuelva a vincular los paquetes que deban seguir ejecutándose en el nivel de compatibilidad de aplicaciones inferior y especifique explícitamente la opción de vinculación APPLCOMPAT para ese nivel.

## Acerca de esta tarea

Una vez que todas las aplicaciones estén listas para ejecutarse en un nivel de compatibilidad de aplicaciones superior o vinculadas explícitamente en un nivel inferior, puede aumentar el valor del parámetro del subsistema APPLCOMPAT para vincular paquetes en un nivel de compatibilidad de aplicaciones superior de forma predeterminada.

El parámetro del subsistema APPLCOMPAT especifica el valor predeterminado que se utilizará cuando la opción de enlace APPLCOMPAT no se especifique en un comando BIND, o cuando el valor APPLCOMPAT no se especifique o no se almacene en el catálogo de enlaces ( Db2 ) para un comando REBIND. Su valor no impide que aplicaciones específicas se ejecuten en niveles de compatibilidad de aplicaciones más altos. Para obtener más información, consulte [APPL COMPAT LEVEL \(parámetro del subsistemaAPPLCOMPAT \) \( Db2 Instalación y migración\)](#).

## Procedimiento

Para habilitar la compatibilidad de la aplicación predeterminada con el nivel de función actual:

1. Cambiar la configuración del parámetro del subsistema APPLCOMPAT. Establezca el valor en V12R1M500 o *function-level* activo superior equivalente.

Puede completar este paso como se describe en [Actualización de los valores predeterminados de la aplicación y parámetros de subsistema \(Db2 Installation and Migration\)](#), o modificando su copia personalizada del trabajo DSNTIJUZ.

El formato es *VvvRrMmm*, donde *vv* es la versión, *r* es el release y *mmm* es el nivel de modificación. Por ejemplo, V12R1M510 identifica nivel de función 510. Para obtener una lista de todos los niveles de función disponibles en Db2 12, consulte [Db2 12Niveles de función \(¿Qué hay de nuevo Db2 for z/OS?\)](#). Consulte los detalles de activación para cada nivel de función para obtener un resumen de las nuevas características controladas por el nivel de compatibilidad de aplicación correspondiente.

2. Ejecute los dos primeros pasos de DSNTIJUZ para reconstruir el módulo de parámetros del subsistema (*DSNZPxxx* ).
3. Utilice el comando -SET SYSPARM o reinicie Db2.

## Resultados

Las futuras operaciones de enlace y reenlace establecen el nivel de compatibilidad de la aplicación del paquete en el valor del parámetro del subsistema APPLCOMPAT, si no se especifica la opción de enlace APPLCOMPAT. Los paquetes que se vinculan o se vuelven a vincular en el nivel superior pueden comenzar a utilizar las capacidades de SQL introducidas en ese nivel.

### Conceptos relacionados

[Niveles de compatibilidad de aplicaciones en Db2 12](#)

*El nivel de compatibilidad* de sus aplicaciones controla la adopción y el uso de nuevas capacidades y mejoras, y a veces reduce el impacto de los cambios incompatibles. La ventaja es que puede completar el proceso de migración de Db2 12 sin necesidad de actualizar sus aplicaciones inmediatamente.

### Tareas relacionadas

[Adoptar nuevas capacidades en Db2 12 entrega continua \( Db2 for z/OS ¿Qué hay de nuevo?\)](#)

[Activación de Db2 12 nueva función en la migración \( Db2 Instalación y migración\)](#)

### Referencia relacionada

[APPL COMPAT LEVEL \(parámetro del subsistemaAPPLCOMPAT \) \( Db2 Instalación y migración\)](#)

[APPLCOMPAT opción bind \(comandos Db2 \)](#)

# Capítulo 9. Preparación de una aplicación para su ejecución en Db2 for z/OS

Para preparar y ejecutar aplicaciones que contengan sentencias SQL estáticas incrustadas o sentencias SQL dinámicas, debe procesar, compilar, editar vínculos y vincular las sentencias SQL.

## Antes de empezar

Para evitar tener que volver a trabajar en el pedido, siga estos pasos:

1. Pruebe sus sentencias SQL utilizando SPUFI.
2. Compile su programa sin instrucciones SQL y resuelva todos los errores del compilador.
3. Proceda con la preparación y el precompilador de Db2 o con el compilador host que admite ese coprocesador de Db2 .

Los siguientes tipos de aplicaciones requieren diferentes métodos de preparación del programa:

- Aplicaciones que contienen llamadas ODBC
- Aplicaciones en lenguajes interpretados, como REXX. Para obtener información sobre la ejecución de programas REXX, que no prepara para su ejecución, consulte “[Ejecutar una aplicación REXX de e Db2](#)” en la página 1004.
- Aplicaciones Java, que pueden contener llamadas JDBC o sentencias de SQL incorporado

## Acerca de esta tarea

Antes de poder ejecutar un programa de aplicación en Db2 for z/OS, necesita prepararlo. Para preparar el programa, se crea un módulo de carga, posiblemente uno o más paquetes, y un plan de aplicación.

Si su programa de aplicación incluye sentencias SQL, debe procesar dichas sentencias SQL utilizando el Db2 coprocessor que se proporciona con un compilador o el Db2 precompiler.

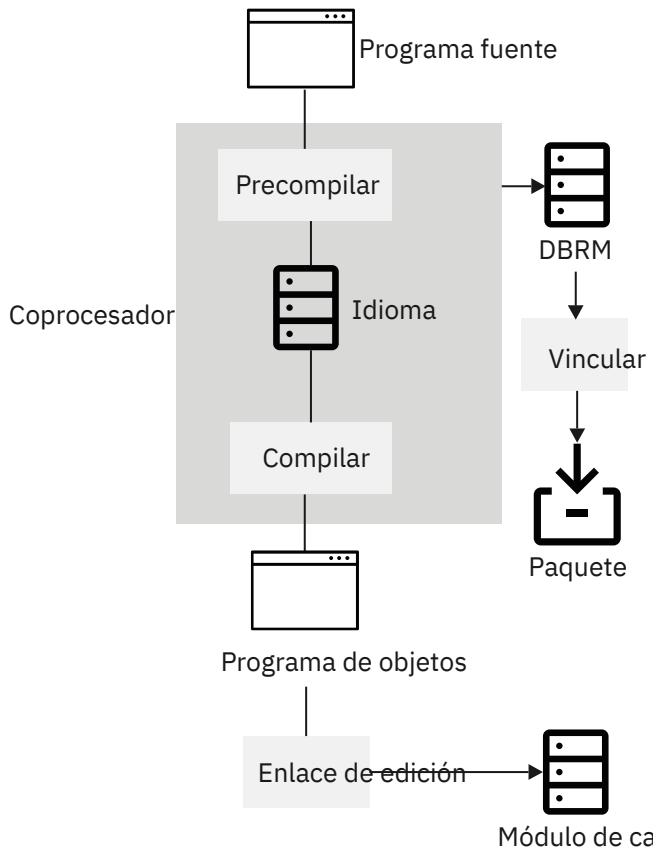
**Consejo:** Db2 coprocessor » es el método recomendado para procesar instrucciones SQL en programas de aplicación. En comparación con el Db2 precompiler, el Db2 coprocessor tiene menos restricciones en los programas SQL y es más compatible con las últimas mejoras de SQL y de los lenguajes de programación. Consulte el “[Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor](#)” en la página 897.

Tanto el Db2 coprocessor como el Db2 precompiler realizan las siguientes acciones:

- Sustituya las instrucciones SQL en sus programas fuente por llamadas a módulos de interfaz de lenguaje Db2
- Crear un módulo de solicitud de base de datos (DBRM), que comunique sus solicitudes SQL a Db2 durante el proceso de enlace

### Db2 coprocessor

La siguiente figura ilustra el proceso de preparación del programa cuando se utiliza el Db2 coprocessor. El proceso es similar al proceso con el Db2 precompiler, excepto que el Db2 coprocessor no crea una fuente modificada para su programa de aplicación. Para obtener más información, consulte “[Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor](#)” en la página 897.



*Figura 43. Visión general del proceso de preparación de un programa para aplicaciones que contienen SQL incorporado. El coprocesador de Db2 puede combinar los pasos de precompilación y compilación para determinados lenguajes.*

### Db2 precompiler

Después de procesar las sentencias SQL en su programa fuente mediante el precompilador Db2 , cree un módulo de carga, posiblemente uno o más paquetes, y un plan de aplicación. La creación de un módulo de carga implica compilar el código fuente modificado que produce el precompilador en un programa objeto, y editar el enlace del programa objeto para crear un módulo de carga. La creación de un paquete o un plan de aplicación, un proceso exclusivo de Db2, implica la vinculación de uno o más DBRM, que son creados por el Db2 precompiler, mediante el comando BIND PACKAGE. Para obtener más información, consulte “[Procesamiento de sentencias SQL mediante el uso del Db2 precompiler](#)” en la página 901.

### Procedimiento

- Complete las tareas utilizando uno de los métodos que se describen a continuación:
  - a) [“Procesamiento de instrucciones SQL para la preparación de programas” en la página 896](#)
  - b) [“Compilación y edición de enlaces de una aplicación” en la página 926](#)
  - c) [“Enlace de planes y paquetes de aplicación” en la página 927](#)
  - d) [Capítulo 10, “Ejecutar una aplicación en Db2 for z/OS”, en la página 1001](#)

No es necesario adjuntar un paquete en todos los casos. Estas instrucciones asumen que agrupa algunos de sus DBRM en paquetes e incluye una lista de paquetes en su plan.

Si utiliza CICS, es posible que tenga que completar pasos adicionales. Para obtener más información, consulte:

- [“Traducción de instrucciones de nivel de comando en un CICS programa” en la página 911](#)
- [“Ejemplo de invocación de aplicaciones en un procedimiento de mandatos” en la página 1013](#)

Puede utilizar los siguientes métodos para completar las tareas de preparación del programa:

- **Preparación de aplicaciones mediante procedimientos JCL**

Existen varios métodos disponibles para preparar una aplicación para ejecutarse. Puede:

- Utilice los paneles interactivos de Db2 (DB2I), que le guiarán paso a paso a través del proceso de preparación.
- Enviar un trabajo en segundo plano utilizando JCL (que los paneles de preparación del programa pueden crear para usted).
- Inicie el DSNH CLIST en primer plano o en segundo plano en TSO.
- Utilizar los prompters TSO y el procesador de comandos DSN.
- Utilice los procedimientos JCL añadidos a su archivo de configuración de Internet ( SYS1.PROCLIB ) (o equivalente) en el momento de la instalación de Db2 .
- Puede invocar el coprocesador desde los servicios del sistema UNIX. Si el DBRM se genera en un archivo HFS, también puede utilizar el Db2 command line processor para vincular el DBRM resultante. Opcionalmente, también puede copiar el DBRM en un miembro de conjunto de datos particionado utilizando los comandos opout y oget y, a continuación, enlazarlo utilizando JCL convencional.

Este tema describe cómo utilizar los procedimientos JCL para preparar un programa. Para obtener información sobre el uso de los paneles de DB2I, consulte Capítulo 9, “[Preparación de una aplicación para su ejecución en Db2 for z/OS](#)”, en la página 891.

- **Preparación de solicitudes por el Programa de Becas de la Fundación Tiffany ( Db2 ) Paneles de preparación**

Si desarrolla programas utilizando TSO y ISPF, puede prepararlos para su ejecución utilizando los paneles de preparación de programas de Db2 . Estos paneles le guiarán paso a paso a través del proceso de preparación de su solicitud para su ejecución. Hay otras formas de preparar un programa para ejecutarlo, pero usar Db2 Interactive (DB2I) es la más fácil porque te lleva automáticamente de una tarea a otra.

**Importante:** Si su programa C++ cumple las dos condiciones siguientes, debe utilizar un procedimiento JCL para prepararlo:

- El programa consta de más de un conjunto de datos o miembro.
- Más de un conjunto de datos o miembro contiene sentencias SQL.

Para preparar una solicitud mediante el Programa de Preparación de Paneles de la Asociación de la Industria de la Publicidad Digital ( Db2 ):

1. Si desea mostrar u ocultar los ID de los mensajes durante la preparación del programa, especifique uno de los siguientes comandos en la línea de comandos de ISPF :

**TSO PROFILE MSGID**

Se muestran los ID de los mensajes

**TSO PROFILE NOMSGID**

Los ID de mensajes están suprimidos

2. Abra el menú de opciones principales de DB2I.
3. Seleccione la opción que corresponda al panel de Preparación del programa.
4. Completar el panel de Preparación del programa y cualquier panel posterior. Después de completar cada panel, DB2I muestra automáticamente el siguiente panel apropiado.

- **Directrices de preparación para programas de lotes DL/I**

Siga las siguientes pautas cuando prepare un programa para acceder a Db2 y DL/I en un programa por lotes:

- [“Procesamiento de sentencias SQL mediante el uso del Db2 precompiler” en la página 901](#)
- [“Enlazar un programa por lotes” en la página 941](#)

- “[Compilación y edición de enlaces de una aplicación](#)” en la página 926
- “[Carga y ejecución de un programa por lotes](#)” en la página 1007

### **Conceptos relacionados**

[Db2 command line processor , \( Db2 Commands\)](#)

[Recurso de conexión de TSO \(Introducción a Db2 para z/OS\)](#)

### **Referencia relacionada**

[El menú de la opción primaria de DB2I \(Introducción a Db2 for z/OS\)](#)

[Procedimiento de comando DSNH \(TSO CLIST\) \(Comandos de Db2 \)](#)

## **Configuración de los valores predeterminados de DB2I**

Cuando utilice los paneles interactivos de Db2 ( DB2I ) para preparar una solicitud, puede especificar los valores predeterminados que debe utilizar DB2I. Estos valores predeterminados pueden incluir el idioma predeterminado de la aplicación y la instrucción JCL JOB predeterminada. De lo contrario, DB2I utiliza los valores predeterminados del sistema que se establecieron en el momento de la instalación.

### **Procedimiento**

A medida que DB2I le guía a través de una serie de paneles, introduzca los valores predeterminados que desea en los siguientes paneles cuando se muestren.

*Tabla 136. DB2I paneles que se utilizarán para establecer valores predeterminados*

<b>Si desea establecer los siguientes valores predeterminados...</b>	<b>Utilizar este panel</b>
<ul style="list-style-type: none"> <li>• ID de subsistema</li> <li>• número de veces adicionales para intentar conectarse a Db2</li> <li>• Lenguaje de programación</li> <li>• número de líneas en cada página de listado o salida SPUFI</li> <li>• nivel más bajo de mensaje para devolverte durante la fase BIND</li> <li>• Delimitador de cadena SQL para programas COBOL</li> <li>• cómo representar los separadores decimales</li> <li>• valor más pequeño del código de retorno (de precompilación, compilación, edición de enlaces o enlace) que impide la ejecución de pasos posteriores</li> <li>• número predeterminado de filas de entrada que se generan en la pantalla inicial de los paneles de ISPF</li> <li>• iD de usuario para asociar con la conexión de confianza para la sesión actual de DB2I</li> </ul>	DB2I Valores predeterminados Panel 1 panel
<ul style="list-style-type: none"> <li>• declaración predeterminada de JOB</li> <li>• símbolo utilizado para delimitar una cadena en una instrucción COBOL en una aplicación COBOL</li> <li>• si DCLGEN genera una cláusula de imagen que tiene la forma PIC G(n) DISPLAY-1 o PIC N(n).</li> </ul>	DB2I Valores predeterminados Panel de 2 paneles

Tabla 136. DB2I paneles que se utilizarán para establecer valores predeterminados (continuación)

Si desea establecer los siguientes valores predeterminados...	Utilizar este panel
Las siguientes características del paquete y del plan <ul style="list-style-type: none"><li>• nivel de aislamiento</li><li>• si se debe comprobar la autorización en tiempo de ejecución o en tiempo de enlace</li><li>• cuándo desbloquear recursos</li><li>• si obtener EXPLAIN información sobre cómo se ejecutan las sentencias SQL en el plan o paquete</li><li>• si necesita la moneda de datos para cursores ambiguos abiertos en ubicaciones remotas</li><li>• si se debe utilizar el procesamiento en paralelo</li><li>• si Db2 determina las rutas de acceso en el momento de la vinculación y de nuevo en el momento de la ejecución</li><li>• si aplazar la preparación de sentencias SQL dinámicas</li><li>• si Db2 mantiene sentencias SQL dinámicas después de los puntos de confirmación</li><li>• el esquema de codificación de la aplicación</li><li>• si desea utilizar sugerencias de optimización para determinar las rutas de acceso</li><li>• cuando Db2 escribe los cambios para las páginas actualizadas dependientes del grupo de búferes</li><li>• si se aplican reglas de tiempo de ejecución (RUN) o de tiempo de enlace (BIND) a las sentencias SQL dinámicas en tiempo de ejecución</li><li>• si continuar creando un paquete después de encontrar errores SQL (solo paquetes)</li><li>• cuándo adquirir bloqueos en los recursos (solo planes)</li><li>• si una sentencia CONNECT (Tipo 2) se ejecuta de acuerdo con las reglas de la especificación de lenguaje de consulta de datos (Db2 ,Db2) o el estándar SQL (STD). (solo planes)</li><li>• qué conexiones remotas finalizan durante una confirmación o una reversión (solo planes)</li></ul>	Valores predeterminados para el panel del paquete de enlace Valores predeterminados para el panel de Plan de vinculación

### Referencia relacionada

#### [Panel de valores predeterminados 1 de DB2I](#)

El Panel 1 de valores predeterminados de DB2I le permite cambiar muchos de los valores predeterminados del sistema que se establecieron durante la instalación de Db2.

#### [Panel 2 de valores predeterminados de DB2I](#)

Después de pulsar Intro en el Panel 1 Predeterminados de DB2I, se visualiza el Panel 2 Predeterminados de DB2I. Si elige IBMCOB como lenguaje en el Panel 1 Predeterminados de DB2I, se visualizan tres campos. De lo contrario, solo se visualiza el primer campo.

#### [Paneles Valores predeterminados para BIND PACKAGE y Valores predeterminados REBIND PACKAGE](#)

Estos paneles DB2I le permiten cambiar los valores predeterminados para las opciones BIND PACKAGE y REBIND PACKAGE.

#### [Paneles Valores predeterminados para BIND PLAN y Valores predeterminados REBIND PLAN](#)

Estos paneles DB2I le permiten cambiar los valores predeterminados para las opciones BIND PLAN y REBIND PLAN.

# Procesamiento de instrucciones SQL para la preparación de programas

El primer paso en la preparación de una aplicación de SQL para su ejecución es procesar las sentencias SQL en el programa. Para procesar las declaraciones, utilice el Db2 coprocessor o el Db2 precompilador. Durante la realización de este paso, las sentencias de SQL se sustituyen por llamadas a módulos de interfaz de lenguaje de Db2 y se crea un módulo de solicitud de base de datos (DBRM).

## Antes de empezar

Asegúrese de que los lenguajes de programación de desarrollo de aplicaciones cumplen los requisitos mínimos enumerados en «Lenguajes de programación» en el *Directorio de programas de Db2 12*. Consulte [Programar directorios para Db2 12 \(Db2 for z/OS en IBM Documentation\)](#).

## Acerca de esta tarea

Dado que la mayoría de los compiladores no reconocen las sentencias SQL, puede evitar errores de compilación utilizando el comando Db2 coprocessor o el comando Db2 precompilador.

Puede utilizar el Db2 coprocessor para el idioma de destino. Cuando utilizas el Db2 coprocessor, el compilador (en lugar del Db2 precompilador) analiza el programa y devuelve el código fuente modificado. El Registro de Datos de la Empresa (Db2 coprocessor) también produce un DBRM.

**Consejo:** Db2 coprocessor » es el método recomendado para procesar instrucciones SQL en programas de aplicación. En comparación con el Db2 precompilador, el Db2 coprocessor tiene menos restricciones en los programas SQL y es más compatible con las últimas mejoras de SQL y de los lenguajes de programación. Consulte el [“Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor” en la página 897](#).

Db2 precompilador , escanea el programa y devuelve el código fuente modificado, que luego puede compilar y editar el enlace. El precompilador también produce un DBRM (módulo de solicitud de base de datos). Puede vincular este DBRM a un paquete utilizando el subcomando BIND. Cuando complete estos pasos, podrá ejecutar su aplicación Db2 .

### Db2 versión en el módulo DSNHDECP

Cuando procesa sentencias SQL en su programa, si la versión de DSNHDECP (Db2) es la versión predeterminada proporcionada por el sistema, Db2 emite una advertencia y el procesamiento continúa. En este caso, asegúrese de que la información en DSNHDECP que utiliza Db2 refleje con precisión su entorno.

## Procedimiento

Para procesar instrucciones SQL en programas de aplicación, utilice uno de los siguientes métodos:

- Invoque el "Db2 coprocessor" para el idioma de destino que está utilizando al compilar su programa. Puede utilizar el Db2 coprocessor con compiladores de host C, C++, COBOL y PL/I.

Para invocar el Db2 coprocessor, especifique la opción del compilador SQL seguida de sus subopciones, que son aquellas opciones que están definidas para el Db2 precompilador. Algunas opciones de Db2 precompilador se ignoran. También puede invocar el comando "Db2 coprocessor" de UNIX System Services en z/OS para generar un DBRM en un conjunto de datos particionado o en un archivo HFS.

Para obtener más información, consulte [“Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor” en la página 897](#).

- Utilice el Db2 precompilador antes de compilar su programa. Para obtener más información, consulte [“Procesamiento de sentencias SQL mediante el uso del Db2 precompilador” en la página 901](#).

Para aplicaciones ensambladoras o de "Fortran", utilice el "Db2 precompilador" para preparar las sentencias SQL.

## Resultados

El resultado principal del sistema de gestión de pedidos ( Db2 coprocessor , o Db2 precompiler ) es un módulo de solicitud de base de datos (DBRM). Sin embargo, el Db2 coprocessor o Db2 precompiler también produce declaraciones de origen modificadas, una lista de declaraciones de origen, una lista de declaraciones que hacen referencia a nombres de host y columnas, y diagnósticos. Para obtener más información, consulte “Salida del Db2 precompiler” en la página 908.

## Qué hacer a continuación

Si la aplicación contiene comandos de tipo " CICS® ", debe traducir el programa antes de compilarlo. Para obtener más información, consulte “Traducción de instrucciones de nivel de comando en un CICS programa” en la página 911.

### Conceptos relacionados

[Uso del precompilador C/C++ de DB2 \(XL C/C++ Programming Guide\)](#)

[Coprocesador de DB2 \(Enterprise COBOL for z/OS Programming Guide\)](#)

[Salida del Db2 precompiler](#)

El resultado principal del sistema de gestión de pedidos ( Db2 precompiler ) es un módulo de solicitud de base de datos (DBRM). Sin embargo, el Db2 precompiler también produce declaraciones de origen modificadas, una lista de declaraciones de origen, una lista de declaraciones que hacen referencia a nombres de host y columnas, y diagnósticos.

[Diferencias entre el coprocesador Db2 y el Db2 precompiler](#)

El Db2 coprocessor y el Db2 precompiler tienen diferencias arquitectónicas. No se puede cambiar de uno a otro sin tener en cuenta estas diferencias y ajustar el programa conforme a ellas.

[Programar directorios para Db2 12 \( Db2 for z/OS en IBM Documentation \)](#)

### Tareas relacionadas

[Conversión de las sentencias de nivel de mandatos en un programa CICS](#)

Puede traducir CICS las aplicaciones con el CICS traductor de lenguaje de comandos como parte del proceso de preparación del programa. CICS los traductores de lenguaje de comandos solo están disponibles para los lenguajes ensamblador, C, COBOL y PL/I.

### Referencia relacionada

[Enterprise COBOL for z/OS](#)

## Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor

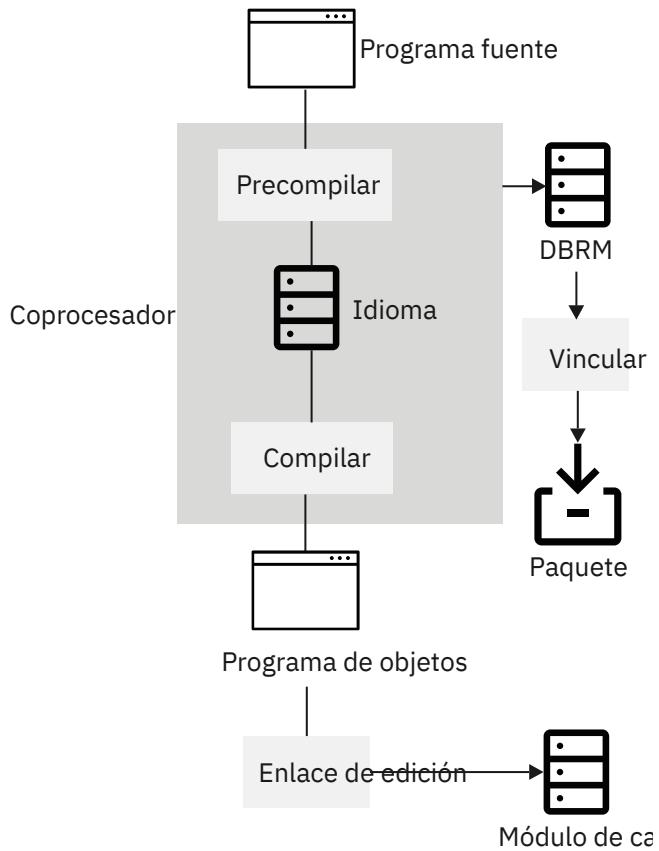
Puede utilizar el Db2 coprocessor para procesar instrucciones SQL en tiempo de compilación. Con el compilador de consultas de bases de datos ( Db2 coprocessor ), el compilador analiza un programa y copia todas las instrucciones SQL y la información de las variables del host en un módulo de solicitud de base de datos (DBRM). Db2 coprocessor » es el método recomendado para procesar instrucciones SQL en programas de aplicación. En comparación con el Db2 precompiler, el Db2 coprocessor tiene menos restricciones en los programas SQL y es más compatible con las últimas mejoras de SQL y de los lenguajes de programación.

### Antes de empezar

Asegúrese de que los lenguajes de programación de desarrollo de aplicaciones cumplen los requisitos mínimos enumerados en «Creación de aplicaciones mediante el uso de la biblioteca de herramientas de desarrollo de aplicaciones ( Db2 coprocessor)» en *el directorio de programas de Db2 12* . Consulte [Programar directorios para Db2 12 \( Db2 for z/OS en IBM Documentation \)](#).

### Acerca de esta tarea

Db2 coprocessor , procesa las sentencias SQL en tiempo de compilación.



*Figura 44. Visión general del proceso de preparación de un programa para aplicaciones que contienen SQL incorporado. El coprocesador de Db2 puede combinar los pasos de precompilación y compilación para determinados lenguajes.*

**Excepción:** Para PL/I, el preprocessador PL/I en lugar del compilador llama a la función Db2 coprocessor .

Db2 coprocessor » es el método recomendado para procesar instrucciones SQL en programas de aplicación. En comparación con el Db2 precompiler, el Db2 coprocessor tiene menos restricciones en los programas SQL y es más compatible con las últimas mejoras de SQL y de los lenguajes de programación.

Por ejemplo, cuando procesa instrucciones SQL con el Db2 coprocessor, puede hacer lo siguiente en su programa:

- Utilice nombres completos para las variables de host estructuradas.
- Incluir instrucciones SQL en cualquier nivel de un programa anidado, en lugar de solo en el archivo fuente de nivel superior.(Aunque puede incluir instrucciones SQL en cualquier nivel de un programa anidado, debe compilar todo el programa como una unidad)
- Utilizar sentencias SQL INCLUDE anidadas.
- Solo para programas en C o C++, escriba aplicaciones con formato de longitud variable.
- Solo para programas C o C++, utilice caracteres dependientes de la página de códigos, como corchetes izquierdos y derechos, sin utilizar notación trigráfica cuando los programas utilicen páginas de códigos diferentes.

## Procedimiento

Para procesar sentencias SQL utilizando el Db2 coprocessor, realice una de las siguientes acciones:

- Enviar un trabajo JCL para procesar esa instrucción SQL. Incluya la información siguiente:
  - Especifique la opción del compilador SQL cuando compile su programa:

La opción del compilador SQL indica que desea que el compilador invoque el Db2 coprocessor. Especifique una lista de opciones de procesamiento SQL entre paréntesis después de la palabra clave SQL. [Tabla 140 en la página 914](#) enumera las opciones que puede especificar.

Para COBOL y PL/I, incluya la lista de opciones de procesamiento SQL entre comillas simples o dobles. Para PL/I, separe las opciones en la lista con una coma, un espacio en blanco o ambos, como se muestra en los siguientes ejemplos:

<b>C/C++</b>	SQL(APOSTSQL STDSQL(NO))
<b>COBOL</b>	SQL("APOSTSQL STDSQL(NO)")
<b>PL/I</b>	PP(SQL("APOSTSQL,STDSQL(NO)"))

- Para los programas PL/I que utilizan tipos de datos BIGINT o LOB, especifique las siguientes opciones del compilador cuando compile su programa:

```
LIMITS(FIXEDBIN(63), FIXEDDEC(31))
```

- Si es necesario, aumente el tamaño de la región del usuario para que pueda acomodar más memoria para el Db2 coprocessor.
- Incluya las declaraciones DD para los siguientes conjuntos de datos en el JCL para su paso de compilación:

#### **Db2 cargar biblioteca (*prefijo.SDSNLOAD*)**

Db2 coprocessor , llama a los módulos de procesamiento de instrucciones SQL ( Db2 ). Por lo tanto, debe incluir el nombre del conjunto de datos de la biblioteca de carga e Db2 es en la concatenación STEPLIB para el paso del compilador.

#### **Biblioteca DBRM**

Db2 coprocessor , produce un DBRM. Los DBRM y la biblioteca DBRM se describen en [“Salida del Db2 coprocessor” en la página 901](#). Debe incluir una instrucción DD DBRMLIB que especifique el conjunto de datos de la biblioteca DBRM.

#### **Biblioteca para sentencias SQL INCLUDE**

Si su programa contiene sentencias SQL INCLUDE de *nombre-miembro* que especifican una entrada secundaria al programa fuente, también debe especificar el conjunto de datos para *nombre-miembro*. Incluya el nombre del conjunto de datos que contiene el *nombre de miembro* en la concatenación SYSLIB para el paso del compilador.

- Invoque el Db2 coprocessor desde z/OS UNIX System Services. Si invoca el DBRM ( Db2 coprocessor ) desde z/OS UNIX System Services, puede elegir que el DBRM se genere en un conjunto de datos particionado o en un archivo HFS.

Cuando invoque el Db2 coprocessor, especifique la opción del compilador SQL. La opción del compilador SQL indica que desea que el compilador invoque el Db2 coprocessor. Especifique una lista de opciones de procesamiento SQL entre paréntesis después de la palabra clave SQL. Para ver la lista de opciones que puede especificar, consulte [Opciones de procesamiento SQL](#).

El nombre de archivo para el DBRM se determina como se describe en [DRBMLIB](#). Para los lenguajes de programación distintos de C y C++, la opción DBRMLIB no es compatible y el nombre de archivo siempre se genera. Para C y C++, puede especificar uno de los siguientes elementos:

- El nombre de un conjunto de datos particionados. El siguiente ejemplo invoca el C/C++ Db2 coprocessor para compilar (con el compilador c89 ) un programa C de muestra y solicita que el DBRM resultante se almacene en el miembro de prueba del conjunto de datos `userid.dbrmlib.data` :

```
c89 -Wc,"sql,dbrmlib(/'userid.dbrmlib.data(test)'),langlvl(extended)" -c t.c
```

- El nombre de un archivo HFS. El nombre puede ser calificado, parcialmente calificado o no calificado. La ruta del archivo puede contener un máximo de 1024 caracteres, y el nombre del archivo puede contener un máximo de 255 caracteres. Los primeros 8 caracteres del nombre del archivo, sin incluir la extensión, deben ser únicos dentro del sistema de archivos.

Por ejemplo, supongamos que la estructura de su directorio es /u/USR001/c/example y que su directorio de trabajo actual es /u/USR001/c. La siguiente tabla muestra ejemplos de cómo especificar los nombres de archivo HFS con la opción DBRMLIB y cómo se resuelven los nombres de archivo.

*Tabla 137. Cómo especificar archivos HFS para almacenar DBRM*

Sí especifica...	El DBRM se genera en...
dbrmlib(/u/USR001/sample.dbm)	/u/USR001/sample.dbm
dbrmlib(example/sample.dbm)	/u/USR001/c/example/sample.dbm
dbrmlib(../sample.dbm)	/u/USR001/sample.dbm
dbrmlib(sample.dbm)	/u/USR001/c/sample.dbm

El siguiente ejemplo invoca el Db2 coprocessor para compilar (con el compilador c89) un programa C de muestra y solicita que el DBRM resultante se almacene en el archivo test.dbm en el directorio tmp:

```
c89 -Wc,"sql,dbrmlib(/tmp/test.dbm),langlvl(extended)" -c t.c
```

El siguiente ejemplo invoca el compilador COBOL (Db2 coprocessor) para compilar un programa COBOL de muestra con el compilador Enterprise COBOL para z/OS 6.2 o compiladores posteriores:

```
cob2 myprogram.cbl -c myprogram -dbrmlib -qsql
```

El siguiente ejemplo invoca el compilador Enterprise PL/I (Db2 coprocessor) para compilar un programa PL/I de muestra para compiladores Enterprise PL/I (Enterprise PL/I for z/OS 5.2) o posteriores:

```
pli -c -qpp=sql -qdbrmlib -qrent myprogram.pli
```

Si solicita que el DBRM se genere en un archivo HFS, puede vincular el DBRM resultante utilizando el comando BIND (vincular) de Db2 command line processor. Para obtener más información sobre el uso del comando BIND de Db2 command line processor, consulte [“Vinculación de un DBRM que está en un archivo HFS a un paquete o colección”](#) en la página 930. Opcionalmente, también puede copiar el DBRM en un miembro de conjunto de datos particionado utilizando los comandos oput y oget y, a continuación, enlazar el DBRM utilizando JCL convencional.

## Resultados

El resultado principal del sistema de gestión de pedidos (Db2 coprocessor) es un módulo de solicitud de base de datos (DBRM). Sin embargo, el Db2 coprocessor también produce declaraciones de origen modificadas, una lista de declaraciones de origen, una lista de declaraciones que hacen referencia a nombres de host y columnas, y diagnósticos. Para obtener más información, consulte [“Salida del Db2 coprocessor”](#) en la página 901.

## Soporte para compilar un programa COBOL que incluya SQL desde un programa ensamblador

El compilador COBOL proporciona una función que le permite invocar el compilador COBOL mediante un programa ensamblador.

Si tiene la intención de utilizar el coprocesador COBOL (Db2) e iniciar el compilador COBOL desde un programa ensamblador como parte de la preparación de su aplicación COBOL (Db2), puede utilizar la opción del compilador SQL y proporcionar el nombre DBRMLIB DD alternativo de la misma manera que puede especificar otros nombres DD alternativos. El coprocesador DBRM (Db2) crea el miembro DBRM de acuerdo con su biblioteca DBRM PDS y el miembro DBRM que especificó utilizando el nombre DD DBRMLIB alternativo.

## Referencia relacionada

[Inicio del compilador desde un programa assembler](#)

## Salida del Db2 coprocessor

El resultado del módulo de solicitud de base de datos (Db2 coprocessor, DBRM) es un módulo de solicitud de base de datos (DBRM).

**Consejo:** Db2 coprocessor » es el método recomendado para procesar instrucciones SQL en programas de aplicación. En comparación con el Db2 precompilador, el Db2 coprocessor tiene menos restricciones en los programas SQL y es más compatible con las últimas mejoras de SQL y de los lenguajes de programación. Consulte el “[Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor](#)” en la página 897.

Db2 coprocessor , produce un módulo de solicitud de base de datos (DBRM). El DBRM es un conjunto de datos que contiene las sentencias SQL y la información de las variables del host que se extraen del programa fuente, junto con la información que identifica el programa y vincula el DBRM a las sentencias fuente traducidas. El DBRM se convierte en la entrada del proceso de enlace.

El conjunto de datos requiere espacio para contener todas las sentencias SQL, además de espacio para cada nombre de variable de host y cierta información de encabezado. La información del encabezado por sí sola requiere aproximadamente dos registros para cada DBRM, 20 bytes para cada registro SQL y 6 bytes para cada variable de host.

Para ver el formato exacto del DBRM, consulte las macros de mapeo DBRM, DSNXDBRM y DSNXNBRM, en *el prefijo* de biblioteca.SDSNMACS. Los atributos DCB del conjunto de datos son RECFM FB, LRECL 80. El precompilador establece las características. Puede utilizar los comandos IEBCOPY, IEHPROGM, TSOCOPY y DELETE, u otras herramientas de gestión de PDS para mantener estos conjuntos de datos.

**Importante:** No modifique el contenido del DBRM. Si lo hace, pueden producirse resultados impredecibles. Db2 no admite DBRM modificados.

Todos los demás campos de caracteres en un DBRM utilizan EBCDIC. El marcador de versión actual (DBRMMRIC) en el encabezado de un DBRM se marca de acuerdo con la versión del precompilador, independientemente del valor de NEWFUN.

En un DBRM, las sentencias SQL y la lista de nombres de variables de host utilizan el esquema de codificación de caracteres UTF-8 (UTF-8 ).

## Procesamiento de sentencias SQL mediante el uso del Db2 precompilador

Db2 precompilador , escanea un programa y copia todas las sentencias SQL y la información de las variables del host en un módulo de solicitud de base de datos (DBRM). Db2 precompilador también devuelve el código fuente que se ha modificado para que las sentencias SQL no causen errores al compilar el programa.

### Antes de empezar

Asegúrese de que los lenguajes de programación de desarrollo de aplicaciones cumplan los requisitos mínimos enumerados en «[Creación de aplicaciones mediante el uso de la biblioteca de herramientas de desarrollo de aplicaciones \(Db2 precompilador\)](#)» en *el directorio de programas de Db2 12*. Consulte [Programar directorios para Db2 12 \(Db2 for z/OS en IBM Documentation\)](#).

**Consejo:** Db2 coprocessor » es el método recomendado para procesar instrucciones SQL en programas de aplicación. En comparación con el Db2 precompilador, el Db2 coprocessor tiene menos restricciones en los programas SQL y es más compatible con las últimas mejoras de SQL y de los lenguajes de programación. Consulte el “[Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor](#)” en la página 897.

Si utiliza el lenguaje de programación C (Db2 precompilador), asegúrese de que los nombres de las variables de host y de las matrices de variables de host sean únicos dentro del programa, incluso si

las variables y las matrices de variables se encuentran en bloques, clases, procedimientos, funciones o subrutinas diferentes. Puede calificar los nombres con un nombre de estructura para hacerlos únicos.

## Acerca de esta tarea

Después de copiar las instrucciones SQL y la información de la variable de host en un DBRM y de devolver el código fuente modificado, puede compilar y editar el enlace de este código fuente modificado.

La siguiente figura ilustra el proceso de preparación del programa cuando se utiliza el precompilador de Db2 . Después de procesar las sentencias SQL en su programa fuente utilizando el precompilador de Oracle ( Db2 ) , se crea un módulo de carga, posiblemente uno o más paquetes, y un plan de aplicación. La creación de un módulo de carga implica compilar el código fuente modificado que produce el precompilador en un programa objeto, y editar el enlace del programa objeto para crear un módulo de carga. La creación de un paquete o un plan de aplicación, un proceso exclusivo de Db2, implica la vinculación de uno o más DBRM, que son creados por el precompilador de Db2 , utilizando el comando BIND PACKAGE.

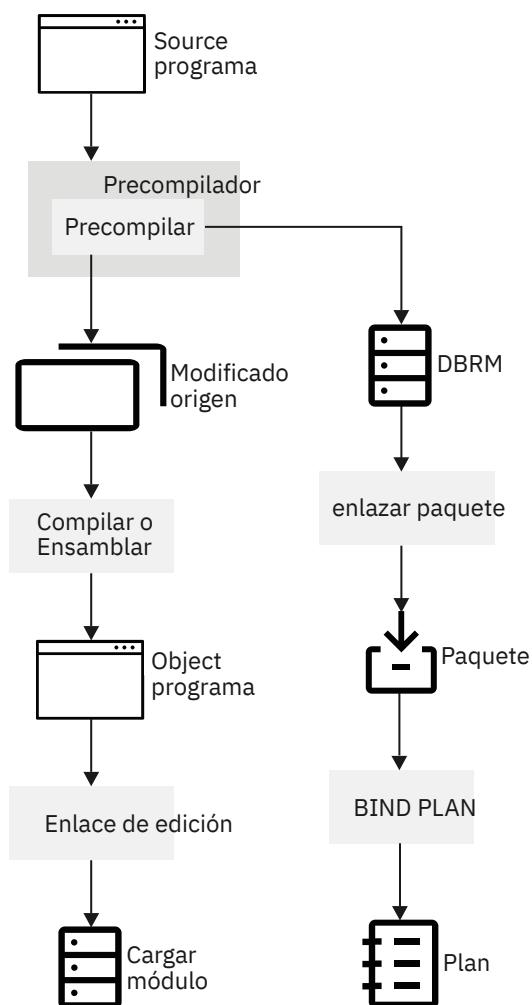


Figura 45. Preparación de programas con el precompilador de Db2

Antes de ejecutar el Db2 precompiler, utilice DCLGEN para obtener instrucciones SQL DECLARE TABLE precisas. Db2 precompiler comprueba las referencias de tabla y columna con las sentencias SQL DECLARE TABLE del programa, no con las tablas y columnas reales.

Db2 no tiene que estar activo cuando precompila tu programa.

No es necesario que precompiles el programa en el mismo subsistema de Db2 s en el que vinculas el DBRM y ejecutas el programa. Puede vincular un DBRM y ejecutarlo en un subsistema de Db2 en el nivel de versión anterior, si el programa original no utiliza ninguna propiedad de Db2 que sea exclusiva

de la versión actual. También puede ejecutar aplicaciones en la versión actual que anteriormente estaban vinculadas a subsistemas en el nivel de versión anterior.

## Procedimiento

Para procesar instrucciones SQL utilizando el lenguaje de consulta estructurado ( Db2 precompiler):

1. Asegúrese de que su programa está listo para ser procesado por el Servicio de Impuestos Internos ( Db2 precompiler ) realizando las siguientes acciones.

Para obtener información sobre los criterios de los programas que se pasan al Db2 precompiler, consulte [“Entrada al Db2 precompiler”](#) en la página 906.

2. Si planea ejecutar varios trabajos de precompilación y no está utilizando el conjunto de datos particionados extendido (PDSE) DFMSdfp, cambie el Db2 procedimientos de preparación del lenguaje (DSNHCOB, DSNHCOB2, DSNHICOB, DSNHFOR, DSNHC, DSNHPLI, DSNHASM, DSNHSQL) para especificar el parámetro DISP=OLD en lugar del parámetro DISP=SHR.

Los procedimientos de preparación de lenguaje de datos ( Db2 ) en el trabajo DSNTIJMV utilizan el parámetro DISP=OLD para reforzar la integridad de los datos. Sin embargo, el proceso de instalación convierte el parámetro DISP=OLD para el conjunto de datos de la biblioteca DBRM en DISP=SHR, lo que puede causar problemas de integridad de datos cuando se ejecutan varios trabajos de precompilación.

3. Inicie el proceso de precompilación utilizando uno de los siguientes métodos:

- DB2I paneles. Utilice el panel **Precompilar** o los paneles **de Preparación del programa** ( Db2 ). Para obtener detalles, consulte [“Paneles de DB2I que se utilizan para la preparación del programa”](#) en la página 965.
- El procedimiento de comando DSNH (un TSO CLIST). Para obtener detalles, consulte [Procedimiento de comando DSNH \(TSO CLIST\) \(Comandos de Db2\)](#).
- Procedimientos JCL que se suministran con Db2. Para obtener detalles, consulte [“Procedimientos JCL proporcionados por Db2 para la preparación de una aplicación”](#) en la página 962.

**Recomendación:** Especifique las opciones del precompilador SOURCE y XREF para obtener un diagnóstico completo del compilador de fuentes ( Db2 precompiler ). Este resultado es útil si necesita precompilar y compilar varias veces las instrucciones de origen del programa antes de que estén libres de errores y listas para la edición de enlaces.

## Resultados

El resultado principal del sistema de gestión de pedidos ( Db2 precompiler ) es un módulo de solicitud de base de datos (DBRM). Sin embargo, el Db2 precompiler también produce declaraciones de origen modificadas, una lista de declaraciones de origen, una lista de declaraciones que hacen referencia a nombres de host y columnas, y diagnósticos. Para obtener más información, consulte [“Salida del Db2 precompiler”](#) en la página 908.

## Qué hacer a continuación

### Preparación de un programa con extensiones orientadas a objetos mediante el uso de JCL

Si su programa C++ o Enterprise COBOL para z/OS cumple estas dos condiciones, necesita un JCL especial para prepararlo:

- El programa consta de más de un conjunto de datos o miembro.
- Más de un conjunto de datos o miembro contiene sentencias SQL.

Debe precompilar el contenido de cada conjunto de datos o miembro por separado, pero el preenlazador debe recibir toda la salida del compilador junta.

Procedimiento JCL DSNHCPP2, que se encuentra en el miembro DSNTIJMV del conjunto de datos DSN1210.SSDSNSAMP, te muestra una forma de hacerlo para C++.

## Precompilación de un programa por lotes

Cuando añada instrucciones SQL a un programa de aplicación, debe precompilar el programa de aplicación y vincular el DBRM resultante en un paquete, como se describe en [Capítulo 9, “Preparación de una aplicación para su ejecución en Db2 for z/OS”, en la página 891.](#)

### Conceptos relacionados

#### DCLGEN (generador de declaraciones)

El programa debe declarar las tablas y vistas a las que accede. El generador de declaraciones de Db2, DCLGEN, crea estas sentencias DECLARE para programas C, COBOL y PL/I programs, y así no necesita codificar las sentencias él mismo. DCLGEN genera también las estructuras de variable de lenguaje principal correspondientes.

#### Salida del Db2 precompiler

El resultado principal del sistema de gestión de pedidos ( Db2 precompiler ) es un módulo de solicitud de base de datos (DBRM). Sin embargo, el Db2 precompiler también produce declaraciones de origen modificadas, una lista de declaraciones de origen, una lista de declaraciones que hacen referencia a nombres de host y columnas, y diagnósticos.

### Referencia relacionada

[Procedimiento de comando DSNH \(TSO CLIST\) \(Comandos de Db2 \)](#)

## Conjuntos de datos que utiliza el precompilador

Cuando invoca el precompilador, necesita proporcionar conjuntos de datos que contienen entrada para el precompilador, como las sentencias de programación de host o sentencias SQL. También necesita proporcionar conjuntos de datos donde el precompilador pueda almacenar la salida, como el código de origen modificado o los mensajes de diagnóstico.

*Tabla 138. Declaraciones DD y conjuntos de datos que utiliza el precompilador de Db2*

Sentencia DD	Descripción del conjunto de datos	¿Necesario?
DBRMLIB	Conjunto de datos de salida, que contiene las sentencias SQL y la información de las variables de host que el precompilador de Db2 extrae del programa fuente. Se llama Módulo de solicitud de base de datos (DBRM). Este conjunto de datos se convierte en la entrada para el proceso de enlace de la interfaz de programación de aplicaciones ( Db2 , API). Los atributos DCB del conjunto de datos son RECFM FB, LRECL 80. DBRMLIB tiene que ser un PDS y debe especificarse un nombre de miembro. Puede utilizar los comandos IEBCOPY, IEHPROGM, TSO, COPY y DELETE, o las herramientas de gestión PDS para mantener el conjunto de datos.	Sí

Tabla 138. Declaraciones DD y conjuntos de datos que utiliza el precompilador de Db2 (continuación)

Sentencia DD	Descripción del conjunto de datos	¿Necesario?
STEPLIB	<p>Biblioteca de pasos para el paso de trabajo. En esta declaración DD, puede especificar el nombre de la biblioteca para el módulo de carga del precompilador, DSNHPC, y el nombre de la biblioteca para su miembro de programación de aplicaciones predeterminado (Db2), DSNHDECP.</p> <p><b>Recomendación:</b> Utilice siempre la instrucción STEPLIB DD para especificar la biblioteca donde reside su módulo DSNHDECP de Db2, a fin de garantizar que el precompilador de Db2 utilice los valores predeterminados adecuados de la aplicación. La biblioteca que contiene el módulo DSNHDECP de Db2 debe asignarse antes que la biblioteca prefix.SDSNLOAD.</p>	No, pero se recomienda
SYSCIN	Conjunto de datos de salida, que contiene la fuente modificada que escribe el precompilador de Db2. Este conjunto de datos se convierte en el conjunto de datos de entrada para el compilador o ensamblador. Este conjunto de datos debe tener los atributos RECFM F o FB, y LRECL 80. SYSCIN puede ser un PDS o un conjunto de datos secuenciales. Si se utiliza un PDS, debe especificarse el nombre del miembro.	Sí
SYSIN	Conjunto de datos de entrada, que contiene sentencias en el lenguaje de programación del host y sentencias SQL incrustadas. Este conjunto de datos debe tener los atributos RECFM F o FB, LRECL 80. SYSIN puede ser un PDS o un conjunto de datos secuenciales. Si se utiliza un PDS, debe especificarse el nombre del miembro.	Sí
SYSLIB	INCLUIR biblioteca, que contiene instrucciones adicionales de SQL y lenguaje de host. El precompilador de SQL INCLUDE (Db2) incluye el miembro o los miembros a los que hacen referencia las sentencias SQL INCLUDE en la entrada SYSIN de esta sentencia DD. Se pueden especificar varios conjuntos de datos, pero deben ser conjuntos de datos particionados con atributos RECFM F o FB, LRECL 80. Las sentencias SQL INCLUDE no se pueden anidar.	Nee
SYSPRINT	Conjunto de datos de salida, que contiene el listado de salida del precompilador de Db2. Este conjunto de datos debe tener un LRECL de 133 y un RECFM de FBA. SYSPRINT debe ser un conjunto de datos secuenciales	Sí

Tabla 138. Declaraciones DD y conjuntos de datos que utiliza el precompilador de Db2 (continuación)

Sentencia DD	Descripción del conjunto de datos	¿Necesario?
SISTEMA	Archivo de salida del terminal, que contiene mensajes de diagnóstico del precompilador de Db2 . Los atributos DCB del conjunto de datos están determinados por el sistema de control de datos ( z/OS ). SYSTERM debe ser un conjunto de datos secuenciales.	Nee
SYSUT1 y SYSUT2	Archivos de trabajo internos que el precompilador utiliza para almacenar información temporal a medida que convierte las sentencias SQL incrustadas en sentencias del lenguaje del host. La precompilación del ensamblador y del código fuente PL/I utiliza únicamente el conjunto de datos de la biblioteca de enlaces dinámicos ( SYSUT1 ). Los valores predeterminados del parámetro SPACE en los procedimientos de preparación de programas suministrados por Db2(DSNHASM, DSNHC, DSNHCPP, DSNHCPP2, DSNHICOB, DSNHPLI y DSNHFOR) son adecuados en la mayoría de los casos. Si su programa de aplicación contiene un gran número de instrucciones SQL incrustadas, es posible que tenga que aumentar esos valores.	No, a menos que necesite anular los valores predeterminados del parámetro SPACE.

#### Referencia relacionada

Parámetro ESPACIO (Referencia de JCL de MVS)

### Entrada al Db2 precompiler

La entrada primaria para el precompilador consta de sentencias en el lenguaje de programación host y sentencias SQL incorporadas.

Puede utilizar la instrucción SQL INCLUDE para obtener entrada secundaria de la biblioteca de inclusión, SYSLIB. La instrucción SQL INCLUDE lee la entrada del miembro especificado de SYSLIB hasta que llega al final del miembro.

Otro preprocessador, como el preprocessador de macros PL/I, puede generar instrucciones de código fuente para el precompilador. Cualquier preprocessador que se ejecute antes del precompilador debe ser capaz de transmitir instrucciones SQL. De manera similar, otros preprocessadores pueden procesar el código fuente, después de que usted precompila y antes de que compile o ensamble.

La entrada en el Db2 precompiler tiene las siguientes restricciones:

- El tamaño de un programa fuente que Db2 puede precompilar está limitado por el tamaño de la región y la memoria virtual disponible para el precompilador. Estas cantidades varían con cada instalación del sistema.
- Las formas de declaraciones de origen que pueden pasar a través del precompilador son limitadas. Por ejemplo, las constantes, los comentarios y otras sintaxis de código fuente que no son aceptadas por los compiladores del host (como una llave derecha que falta en C) pueden interferir con el análisis del código fuente del precompilador y causar errores. Para comprobar si hay declaraciones de origen inaceptables, ejecute el compilador del host antes que el precompilador. Puede ignorar los mensajes de error del compilador para las sentencias SQL o comentar las sentencias SQL. Una vez que las instrucciones de origen no contengan errores de compilación inaceptables, podrá descomentar las instrucciones SQL que haya comentado anteriormente y continuar con el proceso normal de preparación del programa de Db2 para ese lenguaje de programación.

- Debe escribir sentencias en el lenguaje de programación del host y sentencias SQL utilizando los mismos márgenes, tal y como se especifica en la opción MARGINS del precompilador.
- El conjunto de datos de entrada, SYSIN, debe tener los atributos RECFM F o FB, LRECL 80.
- SYSLIB debe ser un conjunto de datos particionado, con atributos RECFM F o FB, LRECL 80.
- La entrada de la biblioteca INCLUDE no puede contener otras sentencias INCLUDE del precompilador.

## **Iniciar el precompilador dinámicamente cuando se utilizan procedimientos JCL**

Puede llamar al precompilador desde un programa ensamblador utilizando una macro.

### **Acerca de esta tarea**

Puede llamar al precompilador desde un programa ensamblador utilizando una de las instrucciones de macro ATTACH, CALL, LINK o XCTL.

Para llamar al precompilador, especifique DSNHPC como nombre del punto de entrada. Puede pasar tres opciones de dirección al precompilador; los siguientes temas describen sus formatos. Las opciones son direcciones de:

- Una lista de opciones de precompilador
- Una lista de nombres DD alternativos para los conjuntos de datos que utiliza el precompilador
- Un número de página para utilizar en la primera página del listado del compilador en SYSPRINT

### **Referencia relacionada**

[Uso de X-macros\(MVS Assembler Services Reference\)](#)

### **Formato de lista de opciones del precompilador**

Cuando llama al precompilador, puede especificar un número de opciones, en una lista, para el proceso de sentencias SQL. Es necesario especificar esta lista de opciones en un formato concreto.

La lista de opciones debe comenzar en un límite de 2 bytes. Los dos primeros bytes contienen un recuento binario del número de bytes de la lista (excluido el campo de recuento). El resto de la lista es EBCDIC y puede contener palabras clave de opciones de precompilador, separadas por uno o más espacios en blanco, una coma o ambos.

### **Formato de lista de nombres DD**

Cuando llame al precompilador, puede especificar una lista de nombres DD alternativos para los conjuntos de datos que utiliza el precompilador. Debe especificar esta lista en un formato concreto.

La lista de nombres DD debe comenzar en un límite de 2 bytes. Los dos primeros bytes contienen un recuento binario del número de bytes de la lista (excluido el campo de recuento). Cada entrada de la lista es un campo de 8 bytes, justificado a la izquierda y relleno con espacios en blanco si es necesario.

La siguiente tabla muestra la secuencia de entradas:

*Tabla 139. Entradas de la lista DDNAME*

<b>Entrada</b>	<b>Nombre de dominio estándar</b>	<b>Uso</b>
1	No aplicable	
2	No aplicable	
3	No aplicable	
4	SYSLIB	Entrada de la biblioteca
5	SYSIN	Entrada de origen
6	SYSPRINT	Listado de diagnóstico

Tabla 139. Entradas de la lista DDNAME (continuación)

Entrada	Nombre de dominio estándar	Uso
7	No aplicable	
8	SYSUT1	Datos laborales
9	SYSUT2	Datos laborales
10	No aplicable	
11	No aplicable	
12	SISTEMA	Listado de diagnóstico
13	No aplicable	
14	SYSCIN	Salida de fuente modificada
15	No aplicable	
16	DBRMLIB	Salida DBRM

### Formato de número de página

Cuando llame al precompilador, puede especificar un número de página para utilizar en la primera página del listado del compilador en SYSPRINT. Debe especificar este número de página en un formato concreto.

Un campo de 6 bytes que comienza en un límite de 2 bytes contiene el número de página. Los dos primeros bytes deben contener el valor binario 4 (la longitud del resto del campo). Los últimos 4 bytes contienen el número de página en formato de caracteres o decimal dividido en zonas.

El precompilador añade 1 al último número de página utilizado en el listado del precompilador y pone este valor en el campo de número de página antes de devolver el control a la rutina de llamada. Por lo tanto, si vuelve a llamar al precompilador, la numeración de las páginas es continua.

## Salida del Db2 precompiler

El resultado principal del sistema de gestión de pedidos ( Db2 precompiler ) es un módulo de solicitud de base de datos (DBRM). Sin embargo, el Db2 precompiler también produce declaraciones de origen modificadas, una lista de declaraciones de origen, una lista de declaraciones que hacen referencia a nombres de host y columnas, y diagnósticos.

**Consejo:** Db2 coprocessor » es el método recomendado para procesar instrucciones SQL en programas de aplicación. En comparación con el Db2 precompiler, el Db2 coprocessor tiene menos restricciones en los programas SQL y es más compatible con las últimas mejoras de SQL y de los lenguajes de programación. Consulte el ["Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor"](#) en la página 897.

En concreto, el Db2 coprocessor o Db2 precompiler produce los siguientes tipos de resultados:

### Salida de listado

El precompilador de Db2 escribe la siguiente información en el conjunto de datos SYSPRINT:

- Listado de código fuente del precompilador

Si se especifica la opción SOURCE del precompilador Db2 , se genera un listado de fuentes. El listado de fuentes incluye instrucciones de fuente del precompilador, con números de línea que son asignados por el precompilador.

- Diagnóstico del precompilador

El precompilador produce mensajes de diagnóstico que incluyen números de línea del precompilador de las sentencias que contienen errores.

- Lista de referencias cruzadas del precompilador

Si se especifica la opción XREF del precompilador Db2 , se genera una lista de referencias cruzadas. La lista de referencias cruzadas muestra los números de línea del precompilador de las sentencias SQL que hacen referencia a nombres de host y columnas.

El conjunto de datos SYSPRINT tiene un LRECL de 133 y un RECFM de FBA. Este conjunto de datos utiliza el CCSID del programa fuente. Los números de declaración en la salida del listado del precompilador se muestran tal y como aparecen en el listado.

### Diagnóstico de terminales

Si existe un archivo de salida de terminal, SYSTEM, el precompilador Db2 escribe mensajes de diagnóstico en él. Una parte de la declaración fuente acompaña a los mensajes de este archivo. A menudo puede utilizar el archivo SYSTEM en lugar del archivo SYSPRINT para encontrar errores. Este conjunto de datos utiliza EBCDIC.

### Declaraciones de origen modificadas

El precompilador de C ( Db2 ) escribe las instrucciones de origen que procesa en SYSCIN, el conjunto de datos de entrada para el compilador o ensamblador. Este conjunto de datos debe tener los atributos RECFM F o FB, y LRECL 80. El código fuente modificado contiene llamadas a la interfaz de lenguaje de programación ( Db2 ). Las sentencias SQL que las llamadas reemplazan aparecen como comentarios. Este conjunto de datos utiliza el CCSID del programa fuente.

### Módulos de solicitud de bases de datos

El módulo de solicitud de base de datos (DBRM) es un conjunto de datos que contiene las sentencias SQL y la información de las variables del host que se extraen del programa fuente, junto con la información que identifica el programa y vincula el DBRM a las sentencias fuente traducidas. Se convierte en la entrada del proceso de encuadernación.

El conjunto de datos requiere espacio para contener todas las sentencias SQL, además de espacio para cada nombre de variable de host y cierta información de encabezado. La información del encabezado por sí sola requiere aproximadamente dos registros para cada DBRM, 20 bytes para cada registro SQL y 6 bytes para cada variable de host.

Para ver el formato exacto del DBRM, consulte las macros de mapeo DBRM, DSNXDBRM y DSNXNBRM, en *el prefijo* de biblioteca.SDSNMACS. Los atributos DCB del conjunto de datos son RECFM FB, LRECL 80. El precompilador establece las características. Puede utilizar los comandos IEBCOPY, IEHPROGM, TSOCOPY y DELETE, u otras herramientas de gestión de PDS para mantener estos conjuntos de datos.

**Importante:** No modifique el contenido del DBRM. Si lo hace, pueden producirse resultados impredecibles. Db2 no admite DBRM modificados.

En un DBRM, las sentencias SQL y la lista de nombres de variables de host utilizan el esquema de codificación de caracteres UTF-8 ( UTF-8 ).

Todos los demás campos de caracteres en un DBRM utilizan EBCDIC. El marcador de versión actual (DBRMMRRC) en el encabezado de un DBRM se marca de acuerdo con la versión del precompilador, independientemente del valor de NEWFUN.

### Tareas relacionadas

#### Procesamiento de instrucciones SQL mediante el uso del Db2 coprocessor

Puede utilizar el Db2 coprocessor para procesar instrucciones SQL en tiempo de compilación. Con el compilador de consultas de bases de datos ( Db2 coprocessor ), el compilador analiza un programa y copia todas las instrucciones SQL y la información de las variables del host en un módulo de solicitud de base de datos (DBRM). Db2 coprocessor » es el método recomendado para procesar instrucciones SQL en programas de aplicación. En comparación con el Db2 precompiler, el Db2 coprocessor tiene menos restricciones en los programas SQL y es más compatible con las últimas mejoras de SQL y de los lenguajes de programación.

#### Procesamiento de instrucciones SQL mediante el uso del Db2 precompiler

Db2 precompiler , escanea un programa y copia todas las sentencias SQL y la información de las variables del host en un módulo de solicitud de base de datos (DBRM). Db2 precompiler también devuelve el código fuente que se ha modificado para que las sentencias SQL no causen errores al compilar el programa.

## Diferencias entre el coprocesador Db2 y el Db2 precompiler

El Db2 coprocessor y el Db2 precompiler tienen diferencias arquitectónicas. No se puede cambiar de uno a otro sin tener en cuenta estas diferencias y ajustar el programa conforme a ellas.

**Consejo:** Db2 coprocessor » es el método recomendado para procesar instrucciones SQL en programas de aplicación. En comparación con el Db2 precompiler, el Db2 coprocessor tiene menos restricciones en los programas SQL y es más compatible con las últimas mejoras de SQL y de los lenguajes de programación. Consulte el ["Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor"](#) en la página 897.

**Recomendación:** Utilice el compilador de variables Unicode ( Db2 coprocessor ) en lugar del precompilador cuando utilice variables Unicode en aplicaciones COBOL o PL/I.

Dependiendo de si utiliza el Db2 coprocessor , que se recomienda en la mayoría de los casos, o el Db2 precompiler, asegúrese de tener en cuenta las siguientes diferencias:

- Diferencias en el manejo de los CCSID de origen:

Tanto Db2 coprocessor como Db2 precompiler convierten las sentencias SQL de su programa fuente a UTF-8 para su análisis.

El Db2 coprocessor o Db2 precompiler utiliza el valor de origen *CCSID(n)* para convertir de ese CCSID a CCSID 1208 ( UTF-8 ). El valor CCSID debe ser un CCSID EBCDIC. Si desea preparar un programa fuente que esté escrito en un CCSID que no se pueda convertir directamente a o desde CCSID 1208, debe crear una conversión indirecta.

- Diferencias en el manejo de los CCSID de variables de host:

– **COBOL:**

**Db2 coprocessor**

El compilador COBOL con soporte de caracteres nacionales siempre establece los CCSID para variables alfanuméricas, incluidas las variables de host que se utilizan en SQL, en el CCSID de origen. Alternativamente, puede especificar que desea que el COBOL Db2 coprocessor maneje los CCSID de la misma manera que el Db2 precompiler.

**Db2 precompiler:**

Db2 precompiler establece CCSID para variables de host alfanuméricas solo cuando el programa incluye una instrucción DECLARE:hv VARIABLE explícita.

**Recomendación:** Si tiene problemas con las variables de host CCSID, cambie su aplicación para incluir la instrucción DECLARE :hv VARIABLE para sobrescribir el CCSID especificado por el compilador COBOL, o utilice el comando Db2 precompiler.

Por ejemplo, supongamos que Db2 ha asignado una columna FOR BIT DATA a una variable de host de la siguiente manera:

```
01 hv1 pic x(5).
01 hv2 pic x(5).

EXEC SQL CREATE TABLE T1 (colwbit char(5) for bit data,
 rowid char(5)) END-EXEC.

EXEC SQL
INSERT INTO T1 VALUES (:hv1, :hv2)
END-EXEC.
```

**Db2 coprocessor: En el código fuente modificado del coprocesador de COBOL** ( Db2 ) con National Character Support, hv1 y hv2 se representan a Db2 de la siguiente manera, con CCSID: (Asumamos que el CCSID del código fuente es 1140)

```
for hv1 and hv2, the value for CCSID is set to '1140' ('474'x) in input SQLDA
of the INSERT statement.
```

```
'7F0000474000000007F'x
```

Para garantizar que no existe ninguna discrepancia entre la columna con FOR BIT DATA y la variable host con CCSID 1140, agregue la siguiente declaración para :hv1 o utilice el precompilador Db2 :

```
EXEC SQL DECLARE : hv1 VARIABLE FOR BIT DATA END-EXEC.

for hv1 declared with for bit data. The value in SQL---AVAR-NAME-DATA is
set to 'FFFF'x for CCSID instead of '474x'.

'7F0000FFFF00000007F'x <=> with DECLARE :hv1 VARIABLE FOR BIT DATA
vs.
'7F00000474000000007F'x <=> without
```

**Db2 precompilador:** En el código fuente modificado del precompilador Db2 , hv1 y hv2 se representan a Db2 a través de SQLDA de la siguiente manera, sin CCSID:

```
for hv1: NO CCSID

20 SQL-PVAR-NAMEL1 PIC S9(4) COMP-4 VALUE +0.
20 SQL-PVAR-NAMEC1 PIC X(30) VALUE ' '.

for hv2: NO CCSID

20 SQL-PVAR-NAMEL2 PIC S9(4) COMP-4 VALUE +0.
20 SQL-PVAR-NAMEC2 PIC X(30) VALUE ' '
```

## - PL/I

### **Db2 coprocessor:**

Puede especificar si los CCSID deben asociarse con variables de host utilizando las siguientes opciones del preprocesador PL/I SQL:

#### **CCSIDO**

Especifica que el preprocesador PL/I SQL no debe establecer los CCSID para todas las variables de host a menos que se definan con la sentencia SQL DECLARE :hv VARIABLE.

#### **NOCCSIDO**

Especifica que el preprocesador PL/I SQL debe establecer los CCSID para todas las variables de host.

### **Conceptos relacionados**

[z/OS Unicode Services User's Guide and Reference](#)

### **Referencia relacionada**

#### Descripciones de opciones de proceso de SQL

Puede especificar cualquier opción de proceso de SQL tanto si utiliza el precompilador de Db2 como si utiliza el coprocesador de Db2. Sin embargo, es probable que el coprocesador de Db2 omita ciertas opciones ya que existen opciones del compilador del lenguaje principal que proporcionan la misma información.

[Enterprise COBOL for z/OS](#)

[Opciones de preprocesador de SQL \(PL/I\) \(Enterprise PL/I for z/OS Programming Guide:\)](#)

## **Traducción de instrucciones de nivel de comando en un CICS programa**

Puede traducir CICS las aplicaciones con el CICS traductor de lenguaje de comandos como parte del proceso de preparación del programa. CICS los traductores de lenguaje de comandos solo están disponibles para los lenguajes ensamblador, C, COBOL y PL/I.

### **Procedimiento**

Prepare su CICS programa en cualquiera de estas secuencias:

Secuencia	Observaciones
a. <b>precompilador de Db2</b>	Esta secuencia es el método preferido de preparación de programas y el que admiten los paneles de preparación de programas de DB2I. Si utiliza los paneles de DB2I para la preparación del programa,

<b>Secuencia</b>	<b>Observaciones</b>
b. <b>CICS Comando Language Translator.</b>	puede especificar las opciones del traductor automáticamente, en lugar de tener que proporcionar una cadena de opciones por separado.
a. <b>CICS traductor de lenguaje de comandos</b> b. <b>precompilador de Db2</b>	Esta secuencia da lugar a un mensaje de advertencia del CICS traductor para cada instrucción EXEC SQL que encuentra. Los mensajes de advertencia no tienen ningún efecto en el resultado. Si utiliza conjuntos de caracteres de doble byte (DBCS), se recomienda la precompilación antes de la traducción, como se ha descrito anteriormente.

Utilice el precompilador Db2 antes que el CICS traductor para evitar que el precompilador confunda CICS la salida del traductor con datos gráficos.

Si su programa fuente está en COBOL, debe especificar un delimitador de cadena que sea el mismo para el precompilador de COBOL (Db2), el compilador de COBOL y el CICS traductor. Los valores predeterminados para el precompilador de COBOL y el precompilador de Db2 no son compatibles con el valor predeterminado para el CICS traductor.

Si las sentencias SQL de su programa fuente hacen referencia a variables de host que un puntero almacena en las CICS Direcciones TWA, debe hacer que las variables del host sean direccionables al TWA antes de ejecutar esas sentencias. Por ejemplo, una aplicación COBOL puede emitir la siguiente declaración para establecer la direccionabilidad al TWA:

```
EXEC CICS ADDRESS
 TWA (address-of-twa-area)
END-EXEC
```

Puede ejecutar CICS aplicaciones solo desde CICS espacios de direcciones. Esta restricción se aplica a la opción RUN en el segundo procesador de comandos DSN del programa. Todas esas posibilidades se dan en TSO.

Para preparar un programa de aplicación, puede añadir JCL de un trabajo creado por el programa Db2 Paneles de preparación para el JCL para el CICS traductor de lenguaje de comandos. Para ejecutar el programa preparado en CICS, es posible que tenga que definir programas y transacciones para CICS. El programador de su sistema debe realizar las CICS entradas de recursos o tablas.

*prefijo.SDSNSAMP* contiene ejemplos del JCL que se utiliza para preparar y ejecutar un CICS programa que incluye sentencias SQL. El conjunto de JCL incluye:

- Fase macro PL/I
- Db2 precompilación
- CICS Traducción de lenguaje de comandos
- Compilación de las declaraciones de origen del idioma anfitrión
- Edición de enlaces de la salida del compilador
- Vinculación del DBRM
- Ejecución de la aplicación preparada.

#### **Referencia relacionada**

[Aplicaciones de ejemplo en CICS](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

#### **Información relacionada**

[Definición de recursos \(CICS Transaction Server for z/OS\)](#)

## Opciones para el proceso de sentencias de SQL

Utilice las opciones de proceso de SQL para especificar cómo el precompilador de Db2 y el coprocesador de Db2 interpretan y procesan la entrada y cómo presentan la salida.

**Consejo:** Db2 coprocessor » es el método recomendado para procesar instrucciones SQL en programas de aplicación. En comparación con el Db2 precompiler, el Db2 coprocessor tiene menos restricciones en los programas SQL y es más compatible con las últimas mejoras de SQL y de los lenguajes de programación. Consulte el [“Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor” en la página 897](#).

### Db2 coprocessor

Si utiliza el Db2 coprocessor, especifique las opciones de procesamiento SQL de una de las siguientes maneras:

- Para C o C++, especifique las opciones como argumento de la opción del compilador SQL.
- Para COBOL, especifique las opciones como argumento de la opción del compilador SQL.
- Para PL/I, especifique las opciones como argumento del PP(SQL('opción,...')) opción del compilador.

Para ver ejemplos de cómo especificar las opciones de Db2 coprocessor, consulte [“Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor” en la página 897](#)

### Db2 precompiler

Si utiliza el Db2 precompiler, especifique las opciones de procesamiento SQL de una de las siguientes maneras:

- Con operandos DSNH
- Con la opción PARM de la sentencia EXEC JCL
- En paneles de DB2I

Db2 asigna valores predeterminados para cualquier opción de procesamiento SQL para la que no especifique explícitamente un valor. Esos valores predeterminados son los valores que se especifican en los paneles de instalación de VALORES PREDETERMINADOS DE PROGRAMACIÓN DE APLICACIONES.

### Descripciones de opciones de proceso de SQL

Puede especificar cualquier opción de proceso de SQL tanto si utiliza el precompilador de Db2 como si utiliza el coprocesador de Db2. Sin embargo, es probable que el coprocesador de Db2 omita ciertas opciones ya que existen opciones del compilador del lenguaje principal que proporcionan la misma información.

La siguiente tabla muestra las opciones que puede especificar cuando utiliza el precompilador Db2 o el coprocesador Db2 . La tabla también incluye abreviaturas para esas opciones e indica qué opciones se ignoran para un idioma de host concreto o por el coprocesador de la e Db2 . Esta tabla utiliza una barra vertical (|) para separar opciones mutuamente excluyentes, y corchetes ([ ]) para indicar que a veces se puede omitir la opción incluida.

Tabla 140. Opciones de procesamiento SQL

Opción de palabra clave	Significado
APOST1	<p>Indica que el precompilador de Db2 debe utilizar el apóstrofo ('') como delimitador de cadenas en las sentencias en el lenguaje del host que genera.</p> <p>Esta opción no está disponible en todos los idiomas.</p> <p>APOST y QUOTE son opciones mutuamente excluyentes. El valor predeterminado se encuentra en el campo STRING DELIMITER (DELIMITADOR DE CADENA) en el Panel 1 de valores predeterminados de programación de aplicaciones durante la instalación. Si STRING DELIMITER es el apóstrofo (''), APOST es el valor predeterminado.</p>
APOSTSQL	<p>Reconoce el apóstrofo ('') como delimitador de cadena y las comillas dobles ("") como carácter de escape SQL dentro de las sentencias SQL.</p> <p>APOSTSQL y QUOTESQL son opciones mutuamente excluyentes. El valor predeterminado se encuentra en el campo SQL STRING DELIMITER en el Panel 1 de valores predeterminados de programación de aplicaciones durante la instalación. Si SQL STRING DELIMITER es el apóstrofo (''), APOSTSQL es el valor predeterminado.</p>
ADJUNTAR (TSO CAF RRSAF  ULI)	<p>Especifica la función de conexión que utiliza la aplicación para acceder Db2. Las aplicaciones TSO, CAF, RRSAF o DSNULI que cargan la función de conexión pueden usar esta opción para especificar la función de conexión correcta, en lugar de codificar un punto de entrada DSNHLI ficticio.</p> <p>Puede especificar ATTACH(ULI) solo cuando utilice el coprocesador de memoria virtual (Db2).</p> <p>Esta opción no está disponible para las aplicaciones de Fortran.</p> <p>El valor predeterminado es ATTACH(TSO).</p>

Tabla 140. Opciones de procesamiento SQL (continuación)

Opción de palabra clave	Significado
CCSID (n)	<p>Especifica el valor numérico <i>n</i> del CCSID en el que está escrito el programa fuente. El número <i>n</i> debe ser un CCSID EBCDIC.</p> <p>La configuración predeterminada es el sistema EBCDIC CCSID, tal como se especifica en el panel DSNTIPF durante la instalación.</p> <p>El coprocesador de CCSID ( Db2 ) utiliza el siguiente proceso para determinar el CCSID de las sentencias fuente:</p> <ol style="list-style-type: none"> <li>1. Si el CCSID del programa fuente se especifica mediante una opción del compilador, como la opción de compilador COBOL CODEPAGE, el coprocesador de memoria virtual ( Db2 ) utiliza ese CCSID. Si también especifica la subopción CCSID de la opción del compilador SQL que es diferente de la opción del compilador CCSID, se devuelve una advertencia y no se utiliza el valor de la subopción CCSID.</li> <li>2. Si el CCSID no se especifica mediante una opción del compilador:             <ol style="list-style-type: none"> <li>a. Si se especifica la subopción CCSID de la opción del compilador SQL y contiene un CCSID EBCDIC válido, se utiliza ese CCSID.</li> <li>b. Si no se especifica la subopción CCSID de la opción del compilador SQL, y el compilador admite una opción para especificar el CCSID, como la opción del compilador COBOL CODEPAGE, se utiliza el valor predeterminado de la opción del compilador CCSID.</li> <li>c. Si no se especifica la subopción CCSID de la opción del compilador SQL, y el compilador no admite una opción para especificar el CCSID, se utiliza el CCSID predeterminado de DSNHDECP o un módulo predeterminado de la aplicación especificado por el usuario.</li> <li>d. Si se especifica la subopción CCSID de la opción SQL y contiene un CCSID no válido, la compilación finaliza.</li> </ol> </li> </ol> <p>CCSID sustituye a las opciones de procesamiento SQL GRÁFICO y NO GRÁFICO. Si especifica CCSID(1026) o CCSID(1155), el coprocesador de la biblioteca de control de sistemas de código ( Db2 ) no admite el punto de código «FC»X para las comillas dobles (").</p>
COMA	<p>Reconoce la coma (,) como el indicador de punto decimal en literales decimales o de punto flotante en los siguientes casos:</p> <ul style="list-style-type: none"> <li>• Para instrucciones SQL estáticas en programas COBOL</li> <li>• Para las sentencias SQL dinámicas, cuando el valor del parámetro de instalación DYNRULS es NO y el paquete o plan que contiene las sentencias SQL tiene un comportamiento de enlace, definición o invocación DYNAMICRULES.</li> </ul> <p>COMA y PUNTO son opciones mutuamente excluyentes. El valor predeterminado (COMA o PUNTO) se elige en DECIMAL POINT IS en el Panel 1 de Programación de Aplicaciones Predeterminadas durante la instalación.</p>

Tabla 140. Opciones de procesamiento SQL (continuación)

Opción de palabra clave	Significado
CONECTAR(2 1) CT(2 1)	Determina si se aplican las reglas de la declaración CONNECT de tipo 1 o de tipo 2.  CONNECT(2) Predeterminado: Aplicar reglas para la instrucción CONNECT (Tipo 2) CONNECT(1) Aplicar reglas para la sentencia CONNECT (Tipo 1)  Si no especifica la opción CONNECT cuando precompila un programa, se aplican las reglas de la sentencia CONNECT (Tipo 2).
FECHA (ISO EE. UU.   EUR   JIS   LOCAL)	Especifica que la salida de fecha siempre debe devolverse en un formato concreto, independientemente del formato que se especifique como ubicación predeterminada.  El valor predeterminado se especifica en el campo FORMATO DE FECHA en el Panel 2 de valores predeterminados de programación de aplicaciones durante la instalación.  El formato predeterminado viene determinado por los valores predeterminados de instalación del sistema en el que está vinculado el programa, no por los valores predeterminados de instalación del sistema en el que está precompilado el programa.  No puede utilizar la opción LOCAL a menos que tenga una rutina de salida de fecha.
DIC(15 31) DEC15   DEC31 D 15.s   D3 1.s	Especifica la precisión máxima para operaciones aritméticas decimales.  El valor predeterminado se encuentra en el campo ARITMÉTICA DECIMAL en el Panel 1 de valores predeterminados de programación de aplicaciones durante la instalación.  Si se especifica la forma D pp.s, pp debe ser 15 o 31, y s, que representa la escala mínima que se utilizará para la división, debe ser un número entre 1 y 9.
DEC P( <i>nombre</i> )	<i>name</i> representa el nombre de 1 a 8 caracteres de la aplicación por defecto del módulo de carga de solo datos que se va a utilizar.  Si se omite este parámetro, se utiliza el nombre predeterminado DSNHDECP.
FLAG(I W E S)1	Suprime los mensajes de diagnóstico por debajo del nivel de gravedad especificado (informativo, advertencia, error y error grave para los códigos de gravedad 0, 4, 8 y 12, respectivamente).  La configuración predeterminada es FLAG(I).
FLOAT ( S390 IEEE )	Determina si el contenido de las variables de host de punto flotante en programas ensamblador, C, C++ o PL/I está en formato de punto flotante IEEE o en formato de punto flotante hexadecimal de arquitectura z. Db2 ignora esta opción si el valor de HOST es distinto de ASM, C, CPP o PLI.  La configuración predeterminada es FLOAT( S390 ).

Tabla 140. Opciones de procesamiento SQL (continuación)

Opción de palabra clave	Significado
GRAPHIC	<p>Esta opción ya no se utiliza para el procesamiento de instrucciones SQL. Utilice la opción CCSID en su lugar.</p> <p>Indica que el código fuente podría utilizar datos mixtos, y que X'OE'y X'OF' son caracteres de control especiales (shift-out y shift-in) para datos EBCDIC.</p> <p>GRÁFICO y NO GRÁFICO son opciones mutuamente excluyentes. El valor predeterminado (GRÁFICO o NO GRÁFICO) se especifica en el campo DATOS MIXTOS del Panel 1 de valores predeterminados de programación de aplicaciones durante la instalación.</p>
HOST1(ASM  C[(FOLD)]  CPP[(FOLD)]  IBMCOB  PLI  FORTRAN  SQL  SQLPL)	<p>Define el lenguaje de programación que contiene las instrucciones SQL.</p> <p>Utilice IBMCOB para Enterprise COBOL para z/OS.</p> <p>Para C, especifique:</p> <ul style="list-style-type: none"> <li>• C si no desea que Db2 convierta las letras minúsculas de los identificadores ordinarios SBCS SQL a mayúsculas</li> <li>• C(FOLD) si quiere que Db2 convierta las letras minúsculas en mayúsculas en identificadores ordinarios SBCS SQL</li> </ul> <p>Para C++, especifique:</p> <ul style="list-style-type: none"> <li>• CPP si no desea que Db2 convierta las letras minúsculas en SBCS SQL los identificadores ordinarios en mayúsculas</li> <li>• CPP(FOLD) si quiere que Db2 ponga en mayúsculas las letras minúsculas en SBCS SQL los identificadores ordinarios</li> </ul> <p>Para el lenguaje de procedimientos SQL, especifique:</p> <ul style="list-style-type: none"> <li>• SQL, para realizar la comprobación de sintaxis y la conversión a un programa C generado para un procedimiento SQL externo.</li> <li>• SQLPL, para realizar la comprobación de sintaxis de un procedimiento SQL nativo.</li> </ul> <p>Si omite la opción HOST, el precompilador Db2 emite un mensaje de diagnóstico ( level-4 ) y utiliza el valor predeterminado para esta opción.</p> <p>El valor predeterminado se encuentra en el campo LANGUAGE DEFAULT (IDIOMA PREDETERMINADO) en el Panel 1 de valores predeterminados de programación de aplicaciones durante la instalación.</p> <p>Esta opción también establece los valores predeterminados dependientes del idioma.</p>
NIVEL[( <i>aaaa</i> )] L	<p>Define el nivel de un módulo, donde <i>aaaa</i> es cualquier valor alfanumérico de hasta siete caracteres. Esta opción no se recomienda para uso general, y los paneles DSNH CLIST y DB2I no la admiten.</p> <p>Para los ensambladores C, C++, Fortran, y PL/I, puede omitir la subopción (<i>aaaa</i>). El token de consistencia resultante está en blanco. Para COBOL, debe especificar la subopción.</p>
RECUENTO DE LÍNEAS <sup>1</sup> ( <i>n</i> ) CB	<p>Define el número de líneas por página que debe ser <i>n</i> para el listado del precompilador de Db2 . Esto incluye las líneas de encabezado que son insertadas por el precompilador Db2 . La configuración predeterminada es LINECOUNT(60).</p>

Tabla 140. Opciones de procesamiento SQL (continuación)

Opción de palabra clave	Significado
MAR GENES1(m,n[,c]) MAR.	Especifica qué parte de cada registro fuente contiene lenguaje de host o sentencias SQL. Para el ensamblador, esta opción también especifica dónde comienzan las continuaciones de columna. La primera opción (m) es la columna inicial para las declaraciones. La segunda opción (n) es la columna final para las declaraciones. La tercera opción (c) especifica dónde comienzan las continuaciones del ensamblador. De lo contrario, el precompilador de Db2 coloca un indicador de continuación en la columna inmediatamente posterior a la columna final. Los valores de margen pueden oscilar entre 1 y 80.  Los valores predeterminados dependen de la opción HOST que especifique.  Los paneles DSNH CLIST y DB2I no admiten esta opción. En ensamblador, la opción de margen debe coincidir con la instrucción ICTL, si se presenta en el código fuente.
NUEVO DIVERTIDO (Vn)	<b>Función en desuso:</b> La opción de procesamiento NEWFUN está obsoleta. Utilice la opción SQLLEVEL en su lugar.  Indica si se acepta la sintaxis de la función que es nueva para Db2 12.  <b>NEWFUN ( V12 )</b> Especifica que se permite cualquier sintaxis hasta Db2 12 . Este valor es equivalente al nivel de función V12R1M501. <b>NEWFUN ( V11 )</b> Especifica que se permite cualquier sintaxis hasta Db2 11 . <b>NEWFUN ( V10 )</b> Especifica que se permite cualquier sintaxis hasta DB2 10 . <b>NEWFUN ( V9 )</b> Especifica que se permite cualquier sintaxis hasta DB2 9 . DB2 9 es compatible, pero hace que el proceso de precompilación solo admita un nivel de función e DB2 9 . <b>NEWFUN ( V8 )</b> Especifica que se permite cualquier sintaxis hasta DB2 version 8 . V8 es compatible, pero hace que el proceso de precompilación solo admita un nivel de función e V8.  La opción NEWFUN se aplica únicamente al proceso de precompilación, ya sea por parte del precompilador o del coprocesador de la biblioteca de enlaces dinámicos ( Db2 ), independientemente de si se activan nuevas funciones en el subsistema. Usted es responsable de garantizar que vincula el DBRM resultante en un subsistema en el modo de migración correcto.

Tabla 140. Opciones de procesamiento SQL (continuación)

Opción de palabra clave	Significado
NO PARA	<p>En SQL estático, elimina la necesidad de la cláusula FOR UPDATE o FOR UPDATE OF en las sentencias DECLARE CURSOR. Cuando utiliza NOFOR, su programa puede realizar actualizaciones posicionadas en cualquier columna que el programa tenga autoridad para actualizar Db2 mente.</p> <p>Cuando no utilice NOFOR, si desea realizar actualizaciones posicionadas en cualquier columna que el programa tenga autoridad para actualizar Db2 , debe especificar FOR UPDATE sin lista de columnas en sus sentencias DECLARE CURSOR. La cláusula FOR UPDATE sin lista de columnas se aplica a sentencias SQL estáticas o dinámicas.</p> <p>Independientemente de si utiliza NOFOR, puede especificar FOR UPDATE OF con una lista de columnas para restringir las actualizaciones únicamente a las columnas que se nombran en la cláusula, y puede especificar la adquisición de bloqueos de actualización.</p> <p>Usted implica NOFOR cuando utiliza la opción STDSQL(YES).</p> <p>Si el DBRM resultante es muy grande, es posible que necesite almacenamiento adicional cuando especifique NOFOR o utilice la cláusula FOR UPDATE sin una lista de columnas.</p>
NOGRÁFICO	<p>Esta opción ya no se utiliza para el procesamiento de instrucciones SQL. Utilice la opción CCSID en su lugar.</p> <p>Indica el uso de X'OE'y X'OF' en una cadena, pero no como caracteres de control.</p> <p>GRÁFICO y NO GRÁFICO son opciones mutuamente excluyentes. El valor predeterminado (GRÁFICO o NO GRÁFICO) se especifica en el campo DATOS MIXTOS del Panel 1 de valores predeterminados de programación de aplicaciones durante la instalación.</p> <p>La opción NOGRÁFICA solo se aplica a datos EBCDIC.</p>
SIN OPCIONES NOOPTN	Suprime la lista de opciones del precompilador de Db2 .
NOPADNTSTR	<p>Indica que las variables de host de salida que son cadenas terminadas en NUL <b>no se rellenan</b> con espacios en blanco. Es decir, no se insertan espacios en blanco adicionales antes de que el terminador NUL se coloque al final de la cadena.</p> <p>PADNTSTR y NOPADNTSTR son opciones mutuamente excluyentes. El valor predeterminado (PADNTSTR o NOPADNTSTR) se especifica en el campo PAD NUL-TERMINATED en el Panel 2 de valores predeterminados de programación de aplicaciones durante la instalación.</p> <p>Esta opción solo se aplica a aplicaciones C y C++.</p>
NOSOURCE2 NOS	Suprime el listado de fuentes del precompilador Db2 . Este es el valor predeterminado.
NOXREF	Suprime la lista de referencias cruzadas del precompilador Db2 . Es el valor predeterminado.

Tabla 140. Opciones de procesamiento SQL (continuación)

Opción de palabra clave	Significado
ONEPASS ON	<p>Procesos en una sola pasada, para evitar el tiempo de procesamiento adicional que supondría hacer dos pasadas. Las declaraciones deben aparecer antes de las referencias SQL.</p> <p>Los valores predeterminados dependen de la opción HOST especificada.</p> <p>ONEPASS y TWOPASS son opciones mutuamente excluyentes.</p>
OPCIONES1 OPTN	Enumera las opciones del precompilador de Db2 . Este es el valor predeterminado.
PADNTSTR	<p>Indica que las variables de host de salida que son cadenas terminadas en NUL se llenan con espacios en blanco con el terminador NUL colocado al final de la cadena.</p> <p>PADNTSTR y NOPADNTSTR son opciones mutuamente excluyentes. El valor predeterminado (PADNTSTR o NOPADNTSTR) se especifica en el campo PAD NUL-TERMINATED en el Panel 2 de valores predeterminados de programación de aplicaciones durante la instalación.</p> <p>Esta opción solo se aplica a aplicaciones C y C++.</p>
PERIOD	<p>Reconoce el punto (.) como indicador de punto decimal en literales decimales o de coma flotante en los siguientes casos:</p> <ul style="list-style-type: none"> <li>Para instrucciones SQL estáticas en programas COBOL</li> <li>Para las sentencias SQL dinámicas, cuando el valor del parámetro de instalación DYNRULS es NO y el paquete o plan que contiene las sentencias SQL tiene un comportamiento de enlace, definición o invocación DYNAMICRULES.</li> </ul> <p>COMA y PUNTO son opciones mutuamente excluyentes. El valor predeterminado (COMA o PUNTO) se especifica en el campo PUNTO DECIMAL EN el Panel 1 de valores predeterminados de programación de aplicaciones durante la instalación.</p>
CITA1 Q	<p>Indica que el precompilador de Db2 debe utilizar las comillas ("") como delimitador de cadenas en las sentencias del lenguaje del host que genera.</p> <p>La CITA es válida solo para aplicaciones COBOL. La CITA no es válida para ninguna de las siguientes combinaciones de opciones de precompilador:</p> <ul style="list-style-type: none"> <li>CCSID(1026) y HOST(IBMCOB)</li> <li>CCSID(1155) y HOST(IBMCOB)</li> </ul> <p>El valor predeterminado se especifica en el campo STRING DELIMITER (DELIMITADOR DE CADENA) en el Panel 1 de valores predeterminados de programación de aplicaciones durante la instalación. Si STRING DELIMITER es el doble comilla ("") o DEFAULT, QUOTE es el valor predeterminado.</p> <p>APOST y QUOTE son opciones mutuamente excluyentes.</p>

Tabla 140. Opciones de procesamiento SQL (continuación)

Opción de palabra clave	Significado
QUOTESQL	<p>Reconoce las comillas dobles ("") como delimitador de cadena y el apóstrofo (') como carácter de escape SQL dentro de las sentencias SQL. Esta opción solo se aplica a COBOL.</p> <p>El valor predeterminado se especifica en el campo DELIMITADOR DE CADENA SQL en el Panel 1 de valores predeterminados de programación de aplicaciones durante la instalación. Si SQL STRING DELIMITER es el doble comilla (") o DEFAULT, QUOTESQL es el valor predeterminado.</p> <p>APOSTSQL y QUOTESQL son opciones mutuamente excluyentes.</p>
FUENTE 1 S	Enumera el código fuente y los diagnósticos del precompilador de Db2 .
SQL(ALL DB2)	<p>Indica si la fuente contiene sentencias SQL distintas de las reconocidas por Db2 for z/OS.</p> <p>SQL(ALL) se recomienda para programas de aplicación cuyas sentencias SQL deben ejecutarse en un servidor distinto de Db2 for z/OS mediante acceso DRDA. SQL(ALL) indica que las sentencias SQL del programa no son necesariamente para Db2 for z/OS. En consecuencia, el procesador de sentencias SQL acepta sentencias que no se ajustan a las reglas de sintaxis de la base de datos SQL (Db2). El procesador de instrucciones SQL interpreta y procesa las instrucciones SQL de acuerdo con las reglas de la arquitectura de base de datos relacional distribuida (DRDA). El procesador de instrucciones SQL también emite un mensaje informativo si el programa intenta utilizar IBM Palabras reservadas de SQL como identificadores ordinarios. SQL(ALL) no afecta a los límites del procesador de sentencias SQL.</p> <p>SQL (Db2), el valor predeterminado, significa interpretar sentencias SQL y comprobar la sintaxis para su uso por Db2 for z/OS. Se recomienda SQL (Db2) cuando el servidor de la base de datos es Db2 for z/OS.</p>

Tabla 140. Opciones de procesamiento SQL (continuación)

Opción de palabra clave	Significado
SQLLEVEL(V10R1 V11R1 / <i>nivel de función</i> )	<p>Indica si se acepta la sintaxis SQL que es nueva en los niveles de función de Db2 12 .</p> <p><b>SQLLEVEL (<i>nivel de función</i>)</b> Especifica el nivel de función permitido por el proceso de precompilación. El formato es <i>VvvRrMmm</i>, donde <i>vv</i> es la versión, <i>r</i> es el release y <i>mmm</i> es el nivel de modificación. SQLLEVEL V12R1M100 es equivalente a V11R1.</p> <p><b>SQLLEVEL( V11R1 )</b> Especifica que se permite cualquier sintaxis SQL hasta Db2 11 .</p> <p><b>SQLLEVEL( V10R1 )</b> Especifica que se permite cualquier sintaxis SQL hasta DB2 10 .</p> <p><b>SQLLEVEL( V9R1 )</b> Especifica que se permite cualquier sintaxis SQL hasta DB2 9 . DB2 9 es compatible, pero hace que el proceso de precompilación solo admite un nivel e DB2 9 e de sintaxis SQL.</p> <p><b>SQLLEVEL( V8R1 )</b> Especifica que se permite cualquier sintaxis SQL hasta DB2 version 8 . DB2 version 8 es compatible, pero hace que el proceso de precompilación solo admite un nivel e DB2 version 8 e de sintaxis SQL.</p> <p>El nivel de función activado en el subsistema Db2 no restringe el valor SQLLEVEL. Sin embargo, debe asegurarse de vincular el DBRM resultante con el nivel de compatibilidad de aplicación correcto en un subsistema de e Db2 , con el nivel de función correcto activado.</p>
STDSQL(NO SÍ) <sup>3</sup>	<p>Indica a qué reglas deben ajustarse las sentencias de salida.</p> <p>STDSQL(YES)<sup>3</sup> indica que las sentencias SQL precompiladas en el programa fuente se ajustan a ciertas reglas del estándar SQL. STDSQL(NO) indica conformidad con las normas de la Asociación Estadounidense de Seguridad ( Db2 ).</p> <p>El valor predeterminado se especifica en el campo STD SQL LANGUAGE en el Panel 2 de valores predeterminados de programación de aplicaciones durante la instalación.</p> <p>STDSQL(YES) implica automáticamente la opción NOFOR.</p>
HORA (ISO EE. UU. EUR JIS LOCAL)	<p>Especifica que la salida de hora siempre se devuelve en un formato concreto, independientemente del formato que se especifique como predeterminado de la ubicación.</p> <p>El valor predeterminado se especifica en el campo FORMATO DE HORA en el Panel 2 de valores predeterminados de programación de aplicaciones durante la instalación.</p> <p>El formato predeterminado viene determinado por los valores predeterminados de instalación del sistema en el que está vinculado el programa, no por los valores predeterminados de instalación del sistema en el que está precompilado el programa.</p> <p>No puede utilizar la opción LOCAL a menos que tenga una rutina de salida de tiempo.</p>

Tabla 140. Opciones de procesamiento SQL (continuación)

Opción de palabra clave	Significado
TWOPASS TW	<p>Procesos en dos pasos, de modo que las declaraciones no tienen que preceder a las referencias. Los valores predeterminados dependen de la opción HOST que se especifique.</p> <p>ONEPASS y TWOPASS son opciones mutuamente excluyentes.</p> <p>Para el coprocesador Db2 , puede especificar la opción TWOPASS solo para aplicaciones PL/I. Para aplicaciones C/C++ y COBOL, el coprocesador Db2 , utiliza la opción ONEPASS.</p>
VERSIÓN (aaaa/AUTO)	<p>Define el identificador de versión de un paquete, programa y el DBRM resultante. Un identificador de versión es un identificador SQL de hasta 64 bytes EBCDIC.</p> <p>Cuando especifique VERSION, el procesador de instrucciones SQL crea un identificador de versión en el programa y DBRM. Esto afecta al tamaño del módulo de carga y del DBRM. Db2 utiliza el identificador de versión cuando vincula el DBRM a un paquete.</p> <p>Si no se especifica una versión en el momento de la precompilación, una cadena vacía es el identificador de versión predeterminado. Si especifica AUTO, el procesador de instrucciones SQL utiliza el token de consistencia para generar el identificador de versión. Si el token de consistencia es una marca de tiempo, esta se convierte al formato de caracteres ISO y se utiliza como identificador de versión. La marca de tiempo que se utiliza se basa en el valor del reloj de la tienda.</p>
XREF5	Incluye una lista de referencias cruzadas ordenadas de los símbolos que se utilizan en las sentencias SQL en el resultado de la lista.

**Notas:**

1. El coprocesador de la biblioteca de enlace dinámico ( Db2 ) ignora esta opción cuando el compilador invoca el coprocesador de la biblioteca de enlace dinámico ( Db2 ) para preparar la aplicación.
2. Esta opción siempre está activa cuando el compilador invoca el coprocesador Db2 para preparar la aplicación.
3. Puede utilizar STDSQL(86) como en versiones anteriores de Db2. El procesador de instrucciones SQL lo trata igual que STDSQL(YES).
4. Las opciones del precompilador no afectan al comportamiento de ODBC.
5. El coprocesador de la biblioteca de enlace dinámico ( Db2 ) ignora esta opción cuando el compilador invoca el coprocesador de la biblioteca de enlace dinámico ( Db2 ) para preparar la aplicación. Sin embargo, si utiliza PL/I V4.1, o posterior, es compatible.

**Conceptos relacionados**

[Precisión para operaciones con números decimales](#)

Db2 acepta dos conjuntos de reglas para determinar la precisión y escala del resultado de una operación con números decimales.

[Valores de fecha y hora \(Db2 SQL\)](#)

**Tareas relacionadas**

[Crear una versión de paquete](#)

Si desea ejecutar diferentes versiones de un programa sin necesidad de realizar cambios en el plan de aplicación asociado, utilice versiones de paquete. Esta técnica es útil si necesita hacer cambios en el programa sin provocar una interrupción en la disponibilidad del programa.

[Establecimiento del nivel de programa](#)

El nivel de programa define el nivel para un módulo concreto. Esta información se almacena en la señal de consistencia, que está en un formato Db2 interno. La sustitución del nivel de programa en la señal de consistencia es posible, si es necesario, pero normalmente no se recomienda.

#### Referencia relacionada

[Valores predeterminados para las opciones de proceso de SQL](#)

Algunas opciones de proceso de sentencias SQL tienen valores predeterminados que se basan en valores especificados en los paneles predeterminados de programación de aplicación de DB2I.

## Valores predeterminados para las opciones de proceso de SQL

Algunas opciones de proceso de sentencias SQL tienen valores predeterminados que se basan en valores especificados en los paneles predeterminados de programación de aplicación de DB2I.

La siguiente tabla muestra esas opciones y valores predeterminados.

Tabla 141. IBM -instalación suministrada instrucción SQL predeterminada opciones de procesamiento. El instalador puede cambiar estos valores predeterminados.

Opción de instalación	Instalar predeterminado	Opción de procesamiento de instrucciones SQL equivalentes	Opciones de procesamiento de instrucciones SQL disponibles
STRING DELIMITER	comillas ("")	QUOTE	COTIZACIÓN
delimitador de serie SQL	comillas ("")	QUOTESQL	APOSTSQLQUOTESQL
EL PUNTO DECIMAL ES	PERIODO	PERIODO	COMPARACIÓN DE PERÍODOS
DATE FORMAT	ISO	FECHA (ISO)	FECHA (ISO EE. UU.  EUR  JIS LOCAL)
ARITMÉTICA DECIMAL	DEC15	DIC(15)	DIC(15 31)
datos mixtos	NEE	CCSID(n)	CCSID(n)
IDIOMA PREDETERMINADO	COBOL	HOST(COBOL)	HOST(ASM C[(FOLD)] CPP[(FOLD)] IBMCOB FORTRAN PLI)
STD LENGUAJE SQL	NEE	STDSQL(NO)	STDSQL( YES NO 86 )
Formato de hora	ISO	HORA (ISO)	HORA (EST USA EUR  JIS LOCAL)

**Notas:** Para las sentencias SQL dinámicas, otra programación predeterminada de la aplicación, USE FOR DYNAMICRULES, determina si Db2 utiliza la programación predeterminada de la aplicación o la opción del procesador de sentencias SQL para las siguientes opciones de instalación:

- STRING DELIMITER
- delimitador de serie SQL
- EL PUNTO DECIMAL ES
- ARITMÉTICA DECIMAL

Si el valor de USE FOR DYNAMICRULES es YES, las sentencias SQL dinámicas utilizan los valores predeterminados de programación de la aplicación. Si el valor de USE FOR DYNAMICRULES es NO, las sentencias SQL dinámicas en paquetes o planes con comportamiento de enlace, definición e invocación utilizan las opciones del procesador de sentencias SQL.

Algunas opciones del procesador de instrucciones SQL tienen valores predeterminados basados en el lenguaje del host. Algunas opciones no se aplican a algunos idiomas. La siguiente tabla muestra las opciones y valores predeterminados en función del idioma.

Tabla 142. Opciones y valores predeterminados del precompilador de Db2 , según el idioma

Valor de host	Valores predeterminados
ASM	APOST1, APOSTSQL1, PERIOD1, TWOPASS, MARGINS(1,71,16)
C o CPP	APOST1, APOSTSQL1, PERIOD1, ONEPASS, MARGINS(1,72)
IBMCOB	QUOTE2, QUOTESQL2, PERIOD, ONEPASS1, MARGINS(8,72)1
FORTRAN	APOST1, APOSTSQL1, PERIOD1, ONEPASS1, MARGINS(1,72)1
PLI	APOST1, APOSTSQL1, PERIOD1, ONEPASS, MARGINS(2,72)
SQL o SQLPL	APOST1, APOSTSQL1, PERIOD1, ONEPASS, MARGINS(1,72)

**Notas:**

1. Obligatorio para este idioma; no se permite ninguna alternativa.
2. El valor predeterminado se elige en el Panel 1 de valores predeterminados de programación de aplicaciones durante la instalación. Los IBM -instalación suministrada por defecto para los delimitadores de cadenas son QUOTE (delimitador de lenguaje host) y QUOTESQL (carácter de escape SQL). El instalador puede sustituir los IBM -valores predeterminados suministrados por otros valores predeterminados. Las opciones del precompilador que especifique anulan cualquier valor predeterminado que esté en vigor.

## Valores predeterminados de procesamiento de instrucciones SQL para instrucciones dinámicas

Por lo general, las sentencias dinámicas utilizan los valores predeterminados que se especifican durante la instalación. Sin embargo, si el valor del parámetro DYNRULS del módulo de valores predeterminados de la aplicación es NO, puede utilizar estas opciones para instrucciones SQL dinámicas en paquetes o planes con comportamiento de enlace, definición o invocación:

- COMA o PUNTO
- APOSTAR o COTIZAR
- APOSTSQL o QUOTESQL
- DIC(15) o DIC(31)

### Conceptos relacionados

Opciones de reglas dinámicas para sentencias de SQL dinámico

La opción de vinculación DYNAMICRULES y el entorno de ejecución determinan las reglas para los atributos de SQL dinámico.

## Opciones SQL para el acceso DRDA

Algunas opciones de procesamiento de sentencias SQL son relevantes cuando se prepara un paquete para ejecutarse con acceso DRDA.

Las siguientes opciones de procesamiento de instrucciones SQL son relevantes para el acceso DRDA :

### CONNECT

Utilice CONNECT(2), explícitamente o por defecto.

CONNECT(1) hace que sus sentencias CONNECT permitan solo la función restringida conocida como "unidad de trabajo remota". Tenga especial cuidado de evitar CONNECT(1) si su aplicación actualiza más de un DBMS en una sola unidad de trabajo.

## **SQL**

Utiliza SQL (ALL) explícitamente para un paquete que se ejecuta en un servidor que no es Db2 for z/OS. El precompilador acepta entonces cualquier declaración que obedezca a las reglas DRDA.

Utiliza SQL (DB2), explícitamente o por defecto, si el servidor es solo Db2 for z/OS . El precompilador rechaza cualquier instrucción que no obedezca las reglas de Db2 for z/OS.

## **Compilación y edición de enlaces de una aplicación**

---

Si utiliza el Db2 coprocessor, procesa las sentencias SQL a medida que compila su programa, y el siguiente paso es el enlace de edición del programa. El propósito del paso de edición de enlaces es producir un módulo de carga ejecutable.

### **Acerca de esta tarea**

Para programas que no sean C y C++, debe utilizar procedimientos JCL cuando utilice el coprocesador Db2 . Para los programas C y C++, puede utilizar procedimientos JCL o UNIX System Services en z/OS para invocar el coprocesador de la biblioteca de enlaces dinámicos ( Db2 ).

Para obtener más información, consulte “[Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor](#)” en la página 897.

**Consejo:** Db2 coprocessor » es el método recomendado para procesar instrucciones SQL en programas de aplicación. En comparación con el Db2 precompiler, el Db2 coprocessor tiene menos restricciones en los programas SQL y es más compatible con las últimas mejoras de SQL y de los lenguajes de programación. Consulte el “[Procesamiento de sentencias SQL mediante el uso del Db2 coprocessor](#)” en la página 897.

Si utiliza el precompilador Db2 precompiler , como se describe en “[Procesamiento de sentencias SQL mediante el uso del Db2 precompiler](#)” en la página 901, el siguiente paso en el proceso de preparación del programa es compilar y editar el enlace del programa.

### **Procedimiento**

- Puede utilizar uno de los siguientes métodos para compilar y editar enlaces de una aplicación:
  - DB2I paneles. Para obtener detalles, consulte “[Paneles de DB2I que se utilizan para la preparación del programa](#)” en la página 965.
  - El procedimiento de comando DSNH (un TSO CLIST). Para obtener detalles, consulte [Procedimiento de comando DSNH \(TSO CLIST\) \(Comandos de Db2\)](#).
  - Procedimientos JCL suministrados con Db2. Para obtener detalles, consulte “[Procedimientos JCL proporcionados por Db2 para la preparación de una aplicación](#)” en la página 962.
  - Procedimientos JCL suministrados con un compilador de lenguaje host.
- Utilizar un procedimiento de edición de enlaces que cree un módulo de carga que satisfaga los requisitos específicos del entorno del programa.

### **TSO y lote**

Incluya el módulo de interfaz de idioma de la función de adjunto de TSO ( Db2 , DSNELI) o el módulo de interfaz de idioma de la función de adjunto de llamada ( Db2 , DSNALI) o el módulo de interfaz de idioma universal (Universal Language Interface, DSNULI).

### **IMS**

Incluye el módulo de interfaz de lenguaje de programación ( Db2 ), que contiene el punto de entrada DSNHLI IMS módulo de interfaz de lenguaje ( DFSLI000 ), que contiene el punto de entrada DSNHLI. Además, la IMS RESLIB debe preceder a la biblioteca SDSNLOAD en la lista de enlaces, concatenaciones JOBLIB o STEPLIB.

IMSDb2 , comparten un nombre de alias común, DSNHLI, para el módulo de interfaz de idioma. Debe hacer lo siguiente cuando concatene sus bibliotecas:

- Si utiliza IMS, asegúrese de concatenar la IMS biblioteca primero para que el programa de aplicación compile con la IMS versión de DSNHLI.
- Si ejecuta su programa de aplicación solo en Db2, asegúrese de concatenar primero la biblioteca Db2 .

### CICS

Incluya el módulo de interfaz de lenguaje de comandos de red (Db2, DSNCLI) o el módulo de interfaz de lenguaje universal (Universal Language Interface, DSNULI) CICS módulo de interfaz de lenguaje (DSNCLI) o el módulo de interfaz de lenguaje universal (DSNULI). Puede vincular DSNCLI con su programa en modo de direccionamiento de 24 o 31 bits (AMODE=31), pero DSNULI debe vincularse con su programa en modo de direccionamiento de 31 bits (AMODE=31). Si su aplicación se ejecuta en modo de direccionamiento de 31 bits, debe vincular y editar el stub DSNCLI o DSNULI a su aplicación con los atributos AMODE=31 y RMODE=ANY para que su aplicación pueda ejecutarse por encima de la línea de 16 MB.

También necesita el CICS Módulo de interfaz EXEC adecuado para el lenguaje de programación. CICS requiere que este módulo sea la primera sección de control (CSECT) en el módulo de carga final.

El tamaño del módulo de carga ejecutable que se produce en el paso de enlace-edición varía en función de los valores que el procesador de instrucciones SQL inserta en el código fuente del programa.

### Edición de vínculos de un programa por lotes

Db2 dispone de rutinas de interfaz de idioma para cada entorno único compatible. Db2 requiere la IMS rutina de interfaz de idioma para el lote DL/I. Necesita tener un DFSLI000 o vinculado con el programa de aplicación.

### Conceptos relacionados

#### Interfaz de lenguaje universal (DSNULI)

El subcomponente de la interfaz de lenguaje universal (DSNULI) determina el entorno en tiempo de ejecución y carga y se bifurca dinámicamente al modelo de interfaz de lenguaje adecuado.

### Tareas relacionadas

#### Preparación de una aplicación para ejecutarse en Db2 for z/OS

Para preparar y ejecutar aplicaciones que contengan sentencias SQL estáticas incrustadas o sentencias SQL dinámicas, debe procesar, compilar, editar vínculos y vincular las sentencias SQL.

### Referencia relacionada

#### Procedimiento de comando DSNH (TSO CLIST) (Comandos de Db2 )

### Información relacionada

#### Pasos de preparación de programas de CICS (CICS Transaction Server for z/OS)

## Enlace de planes y paquetes de aplicación

Debe vincular el DBRM que produce el procesador de instrucciones SQL a un paquete antes de que su aplicación de e Db2 encia pueda ejecutarse. El proceso de enlace establece una relación entre un programa de aplicación y sus datos relacionales.

### Antes de empezar

Debe tener los privilegios adecuados. Para obtener más información, consulte Privilegios requeridos para la gestión de planes y paquetes (Managing Security)

### Acerca de esta tarea

Durante el proceso de precompilación, el precompilador Db2 produce tanto código fuente modificado como un módulo de solicitud de base de datos (DBRM) para cada programa de aplicación. El código fuente modificado debe compilarse y editarse mediante vínculos antes de que el programa pueda

ejecutarse. Los DBRM deben estar vinculados a un paquete. A continuación, puede asociar ese paquete a un plan de aplicación concreto.

Durante el proceso de vinculación, Db2 también completa las siguientes acciones:

- Valida las referencias de objetos en las sentencias SQL del programa, como los nombres de tablas, vistas y columnas, con el catálogo de objetos de SQL ( Db2 ). Dado que el proceso de enlace se produce antes de la ejecución del programa, los errores se detectan y pueden corregirse antes de que se ejecute el programa.
- Verifica la autorización del proceso de enlace para especificar el propietario del programa y la autorización del propietario especificado para acceder a los datos solicitados por las sentencias SQL en el programa.
- Selecciona las rutas de acceso que utiliza Db2 para acceder a los datos del programa. Db2 considera factores como el tamaño de la tabla, los índices disponibles y otros, al seleccionar las rutas de acceso.

Al determinar el tamaño máximo de un plan, debe tener en cuenta varias limitaciones físicas, como el tiempo necesario para vincular el plan, el tamaño del grupo de EDM y la fragmentación. Como regla general, el fondo de EDM debe ser al menos 10 veces mayor que el mayor DBD o plan, lo que sea mayor.

Cada paquete que se vincula sólo puede contener un DBRM.

**Excepción:** No es necesario vincular un DBRM si la única instrucción SQL del programa es SET CURRENT PACKAGESET.

Dado que no se necesita un plan o paquete para ejecutar la sentencia SET CURRENT PACKAGESET, la opción de enlace ENCODING no afecta a la sentencia SET CURRENT PACKAGESET. Una aplicación que necesite proporcionar un valor de variable de host en un esquema de codificación distinto del esquema de codificación predeterminado del sistema debe utilizar la instrucción DECLARE VARIABLE para especificar el esquema de codificación de la variable de host.

Debe vincular los planes localmente, independientemente de si hacen referencia a paquetes que se ejecutan de forma remota. Sin embargo, debe vincular los paquetes que se ejecutan en ubicaciones remotas en esas ubicaciones remotas.

Desde un solicitante de Db2 , puede ejecutar un plan especificándolo en el subcomando RUN, pero no puede ejecutar un paquete directamente. Debe incluir el paquete en un plan y, a continuación, ejecutar el plan.

**Consejo:** Desarrolle una convención de nomenclatura y una estrategia para el uso más eficaz y eficiente de sus planes y paquetes.

## Procedimiento

Para vincular programas de aplicación, realice las siguientes acciones.

1. Para vincular DBRM individuales en paquetes, utilice los comandos BIND PACKAGE con ACTION(REPLACE). Los paquetes le ofrecen la flexibilidad de probar diferentes versiones de un programa sin tener que volver a vincular todo en el plan de aplicación.

Para los programas cuyos DBRM correspondientes están en archivos HFS, puede utilizar el Db2 command line processor para vincular los DBRM a los paquetes. Opcionalmente, también puede copiar el DBRM en un miembro de conjunto de datos particionado utilizando los comandos **oput** y **oget** y, a continuación, enlazarlo mediante JCL convencional.

Para crear nuevos paquetes de activadores para activadores existentes, debe volver a crear el activador que está asociado con el paquete. Para obtener más información, consulte “[Paquetes desencadenantes](#)” en la página 172.

2. Para designar paquetes en planes de aplicación, utilice el comando BIND PLAN con ACTION(REPLACE). Los planes pueden especificar paquetes, colecciones de paquetes o una combinación de estos elementos. Si especifica uno o más DBRM para incluirlos en el plan (utilizando la opción MIEMBRO de VINCULAR PLAN), Db2 vincula automáticamente esos DBRM en paquetes y luego vincula esos paquetes en el plan. El plan contiene información sobre los paquetes designados y sobre

los datos que los programas de aplicación pretenden utilizar. El plan se almacena en el catálogo de Db2 .

### Conceptos relacionados

[Copias de paquetes para la gestión de planes \(Rendimiento de Db2\)](#)

[Revinculación automática](#)

*Las reasociaciones automáticas* (a veces llamadas "autobindings") se producen cuando un usuario autorizado ejecuta un paquete o plan y las estructuras de tiempo de ejecución del plan o paquete no se pueden utilizar. Esta situación se da normalmente cuando se modifican los atributos de los datos de los que depende el paquete o plan, o si se modifica el entorno en el que se ejecuta el paquete o plan.

### Tareas relacionadas

[Vinculación de un DBRM que está en un archivo HFS a un paquete o colección](#)

Si los DBRM están en archivos HFS de UNIX de l z/OS , puede utilizar el Db2 command line processor para vincular los DBRM a paquetes en el servidor de destino de l Db2 . De manera opcional, también puede copiar el DBRM en un miembro de conjunto de datos particionado utilizando los mandatos oput y oget de TSO/E y, a continuación, vincular el DBRM utilizando el JCL convencional.

### Referencia relacionada

[BIND PACKAGE subcomando \(DSN\) \(Comandos de Db2\)](#)

[BIND PLAN subcomando \(DSN\) \(Comandos de Db2\)](#)

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2\)](#)

## Crear una versión de paquete

Si desea ejecutar diferentes versiones de un programa sin necesidad de realizar cambios en el plan de aplicación asociado, utilice versiones de paquete. Esta técnica es útil si necesita hacer cambios en el programa sin provocar una interrupción en la disponibilidad del programa.

### Acerca de esta tarea

Puede crear una versión de paquete diferente para cada versión del programa. Cada paquete tiene el mismo nombre de paquete y nombre de colección, pero cada paquete tiene asociado un número de versión diferente. El plan que incluye ese paquete incluye todas las versiones de ese paquete. Por lo tanto, puede ejecutar un programa que esté asociado con cualquiera de las versiones del paquete sin tener que volver a vincular el plan de aplicación, cambiar el nombre del plan o modificar los subcomandos RUN que lo utilizan.

### Procedimiento

Para crear una versión de paquete:

1. Precompila tu programa con la opción VERSION (*identificador-de-versión* ).
2. Enlazar el DBRM resultante con el mismo nombre de colección y nombre de paquete que cualquier versión existente de ese paquete. Cuando ejecutas el programa, Db2 utiliza la versión del paquete que especificaste cuando lo precompilaste.

### Ejemplo

Supongamos que vincula un plan con la siguiente declaración:

```
BIND PLAN (PLAN1) PKLIST (COLLECT.*)
```

Los siguientes pasos muestran cómo crear dos versiones de un paquete, una para cada uno de los dos programas.

Número de paso	Para la versión 1 del paquete	Para la versión 2 del paquete
1	Programa de precompilación 1. Especifique VERSIÓN(1).	Precompilar programa versión 2. Especifique VERSIÓN(2).

Número de paso	Para la versión 1 del paquete	Para la versión 2 del paquete
2	Enlazar el DBRM con el nombre de la colección COLLECT y el nombre del paquete PACKA.	Enlazar el DBRM con el nombre de la colección COLLECT y el nombre del paquete PACKA.
3	Enlace el programa de edición 1 a su aplicación.	Enlace el programa de edición 2 a su aplicación.
4	Ejecute la aplicación; utiliza el programa 1 y PACKA, VERSIÓN 1.	Ejecute la aplicación; utiliza el programa 2 y PACKA, VERSIÓN 2.

## Vinculación de un DBRM que está en un archivo HFS a un paquete o colección

Si los DBRM están en archivos HFS de UNIX de I z/OS , puede utilizar el Db2 command line processor para vincular los DBRM a paquetes en el servidor de destino de I Db2 . De manera opcional, también puede copiar el DBRM en un miembro de conjunto de datos particionado utilizando los mandatos oput y oget de TSO/E y, a continuación, vincular el DBRM utilizando el JCL convencional.

### Acerca de esta tarea

#### Restricciones:

No se puede especificar el comando REBIND con el comando Db2 command line processor. De forma alternativa, especifique el comando BIND con la opción ACTION( REPLACE).

No puede especificar el comando FREE PACKAGE con el comando Db2 command line processor. De forma alternativa, especifique la sentencia DROP PACKAGE para eliminar los paquetes existentes.

### Procedimiento

Para vincular un DBRM que se encuentra en un archivo HFS a un paquete o colección:

1. Invoque el Db2 command line processor y conéctese al servidor de Db2 de destino.
2. Especifique el comando BIND con las opciones adecuadas.

#### Conceptos relacionados

[Db2 command line processor , \( Db2 Commands\)](#)

#### Tareas relacionadas

[Procesamiento de instrucciones SQL mediante el uso del Db2 coprocessor](#)

Puede utilizar el Db2 coprocessor para procesar instrucciones SQL en tiempo de compilación. Con el compilador de consultas de bases de datos ( Db2 coprocessor), el compilador analiza un programa y copia todas las instrucciones SQL y la información de las variables del host en un módulo de solicitud de base de datos (DBRM). Db2 coprocessor » es el método recomendado para procesar instrucciones SQL en programas de aplicación. En comparación con el Db2 precompilador, el Db2 coprocessor tiene menos restricciones en los programas SQL y es más compatible con las últimas mejoras de SQL y de los lenguajes de programación.

#### Referencia relacionada

[Db2 command line processor BIND, mandato](#)

Utilice el comando BIND de Db2 command line processor para vincular DBRM que están en z/OS Archivos HFS de UNIX a paquetes.

## Db2 command line processor BIND, mandato

Utilice el comando BIND de Db2 command line processor para vincular DBRM que están en z/OS Archivos HFS de UNIX a paquetes.

El siguiente diagrama muestra la sintaxis del comando BIND de Db2 command line processor .

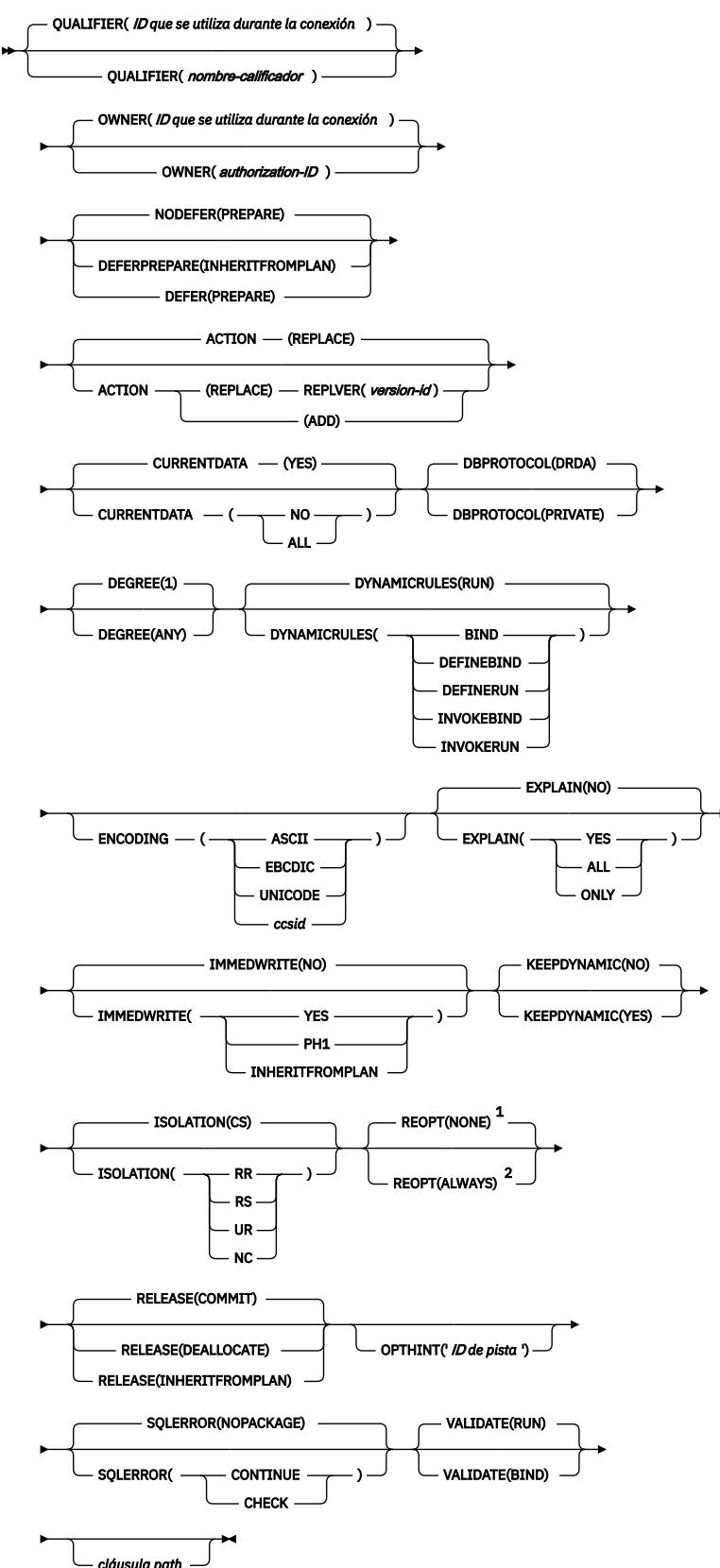
```
► BIND — dbrm-nombre-archivo —
 ↗
 ↘ -COLLECTION — nombre-colección 1
 ↗
 ↘ ►— cláusula de opciones —2 ►
```

Notas:

<sup>1</sup> Si no especifica una colección, Db2 utiliza NULLID.

<sup>2</sup> Puede especificar las opciones después de *collection-name* en cualquier orden.

***cláusula-de-opciones:***

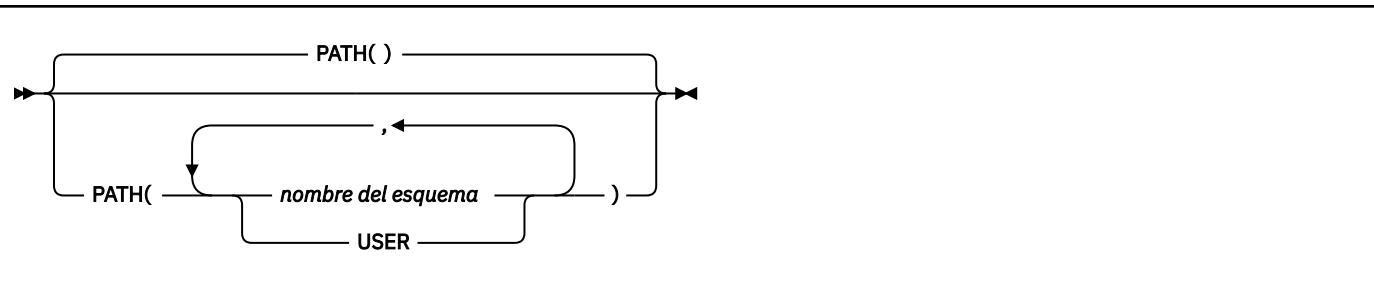


Notas:

<sup>1</sup> Puede especificar NOREOPT(VARS) como sinónimo de REOPT(NONE).

<sup>2</sup> Puede especificar REOPT(VARS) como sinónimo de REOPT(ALWAYS).

### **cláusula path:**



Las siguientes opciones son exclusivas de este diagrama:

#### **DATOS ACTUALES (TODOS)**

Especifica que para todos los cursorse requiere la moneda de los datos y se inhibe la obtención de bloques.

#### **SQLERROR(CHECK)**

Especifica que el procesador de Db2 command line processor solo debe comprobar si hay errores SQL en el DBRM. No se genera ningún paquete.

#### **IMMEDWRITE( PH1 )**

Especifica que se realiza una actividad de escritura normal. Esta opción es equivalente a IMMEDWRITE(NO).

#### **EXPLICAR (TODO)**

Especifica que Db2 debe insertar información en las tablas EXPLAIN apropiadas. Esta opción equivale a EXPLAIN (YES).

#### **Referencia relacionada**

Opciones BIND y REBIND para paquetes, planes y servicios (comandos Db2 )

## **Enlace de un plan de aplicación**

Un plan de aplicación puede incluir listas de paquetes.

### **Procedimiento**

Para vincular un plan de aplicación, utilice el subcomando BIND PLAN con al menos una de las siguientes opciones:

#### **MIEMBRO**

Especifique esta opción para vincular DBRM a un paquete y, a continuación, vincular la lista de paquetes a un plan. Después de la palabra clave MEMBER, especifique los nombres de los miembros del DBRMS.

#### **PKLIST**

Especifique esta opción para incluir listas de paquetes en el plan. Después de la palabra clave PKLIST, especifique los nombres de los paquetes que se incluirán en la lista de paquetes. Para incluir una colección completa de paquetes en la lista, utilice un asterisco después del nombre de la colección. Por ejemplo, PKLIST(GROUP1.\*).

#### **Especificación de la lista de paquetes para la opción PKLIST de BIND PLAN**

El orden en el que especifique los paquetes en una lista de paquetes puede afectar al rendimiento del tiempo de ejecución. La búsqueda de un paquete específico implica buscar en el directorio Db2 , lo que puede resultar costoso. Cuando utiliza *collection-id.\** con la palabra clave PKLIST, debe especificar primero las colecciones en las que Db2 es más probable que encuentre un paquete.

Por ejemplo, supongamos que realiza el siguiente enlace:

```
BIND PLAN (PLAN1) PKLIST (COLL1.* , COLL2.* , COLL3.* , COLL4.*)
```

A continuación, ejecute el programa PROG1. Db2 realiza la siguiente búsqueda de paquetes:

- a. Comprueba si el programa PROG1 está vinculado como parte del plan
- b. Búsquedas de *COLL1.PROG1.timestamp*
- c. Si no encuentra *COLL1.PROG1.timestamp*, busca *COLL2.PROG1.timestamp*
- d. Si no encuentra *COLL2.PROG1.timestamp*, busca *COLL3.PROG1.timestamp*
- e. Si no encuentra *COLL3.PROG1.timestamp*, busca *COLL4.PROG1.timestamp*.

**Cuando ambos registros especiales CURRENT PACKAGE PATH y CURRENT PACKAGESET contienen una cadena vacía**

Si no configura estos registros especiales, Db2 busca un DBRM o un paquete en una de estas secuencias:

- En la ubicación local (si SERVIDOR ACTUAL está en blanco o especifica esa ubicación explícitamente), el orden es:
  - a. Todos los paquetes que ya están asignados al plan mientras este está en funcionamiento.
  - b. Todos los paquetes no asignados que se especifican explícitamente en la lista de paquetes del plan y todas las colecciones que se incluyen completamente en ella. Db2 busca paquetes en el orden en que aparecen en la lista de paquetes.
- En una ubicación remota, el orden es:
  - a. Todos los paquetes que ya están asignados al plan en esa ubicación mientras el plan está en funcionamiento.
  - b. Todos los paquetes no asignados que se especifican explícitamente en la lista de paquetes del plan, y todas las colecciones que están completamente incluidas en ella, cuyas ubicaciones coinciden con el valor de CURRENT SERVER. Db2 busca paquetes en el orden en que aparecen en la lista de paquetes.

Si utiliza la opción DIFERIR (PREPARAR) de ENLAZAR PLAN, Db2 no busca en todas las colecciones de la lista de paquetes.

**Si el orden de búsqueda no es importante**

En muchos casos, el orden en el que Db2 busca los paquetes no es importante para usted y no afecta al rendimiento. Para una aplicación que se ejecute únicamente en su sistema de correo electrónico local (Db2), puede nombrar cada paquete de forma diferente e incluirlos todos en la misma colección. La lista de paquetes en su subcomando BIND PLAN puede decir:

```
PKLIST (collection.*)
```

El plan resultante consta de la siguiente información:

- Cualquier programa que esté asociado con DBRM en la lista de MIEMBROS
- Cualquier programa que esté asociado con paquetes y colecciones que estén identificados en PKLIST

Puede añadir paquetes a la colección incluso después de vincular el plan. Db2 le permite agrupar paquetes que tengan el mismo nombre de paquete en la misma colección solo si sus identificadores de versión son diferentes.

Si su aplicación utiliza acceso DRDA, debe vincular algunos paquetes en ubicaciones remotas. Utilice el mismo nombre de colección en cada ubicación e identifique su lista de paquetes como:

```
PKLIST (*.collection.*)
```

Si utiliza un asterisco para parte de un nombre en una lista de paquetes, Db2 comprueba la autorización del paquete al que se resuelve el nombre en tiempo de ejecución. Para evitar la comprobación en tiempo de ejecución del ejemplo anterior, puede conceder la autoridad EXECUTE para toda la colección al propietario del plan antes de vincular el plan.

**Tareas relacionadas**

[Mejora del rendimiento para las aplicaciones que acceden a datos distribuidos \(Db2 Performance\)](#)

## **Referencia relacionada**

[BIND PLAN subcomando \(DSN\) \(Comandos de Db2 \)](#)

[REGISTRO ESPECIAL DE RUTA DE PAQUETE ACTUAL \( Db2 SQL\)](#)

[PAQUETES ACTUALES Registro especial \( Db2 SQL\)](#)

## **Cómo identifica Db2 paquetes en tiempo de ejecución**

El precompilador de Db2 o coprocesador de Db2 identifican cada llamada a Db2 con una señal de consistencia. La misma señal de consistencia identifica al DBRM que produce el procesador de sentencias SQL y paquete al que vincula el DBRM.

Cuando ejecutas el programa, Db2 utiliza el token de consistencia para hacer coincidir la llamada a Db2 con el DBRM correcto. Por lo general, el token de consistencia está en un formato de Db2 . Puede anular ese token si lo desea.

También necesita otros identificadores. El token de consistencia por sí solo no identifica necesariamente un paquete único. Puede vincular el mismo DBRM a muchos paquetes, en diferentes ubicaciones y en diferentes colecciones, y puede incluir todos esos paquetes en la lista de paquetes del mismo plan. Todos esos paquetes tendrán el mismo token de consistencia. Puede especificar una ubicación concreta o una colección concreta en tiempo de ejecución.

## **Tareas relacionadas**

[Establecimiento del nivel de programa](#)

El nivel de programa define el nivel para un módulo concreto. Esta información se almacena en la señal de consistencia, que está en un formato Db2 interno. La sustitución del nivel de programa en la señal de consistencia es posible, si es necesario, pero normalmente no se recomienda.

## **Especificación de la ubicación del paquete que debe utilizar Db2**

Cuando su programa ejecuta sentencias SQL, Db2 utiliza el valor del registro especial CURRENT SERVER para determinar la ubicación del paquete necesario. Si el servidor actual es su subsistema local de Db2 y no tiene un nombre de ubicación, el valor en el registro especial está en blanco.

## **Acerca de esta tarea**

Puede cambiar el valor de CURRENT SERVER utilizando la instrucción SQL CONNECT en su programa. Si no utiliza CONNECT, el valor de CURRENT SERVER es el nombre de ubicación de su subsistema local Db2 (o está en blanco, si su subsistema Db2 no tiene nombre de ubicación).

## **Especificación de la recogida de paquetes que utiliza Db2 para las solicitudes**

Puede asegurarse de que Db2 utiliza el punto de recogida de paquetes previsto y no pierde tiempo buscando, especificando explícitamente el punto de recogida de paquetes que desea que Db2 utilice.

## **Acerca de esta tarea**

Puede utilizar el registro especial CURRENT PACKAGE PATH o CURRENT PACKAGESET (si CURRENT PACKAGE PATH no está configurado) para especificar las colecciones que utiliza Db2 para la resolución de paquetes. El registro especial PAQUETE ACTUAL contiene el nombre de una sola colección, y el registro especial RUTA DEL PAQUETE ACTUAL contiene una lista de nombres de colecciones.

Si no establece estos registros especiales, contendrán una cadena vacía cuando la aplicación comience a ejecutarse, y permanecerán como una cadena vacía. En este caso, Db2 busca en las colecciones de paquetes disponibles.

Sin embargo, especificar explícitamente la recogida prevista mediante el uso de los registros especiales puede evitar una búsqueda potencialmente costosa a través de una lista de paquetes que tiene muchas entradas que cumplen los requisitos. Además, Db2 utiliza los valores de estos registros especiales para aplicaciones que no se ejecutan bajo un plan.

Cuando llama a un procedimiento almacenado externo, el registro especial CURRENT PACKAGESET contiene el valor que especificó para el parámetro COLLID cuando definió el procedimiento almacenado. Si la rutina se definió sin un valor para el parámetro COLLID, el valor para el registro especial se hereda del programa de llamada.

Si el procedimiento almacenado externo no está definido con un valor COLLID especificado y el paquete de programa de llamada no tiene el mismo ID de colección que el paquete para el procedimiento almacenado, es posible que reciba SQLCODE -805. En esta situación, puede emitir una declaración ALTER PROCEDURE con la cláusula COLLID para solucionar el problema.

Además, el registro especial CURRENT PACKAGE PATH contiene el valor que especificó para el parámetro PACKAGE PATH cuando definió el procedimiento almacenado. Cuando el procedimiento almacenado devuelve el control al programa de llamada, Db2 restaura este registro al valor que contenía antes de la llamada.

**Especificación de la colección de paquetes para una rutina Java en un archivo JAR que está instalado en el catálogo de Db2** : Si la definición de la rutina Java especifica un valor COLLID que es diferente de la colección a la que están vinculados los paquetes de IBM Data Server Driver for JDBC and SQLJ , debe ejecutar la utilidad DB2Binder con la opción -collection para vincular los paquetes de controladores a la colección especificada por el valor COLLID en la definición de la rutina Java. Si no lo hace, podría recibir el código SQLCODE -805.

#### Tareas relacionadas

[Enlace de un plan de aplicación](#)

Un plan de aplicación puede incluir listas de paquetes.

[Alteración temporal de los valores que Db2 utiliza para resolver listas de paquetes](#)

Db2 resuelve listas de paquetes buscando las recopilaciones disponibles en un orden determinado. Para evitar esta búsqueda, puede especificar los valores que debe utilizar Db2 para la resolución de paquetes.

#### Referencia relacionada

[PAQUETES ACTUALES Registro especial \( Db2 SQL\)](#)

[REGISTRO ESPECIAL DE RUTA DE PAQUETE ACTUAL \( Db2 SQL\)](#)

[Programa de utilidad DB2Binder \(Db2 Application Programming for Java\)](#)

#### Información relacionada

[-805 \(Db2 Codes\)](#)

## Sustitución de los valores que utiliza Db2 para resolver listas de paquetes

Db2 resuelve listas de paquetes buscando las recopilaciones disponibles en un orden determinado. Para evitar esta búsqueda, puede especificar los valores que debe utilizar Db2 para la resolución de paquetes.

#### Acerca de esta tarea

Si establece el registro especial CURRENT PACKAGE PATH o CURRENT PACKAGESET, Db2 omite la comprobación de los programas que forman parte de un plan y utiliza los valores de estos registros para la resolución del paquete.

Si establece CURRENT PACKAGE PATH, Db2 utiliza el valor de CURRENT PACKAGE PATH como lista de nombres de colección para la resolución de paquetes. Por ejemplo, si CURRENT PACKAGE PATH contiene la lista COLL1, COLL2, COLL3, COLL4, Db2 busca el primer paquete que existe en el siguiente orden:

*COLL 1.PROG1.timestamp*

*COLL2.PROG1. marca de tiempo*

*COLL 3.PROG1.timestamp*

*COLL 4.PROG1.timestamp*

Si establece CURRENT PACKAGESET y **no** CURRENT PACKAGE PATH, Db2 utiliza el valor de CURRENT PACKAGESET como colección para la resolución de paquetes. Por ejemplo, si CURRENT PACKAGESET contiene COLL5, Db2 utiliza *COLL5.PROG1.timestamp* para la búsqueda de paquetes.

Cuando se establece CURRENT PACKAGE PATH, el servidor que recibe la solicitud ignora la colección especificada por la solicitud y, en su lugar, utiliza el valor de CURRENT PACKAGE PATH en el servidor para resolver el paquete. Especificar una lista de recopilación con el registro especial CURRENT PACKAGE PATH puede evitar la necesidad de emitir múltiples instrucciones SET CURRENT PACKAGESET para cambiar las recopilaciones para la búsqueda de paquetes.

La siguiente tabla muestra ejemplos de la relación entre el registro especial CURRENT PACKAGE PATH y el registro especial CURRENT PACKAGESET.

*Tabla 143. Alcance del PROCESO ACTUAL DEL PAQUETE*

Ejemplo	Qué ocurre
SET CURRENT PACKAGESET SELECT ... FROM T1 ...	La colección en PACKAGESET determina qué paquete se invoca.
SET CURRENT PACKAGE PATH SELECT ... FROM T1 ...	Las colecciones en PACKAGE PATH determinan qué paquete se invoca.
SET CURRENT PACKAGESET SET CURRENT PACKAGE PATH SELECT ... FROM T1 ...	Las colecciones en PACKAGE PATH determinan qué paquete se invoca.
SET CURRENT PACKAGE PATH CONNECT TO S2 ... SELECT ... FROM T1 ...	PACKAGE PATH en el servidor S2 es una cadena vacía porque no se ha establecido explícitamente. Los valores de la opción de enlace PKLIST del plan que se encuentra en el solicitante determinan qué paquete se invoca. <sup>1</sup>
SET CURRENT PACKAGE PATH = 'A,B' CONNECT TO S2 ... SET CURRENT PACKAGE PATH = 'X,Y' SELECT ... FROM T1 ...	Las colecciones en PACKAGE PATH que se establecen en el servidor S2 determinan qué paquete se invoca.
SET CURRENT PACKAGE PATH SELECT ... FROM S2.QUAL.T1 ...	Nombre de la tabla en tres partes. En conexión implícita con el servidor S2, PACKAGE PATH en el servidor S2 se hereda del servidor local. Las colecciones en PACKAGE PATH en el servidor S2 determinan qué paquete se invoca.

#### Notas:

1. Cuando CURRENT PACKAGE PATH se establece en el solicitante (y no en el servidor remoto), Db2 pasa una colección a la vez de la lista de colecciones al servidor remoto hasta que se encuentra un paquete o hasta el final de la lista. Cada vez que no se encuentra un paquete en el servidor, Db2 devuelve un error al solicitante. A continuación, el solicitante envía la siguiente colección de la lista al servidor remoto.

## Proceso de enlace para acceso remoto

Puede utilizar varios procesos de enlace diferentes para permitir el acceso a los datos en un servidor remoto.

Estos procesos funcionan cuando el servidor remoto es un sistema de base de datos relacional ( Db2 for z/OS ) u otro tipo de sistema de base de datos que utiliza acceso DRDA.

### Enviar un paquete al sitio local y al sitio remoto

Si aún no ha vinculado un paquete local, utilice esta técnica.

1. Envuelva el DBRM en un paquete en el sitio local.

2. Envuelva el DBRM en un paquete en el sitio remoto.
3. Enlazar un plan con una lista de paquetes que incluya el paquete local y el paquete remoto.

### **Ejemplo: Enlazar un paquete en el sitio local y en el sitio remoto**

Supongamos que usted precompiló el programa MYPROG para generar DBRM MYPROG, y compiló y editó el enlace del programa MYPROG. Desea ejecutar MYPROG para acceder a las tablas del sitio CHICAGO desde su sitio local. Utilice comandos como estos para vincular paquetes locales y remotos y un plan. El número al final de cada línea corresponde a un paso descrito anteriormente.

```
BIND PACKAGE(LOCALCOLLID) MEMBER(MYPROG) other bind options
BIND PACKAGE (CHICAGO.REMOTECOLLID) MEMBER(MYPROG) other bind options
BIND PLAN (MYPLAN) PKLIST(LOCALCOLLID.* ,*.REMOTECOLLID.*)
```

### **Enviar una copia de un paquete local existente al sitio remoto**

Si ya ha vinculado un paquete local, puede utilizar esta técnica.

1. Emitir el comando BIND con COPY y OPTIONS para crear una copia del paquete local en el sitio remoto.

OPTIONS controla los valores de las opciones de enlace que no especifique.

2. Enlazar un plan con una lista de paquetes que incluya el paquete local y el paquete remoto.

### **Ejemplo: Vincular una copia de un paquete local existente en el sitio remoto**

Supongamos que ha preparado previamente el programa MYPROG para su ejecución en el sitio local. Como parte de la preparación del programa, usted vinculó el paquete de información ( LOCALCOLLID.MYPROG ) en el sitio local. Ahora desea ejecutar MYPROG para acceder a las tablas del sitio CHICAGO desde su sitio local. Utilice comandos como estos para vincular una copia del paquete local en el sitio CHICAGO y, a continuación, vincular un plan. El número al final de cada línea corresponde a un paso descrito anteriormente.

```
BIND PACKAGE(CHICAGO.REMOTECOLLID) COPY(LOCALCOLLID.MYPROG) -
 OPTIONS(COMPOSITE|COMMAND) -
other bind options
BIND PLAN (MYPLAN) PKLIST(LOCALCOLLID.* ,*.REMOTECOLLID.*)
```

### **Opciones de vinculación para el acceso remoto**

El enlace de un paquete para que se ejecute en una ubicación remota es como vincular un paquete para que se ejecute en su subsistema Db2 local. La vinculación de un plan para ejecutar el paquete es similar a la vinculación de cualquier otro plan. Sin embargo, hay algunas diferencias.

Para las instrucciones generales, consulte Capítulo 9, “Preparación de una aplicación para su ejecución en Db2 for z/OS”, en la página 891.

### **Enlazar opciones de plan para acceso DRDA**

Las siguientes opciones de ENLACES DE PLAN son especialmente relevantes para enlazar un plan que utiliza acceso DRDA :

#### **DISCONNECT**

Para mayor flexibilidad, utilice DISCONNECT(EXPLICIT), explícitamente o por defecto. Para ello, debe utilizar sentencias RELEASE en su programa para finalizar las conexiones de forma explícita.

Los otros valores de la opción también son útiles.

**DISCONNECT(AUTOMATIC)** finaliza todas las conexiones remotas durante una operación de confirmación, sin necesidad de instrucciones RELEASE en su programa.

**DISCONNECT(conditional)** finaliza las conexiones remotas durante una operación de confirmación, excepto cuando un cursor abierto definido como WITH HOLD está asociado a la conexión.

#### **SQLRULES**

Utilice **SQLRULES(Db2)**, explícitamente o por defecto.

**SQLRULES(STD)** aplica las reglas del estándar SQL a sus sentencias CONNECT, de modo que CONNECT TO x es un error si ya está conectado a x. Utilice STD solo si desea que esa instrucción devuelva un código de error.

Si su programa selecciona datos LOB desde una ubicación remota y vincula el plan para el programa con SQLRULES(Db2), el formato en el que recupera los datos LOB con un cursor está restringido. Después de abrir el cursor para recuperar los datos LOB, debe recuperar todos los datos utilizando una variable LOB, o recuperar todos los datos utilizando una variable localizadora LOB. Si el valor de SQLRULES es STD, esta restricción no existe.

Si tiene la intención de alternar entre variables LOB y localizadores LOB para recuperar datos de un cursor, ejecute la instrucción SET SQLRULES=STD antes de conectarse a la ubicación remota.

#### **CURRENTDATA**

Utilice **CURRENTDATA(NO)** para forzar el bloqueo de la obtención de cursores ambiguos.

#### **ENCODING**

Utilice esta opción para controlar el esquema de codificación que se utiliza para las sentencias SQL estáticas en el plan y para establecer el valor inicial del registro especial CURRENT APPLICATION ENCODING SCHEME.

Para las aplicaciones que se ejecutan de forma remota y utilizan sentencias CONNECT explícitas, Db2 utiliza el valor ENCODING para el plan. Para las aplicaciones que se ejecutan de forma remota y utilizan sentencias CONNECT implícitas, Db2 utiliza el valor ENCODING para el paquete que se encuentra en el sitio donde se ejecuta una sentencia.

### **ENLACE Opciones de paquete para acceso DRDA**

Las siguientes opciones de ENLAZAR PAQUETE son relevantes para enlazar un paquete que se ejecutará utilizando acceso DRDA :

#### **nombre-ubicación**

Indique la ubicación del servidor en el que se ejecuta el paquete.

Los privilegios necesarios para ejecutar el paquete deben concederse al propietario del paquete en el servidor. Si no es el propietario, también debe tener la autoridad SYSCTRL o el privilegio BINDAGENT que se otorga localmente.

#### **SQLERROR**

Utilice **SQLERROR(CONTINUE)** si utilizó SQL(ALL) al precompilar. Esto crea un paquete incluso si el proceso de enlace encuentra errores SQL, como sentencias que son válidas en el servidor remoto pero que el precompilador no reconoció. De lo contrario, utilice SQLERROR(NOPACKAGE), explícitamente o por defecto.

#### **COPiar**

Si vincula con la opción COPY para copiar un paquete local a un sitio remoto, Db2 realiza la comprobación de autorización, lee y actualiza el catálogo, y crea el paquete en el sitio remoto. Db2 lee los registros del catálogo que están relacionados con el paquete copiado en el sitio local. Db2 convierte los valores devueltos por el sitio remoto en formato ISO si se cumplen todas las condiciones siguientes:

- Si el sitio local está instalado con formato de hora o fecha LOCAL
- Se crea un paquete en un sitio remoto con la opción COPY
- La instrucción SQL no especifica un formato diferente.

#### **CURRENTDATA**

Utilice **CURRENTDATA(NO)** para forzar el bloqueo de la obtención de cursores ambiguos.

## **OPTIONS**

Cuando realice una copia remota de un paquete utilizando BIND PACKAGE con la opción COPY, utilice esta opción para controlar las opciones de enlace predeterminadas que utiliza Db2 . Especifique:

**COMPOSITE** para hacer que Db2 utilice las opciones que especifique en el comando BIND PACKAGE. Para todas las demás opciones, Db2 utiliza las opciones del paquete copiado. COMPUESTO es el valor predeterminado.

**COMMAND** para hacer que Db2 utilice las opciones que especifique en el comando BIND PACKAGE. Para todas las demás opciones, Db2 utiliza los valores predeterminados para el servidor al que está vinculado el paquete. Esto ayuda a garantizar que el servidor admite las opciones con las que está vinculado el paquete.

## **ENCODING**

Utilice esta opción para controlar el esquema de codificación que se utiliza para las sentencias SQL estáticas en el paquete y para establecer el valor inicial del registro especial CURRENT APPLICATION ENCODING SCHEME.

Cuando vinculas el mismo paquete de forma local y remota, y especificas la opción de vinculación ENCODING para el paquete, la opción de vinculación ENCODING para el paquete local se aplica a la aplicación remota. El valor predeterminado de ENCODING para un paquete vinculado a un servidor de Db2 for z/OS remoto es el predeterminado del sistema para ese servidor. El valor predeterminado del sistema se especifica en el momento de la instalación en el campo APPLICATION ENCODING del panel DSNTIPF, que es el valor APPENSCH DECP.

## **EXPLAIN**

Si especifica la opción EXPLAIN(YES) o EXPLAIN(ONLY), y no especifica la opción SQLERROR(CONTINUE), PLAN\_TABLE debe existir en la ubicación remota a la que está vinculado el paquete.

### **Conceptos relacionados**

[Opciones de vinculación para bloqueos \(Db2 Performance\)](#)

### **Tareas relacionadas**

[Opciones de vinculación para aplicaciones distribuidas \(Db2 Performance\)](#)

### **Referencia relacionada**

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2 \)](#)

## **Consideraciones para vincular paquetes en una ubicación remota**

Cuando vincula paquetes en una ubicación remota, debe entender cómo el comportamiento de los paquetes remotos es diferente del comportamiento de los paquetes locales.

### **Alcance de una vinculación remota**

Cuando vinculas o revinculas un paquete en un sitio remoto, Db2 comprueba las autorizaciones, lee y actualiza el catálogo y crea el paquete en el directorio del sitio remoto. Db2 no lee ni actualiza catálogos ni comprueba autorizaciones en el sitio local.

Después de vincular un paquete remoto, puede vincular, revincular o liberar el paquete remoto desde el sitio local o en el sitio remoto.

### **Autorización para vincular y ejecutar paquetes en una ubicación remota**

Para vincular un paquete en un sistema de e Db2 encia remoto, debe tener todos los privilegios o la autoridad que necesitaría para vincular el paquete en su sistema local. Para vincular un paquete a otro tipo de sistema, como Db2 for Linux, UNIX, and Windows, necesita todos los privilegios que el otro sistema requiere para ejecutar las sentencias SQL en el paquete y para acceder a los objetos de datos a los que se refiere el paquete.

## **Requisitos de la base de datos de comunicaciones para paquetes vinculantes en una ubicación remota**

Cuando vinculas un paquete en un sitio remoto, la base de datos de comunicaciones local debe poder resolver el nombre de ubicación en el paquete a una ubicación remota.

### **Acceso remoto a través de un procedimiento almacenado**

Si un procedimiento almacenado local utiliza un cursor para acceder a los datos, y la instrucción relacionada con el cursor está vinculada en un paquete separado bajo el procedimiento almacenado, debe vincular este paquete separado tanto de forma local como remota. Además, el invocador o propietario del procedimiento almacenado debe estar autorizado para ejecutar paquetes tanto locales como remotos. En su sistema local solicitante, debe vincular un plan cuya lista de paquetes incluya todos esos paquetes locales y remotos.

## **Comprobación de las opciones de BIND PACKAGE que admite un servidor concreto**

Puede solicitar solo las opciones del comando BIND PACKAGE que sean compatibles con el servidor especificando dichas opciones en el solicitante.

### **Acerca de esta tarea**

Para saber qué opciones son compatibles con un sistema de gestión de bases de datos (DBMS) de servidor específico, consulte la documentación proporcionada para ese servidor.

Para obtener información específica sobre enlaces Db2 , consulte la siguiente documentación:

- Para obtener orientación sobre el uso de las opciones de enlace de Db2 y la realización de un proceso de enlace, consulte [Capítulo 9, “Preparación de una aplicación para su ejecución en Db2 for z/OS”, en la página 891](#).
- Para la sintaxis del comando BIND de Db2 , consulte los temas [BIND PACKAGE subcomando \(DSN\) \(Comandos de Db2\)](#) y [BIND PLAN subcomando \(DSN\) \(Comandos de Db2\)](#).
- Para la sintaxis del comando REBIND de Db2 , consulte los temas [REBIND PACKAGE subcomando \(DSN\) \(Comandos de Db2\)](#) y [REBIND PLAN \(Db2\)](#).

## **Enlazar un programa por lotes**

Antes de que un programa por lotes pueda emitir sentencias SQL, debe existir un plan de ejecución ( Db2 ).

### **Acerca de esta tarea**

El propietario del plan o paquete debe tener todos los privilegios necesarios para ejecutar las sentencias SQL integradas en él.

Puede especificar el nombre del plan a Db2 de una de las siguientes maneras:

- En el conjunto de datos de entrada de la e DDITV02.
- En la especificación de miembro del subsistema.
- Por defecto, el nombre del plan es el nombre del módulo de carga de la aplicación que se especifica en DDITV02.

Db2 pasa el nombre del plan al IMS adjuntar paquete. Si no especifica un nombre de plan en DDITV02 y no existe una tabla de traducción de recursos (RTT) o el nombre no está en la RTT, Db2 utiliza el nombre pasado como nombre del plan. Si el nombre existe en el RTT, el nombre se traduce al plan que se especifica para el RTT.

**Recomendación:** Dé al plan Db2 el mismo nombre que el del módulo de carga de la aplicación, que es la IMS configuración predeterminada del adjunto. El nombre del plan debe ser el mismo que el nombre del programa.

## Conversión de DBRM vinculados a un plan en DBRM vinculados a un paquete

Debe agrupar todos los DBRM en un paquete y agrupar los paquetes en un plan. Un paquete solo puede tener un DBRM.

La opción predeterminada REBIND PLAN COLLID (\*) convierte todos los planes con DBRM en planes con una lista de paquetes. Puede utilizar esta técnica solo para aplicaciones locales. Si el plan que especifique ya contiene tanto DBRM como listas de paquetes, las entradas de paquetes recién convertidas se insertarán al principio de la lista de paquetes existente.

**Importante:** Si el mismo DBRM está en varios planes y ejecutas REBIND PLAN con el mismo valor de opción COLLID en más de uno de esos planes, Db2 superpone el paquete creado anteriormente en la colección cada vez que ejecutas REBIND con COLLID. Para evitar superponer paquetes, especifique un valor COLLID diferente para cada plan que contenga DBRM que también estén en otros planes.

Para obtener más información sobre cómo desarrollar una estrategia para convertir sus planes de modo que incluyan solo paquetes, consulte [DB2 9 para z/OS : Paquetes revisados \( IBM Redbooks \)](#).

### Ejemplo: conversión de todos los planes

Los siguientes ejemplos convierten todos los DBRM vinculados al plan X en paquetes bajo el ID de colección: DSN\_DEFAULT\_COLLID\_X.

```
REBIND PLAN(X) COLLID(*);
```

### Ejemplo: especificar un ID de colección

Los siguientes ejemplos convierten los DBRM vinculados al plan X en paquetes con el ID de colección *my\_collection*.

```
REBIND PLAN(x) COLLID('my_collection');
```

### Ejemplo: reasignación de varios planes que pueden contener DBRM

En el siguiente ejemplo, BIND recorrerá cada plan especificado en la instrucción de comando REBIND PLAN y convertirá los DBRM en consecuencia, hasta que ninguno de los DBRM esté vinculado a planes.

```
REBIND PLAN (X1, X2, X3) COLLID (collection_id|*);
```

### Ejemplo: reasignación de todos los planes que puedan contener DBRM

En el siguiente ejemplo, BIND recorrerá todos los planes especificados en la tabla SYSPLAN y convertirá los DBRM en consecuencia, hasta que ninguno de los DBRM esté vinculado a planes.

```
REBIND PLAN (*) COLLID (collection_id|*);
```

### Ejemplo: especificación de una lista de paquetes

Los siguientes ejemplos convierten todos los DBRM vinculados al plan X en paquetes bajo el ID de colección: DSN\_DEFAULT\_COLLID\_X.

- Si el plan X no tiene una lista de paquetes, las entradas de paquetes recién convertidas se añadirán al principio de la lista de paquetes Z y, a continuación, la lista de paquetes Z se añadirá al plan X.
- Si el plan X tiene tanto una lista de paquetes como DBRM, las entradas de paquetes recién convertidas se añadirán al principio de la lista de paquetes Z y, a continuación, la lista de paquetes Z sustituirá a la lista de paquetes existente.

- Si el plan X solo tiene una lista de paquetes, la lista de paquetes Z sustituirá a la lista de paquetes existente.

```
REBIND PLAN (x) COLLID (collection_id|*) PKLIST(Z);
```

### Ejemplo: especificar que no hay lista de paquetes

Los siguientes ejemplos convierten todos los DBRM vinculados al plan X en paquetes bajo el ID de colección: DSN\_DEFAULT\_COLLID\_X.

- Si el plan X tiene tanto una lista de paquetes como DBRM, la lista de paquetes existente se eliminará y la nueva lista de paquetes se vinculará al plan X.
- Si el plan X solo tiene DBRM, los DBRM se convertirán en paquetes en consecuencia y se añadirán al plan X. La opción NOPKLIST se ignorará.
- Si el plan X no tiene DBRM, se eliminará la lista de paquetes existente, si la hay.

```
REBIND PLAN (x) COLLID (collection_id|*) NOPKLIST;
```

## Convertir un plan existente en paquetes para ejecutarlo de forma remota

Si tiene una aplicación existente que desea ejecutar en una ubicación remota mediante el acceso remoto, necesita un nuevo plan que incluya esos paquetes remotos en su lista de paquetes.

### Procedimiento

Para convertir un plan existente con DBRM de miembro en paquetes para ejecutarse de forma remota, realice las siguientes acciones para cada ubicación remota:

1. Elija un nombre para una colección que contenga DBRM de miembros, como REMOTE1.
2. Convertir el plan en un plan con una lista de paquetes.

```
REBIND PLAN(REMOTE1)COLLID(*)
```

Especificar COLLID(\*) produce los paquetes bajo la colección de DSN\_DEFAULT\_COLLID\_planname.

3. Consulte SYSIBM.SYSPACKDEP para ver si alguno de los paquetes tiene una dependencia de un alias. Ese alias es una definición para un nombre de 3 partes.
  - a) Para cada uno de los paquetes que tienen una dependencia de un alias:

```
BIND PACKAGE(location.remote_server_collid)
COPY(DSN_DEFAULT_COLLID_planname.pgkid)
COPYVER(...) OPTIONS(COMPOSITE)
```

4. Ajustar la lista de paquetes de la ubicación. Si antes de este proceso el plan no contaba con lista de paquetes, después [“2” en la página 943](#) tendrá una lista de paquetes que contiene DSN\_DEFAULT\_COLLID\_planname.pgkid.

```
REBIND PLAN PKLIST
(*.DSN_DEFAULT_COLLID_planname.pgkid* *.remote_server_collid.*)
```

## Resultados

Cuando ejecute la aplicación existente en su sistema local de Db2 utilizando el nuevo plan de aplicación, sucederán estas cosas:

- Se conecta inmediatamente a la ubicación remota que se indica en la opción CURRENTSERVER.
- Db2 busca el paquete en el REMOTE1 de recogida en la ubicación remota.
- Cualquier declaración de ACTUALIZAR, BORRAR o INSERTAR en su aplicación afecta a las tablas en la ubicación remota.

- Cualquier resultado de las sentencias SELECT se devuelve a su programa de aplicación existente, que los procesa como si provinieran de su sistema de Db2 local.

## Establecimiento del nivel de programa

El nivel de programa define el nivel para un módulo concreto. Esta información se almacena en la señal de consistencia, que está en un formato Db2 interno. La sustitución del nivel de programa en la señal de consistencia es posible, si es necesario, pero normalmente no se recomienda.

### Procedimiento

Utilice la opción NIVEL (*aaaa*).

Db2 utiliza el valor que elija para *aaaa* para generar el token de consistencia. Aunque este método no se recomienda para uso general y los paneles DSNH CLIST o de preparación del programa Db2 no lo admiten, este método le permite realizar las siguientes acciones:

- a. Cambiar el código fuente (pero no las sentencias SQL) en la salida del precompilador de Db2 de un programa vinculado.
- b. Compilar y editar el enlace del programa modificado.
- c. Ejecutar la aplicación sin volver a vincular un plan o paquete.

## Opciones de reglas dinámicas para sentencias de SQL dinámico

La opción de vinculación DYNAMICRULES y el entorno de ejecución determinan las reglas para los atributos de SQL dinámico.

La opción BIND o REBIND DYNAMICRULES determina qué valores se aplican en tiempo de ejecución para los siguientes atributos dinámicos de SQL:

- El ID de autorización que se utiliza para comprobar la autorización
- El calificador que se utiliza para objetos no calificados
- La fuente de las opciones de programación de aplicaciones que utiliza Db2 para analizar y verificar semánticamente las sentencias SQL dinámicas
- Indicación de si las sentencias de SQL dinámico pueden incluir las sentencias GRANT, REVOKE, ALTER, CREATE, DROP y RENAME

Además, el entorno de tiempo de ejecución de un paquete controla el comportamiento de las sentencias SQL dinámicas en tiempo de ejecución. Los dos posibles entornos de ejecución son:

- El paquete se ejecuta como parte de un programa independiente.
- El paquete se ejecuta como un procedimiento almacenado o un paquete de funciones definido por el usuario, o se ejecuta bajo un procedimiento almacenado o una función definida por el usuario.

Un paquete que se ejecuta bajo un procedimiento almacenado o una función definida por el usuario es un paquete cuyo programa asociado cumple una de las siguientes condiciones:

- El programa es llamado por un procedimiento almacenado o una función definida por el usuario.
- El programa está en una serie de llamadas anidadas que comienzan con un procedimiento almacenado o una función definida por el usuario.

## Comportamiento de la instrucción SQL dinámica

Los atributos SQL dinámicos que están determinados por el valor de la opción de enlace DYNAMICRULES y el entorno de tiempo de ejecución se denominan colectivamente *comportamiento de la instrucción SQL dinámica* o *comportamiento de las reglas dinámicas*. Las cuatro reglas dinámicas de comportamiento son: ejecutar, vincular, definir e invocar.

La siguiente tabla muestra la combinación del valor DYNAMICRULES y el entorno de tiempo de ejecución que producen cada comportamiento de SQL dinámico.

*Tabla 144. Forma en que DYNAMICRULES y el entorno de ejecución determinan el comportamiento de las sentencias de SQL dinámico*

Valor de DYNAMICRULES	Comportamiento de las sentencias de SQL dinámico	
	Entorno de programa independiente	Función definida por el usuario o entorno de procedimiento almacenado
EJECUTAR	Ejecute	Ejecute
BIND	Vincular	Vincular
DEFINERUN	Ejecute	Definir
DEFINEBIND	Vincular	Definir
INVOKERUN	Ejecute	Invocar
INVOKEBIND	Vincular	Invocar

**Nota:** Los valores BIND y RUN se pueden especificar para paquetes, planes y procedimientos SQL nativos. Los demás valores pueden especificarse para paquetes y procedimientos SQL nativos, pero no para planes.

La siguiente tabla muestra los valores de atributo SQL dinámico para cada tipo de comportamiento SQL dinámico.

*Tabla 145. Definiciones de comportamientos de sentencias SQL dinámicas*

Atributo SQL dinámico	Configuración de los atributos de comportamiento de SQL dinámico			
	Vincular	Ejecute	Definir	Invocar
ID de autorización	Propietario del plan o paquete	SQLID actual	Función definida por el usuario o propietario del procedimiento almacenado	ID de autorización de invoker1
Calificador predeterminado para objetos no calificados	Enlazar el valor PROPIETARIO o CALIFICADOR	CURRENT SCHEMA	Función definida por el usuario o propietario del procedimiento almacenado	ID de autorización del solicitante
ACTUAL SQLID2	No aplicable	Se aplica	No aplicable	No aplicable
Fuente de opciones de programación de aplicaciones	Determinado por DSNHDECP o un parámetro de módulo predeterminado de aplicación especificado por el usuario DYNRULS3	Instalar panel DSNTIP4	Determinado por DSNHDECP o un parámetro de módulo predeterminado de aplicación especificado por el usuario DYNRULS3	Determinado por DSNHDECP o un parámetro de módulo predeterminado de aplicación especificado por el usuario DYNRULS3
¿Puede ejecutar GRANT, REVOKE, CREATE, ALTER, DROP, RENAME?	Nee	Sí	Nee	Nee

Tabla 145. Definiciones de comportamientos de sentencias SQL dinámicas (continuación)

Atributo SQL dinámico	Configuración de los atributos de comportamiento de SQL dinámico			
	Vincular	Ejecutar	Definir	Invocar
<b>Notas:</b>				
1.	Si el invocador es el ID de autorización principal del proceso o el valor CURRENT SQLID, también se comprueban los ID de autorización secundarios si son necesarios para la autorización requerida. De lo contrario, solo se comprueba una identificación, la del solicitante, para la autorización requerida.			
2.	Db2 utiliza el valor de CURRENT SQLID como ID de autorización para sentencias SQL dinámicas solo para planes y paquetes que tienen comportamiento de ejecución. Para los demás comportamientos de SQL dinámico, Db2 utiliza el ID de autorización asociado a cada comportamiento de SQL dinámico, como se muestra en esta tabla.			
	El valor al que se inicializa CURRENT SQLID es independiente del comportamiento dinámico de SQL. Para los programas independientes, CURRENT SQLID se inicializa con el ID de autorización principal.			
	Puede ejecutar la instrucción SET CURRENT SQLID para cambiar el valor de CURRENT SQLID para paquetes con cualquier comportamiento SQL dinámico, pero Db2 utiliza el valor CURRENT SQLID solo para planes y paquetes con comportamiento de ejecución.			
3.	El valor de DSNHDECP o un parámetro de módulo predeterminado de aplicación especificado por el usuario DYNRULS, que se especifica en el campo USE FOR DYNAMICRULES en el panel de instalación DSNTIP4, determina si Db2 utiliza las opciones de procesamiento de instrucciones SQL o los valores predeterminados de programación de aplicaciones para instrucciones SQL dinámicas. Consulte “ <a href="#">Opciones para el proceso de sentencias de SQL</a> ” en la página 913 para obtener más información.			

#### Conceptos relacionados

[ID de autorización y SQL dinámico \(Db2 SQL\)](#)

[Comportamientos de autorización para sentencias de SQL dinámico \(Managing Security\)](#)

#### Referencia relacionada

[DYNAMICRULES opción bind \(comandos Db2 \)](#)

## Selección dinámica de planes

Es beneficioso utilizar la selección dinámica de planes y los paquetes juntos. Puede convertir programas individuales en una aplicación que contenga muchos programas y planes, uno a uno, para utilizar una combinación de planes y paquetes. Este proceso reduce el número de planes por aplicación; tener menos planes reduce el esfuerzo necesario para mantener la rutina dinámica de salida del plan.

**CICS** Puede utilizar paquetes y selección dinámica de planes juntos, pero cuando cambie dinámicamente de plan, deben darse las siguientes condiciones:

- Todos los registros especiales, incluido CURRENT PACKAGESET, deben contener sus valores iniciales.
- El valor en el registro especial CURRENT DEGREE no puede haber cambiado durante la transacción actual.

Suponga que desarrolla los siguientes programas y DBRM:

Tabla 146. Programas de ejemplo y DBRM

Nombre del programa	Nombre del DBRM
MAIN	MAIN
PROGA	PLANA
PROGRAMA	PKGB
PROGC	PLANC

Podría crear paquetes utilizando la siguiente instrucción bind:

```
BIND PACKAGE(PKGB) MEMBER(PKGB)
```

El siguiente escenario ilustra la asociación de hilos para una tarea que ejecuta el programa MAIN. Supongamos que ejecuta las siguientes sentencias SQL en el orden indicado. Para cada instrucción SQL, se describe el evento resultante.

1. EXEC CICS START TRANSID(MAIN)

TRANSID(MAIN) ejecuta el programa MAIN.

2. EXEC SQL SELECT...

El programa MAIN emite una instrucción SQL SELECT. La rutina de salida del plan dinámico predeterminado selecciona el plan MAIN.

3. EXEC CICS LINK PROGRAM(PROGA)

Se invoca el programa PROGA.

4. EXEC SQL SELECT...

Db2 no llama a la rutina de salida del plan dinámico predeterminado, porque el programa no emite un punto de sincronización. El plan es PRINCIPAL.

5. EXEC CICS LINK PROGRAM(PROGB)

Se invoca el programa PROGB.

6. EXEC SQL SELECT...

Db2 no llama a la rutina de salida del plan dinámico predeterminado, porque el programa no emite un punto de sincronización. El plan es MAIN y el programa utiliza el paquete PKGB.

7. EXEC CICS SYNCPOINT

Db2 llama a la rutina de salida del plan dinámico cuando se ejecuta la siguiente instrucción SQL.

8. EXEC CICS LINK PROGRAM(PROGC)

Se invoca el programa PROGC.

9. EXEC SQL SELECT...

Db2 llama a la rutina de salida del plan dinámico predeterminado y selecciona PLAN.C.

10. EXEC SQL SET CURRENT SQLID = 'ABC'

Al registro especial CURRENT SQLID se le asigna el valor 'ABC'

11. EXEC CICS SYNCPOINT

Db2 no llama a la rutina de salida del plan dinámico cuando se ejecuta la siguiente instrucción SQL porque la instrucción anterior modifica el registro especial CURRENT SQLID.

12. EXEC CICS RETURN

El control vuelve al programa PROGB.

13. EXEC SQL SELECT...

**CICS** Con los paquetes, probablemente no necesite la selección dinámica de planes y su rutina de salida asociada. No se accede a un paquete que figura en un plan hasta que se ejecuta. Sin embargo, puede utilizar la selección dinámica de planes y los paquetes juntos, lo que puede reducir el número de planes en una aplicación y el esfuerzo para mantener la rutina dinámica de salida del plan.

## Revinculación de aplicaciones

Debe volver a enlazar las aplicaciones para cambiar las opciones de enlace. También es necesario volver a enlazar las aplicaciones cuando realice cambios que afecten al plan o al paquete, como por ejemplo la creación de un índice, pero no haya cambiado las sentencias SQL.

## Acerca de esta tarea

En algunos casos, Db2 vuelve a vincular automáticamente los planes o paquetes por usted, dependiendo del valor del parámetro del subsistema ABIND. Para obtener detalles, consulte “[Revinculación automática](#)” en la página 957.

Las siguientes acciones pueden requerir que vuelva a vincular un paquete:

- Cambiar el idioma del host o las sentencias SQL en la aplicación. Debe reemplazar el paquete. Precompilar, compilar y vincular el programa de aplicación. A continuación, emita un comando BIND con la opción ACTION(REPLACE).
- Cambiar los atributos de sus datos de manera que invaliden el paquete. Para obtener detalles, consulte “[Cambios que invalidan paquetes](#)” en la página 16.
- Mejorar la selección de rutas de acceso después de reorganizar datos con la utilidad REORG o recopilar estadísticas de bases de datos con RUNSTATS u otras utilidades. Para obtener más información, consulte [Mantenimiento de estadísticas y organización de datos \(Db2 Performance\)](#).
- Permitir que Db2 seleccione una ruta de acceso que utilice un índice recién creado para acceder a una tabla.
- Cambiar las opciones de encuadernación de un paquete. Si una opción que desea cambiar no está disponible para el comando REBIND, emita el comando BIND con ACTION(REPLACE) en su lugar.
- Preparación para la migración a una nueva versión de Db2 . Para obtener más información, consulte [Vuelva a vincular los planes y paquetes antiguos en Db2 11 para evitar vinculaciones automáticas perjudiciales en Db2 12 \(Instalación y migración Db2 \)](#).

FL 505 Db2 , con *la fase de reasignación activada*, puede reasignar un paquete al mismo tiempo que lo ejecuta. Una operación de reimpresión crea una nueva copia del paquete. Cuando finaliza la operación de reenlace, las nuevas hebras pueden utilizar la nueva copia de paquete inmediatamente y las hebras existentes pueden continuar utilizando la copia que estaba en uso antes del reenlace (la copia de eliminación gradual) sin interrupciones. Para más información, consulte “[Introducción gradual de los reenvíos de paquetes](#)” en la página 950.

### Tareas relacionadas

[Identificación de paquetes con características que afectan al rendimiento, la simultaneidad o la capacidad de ejecución \(Rendimiento de Db2\)](#)

### Referencia relacionada

[AUTO BIND \(parámetro del subsistemaABIND \) \( Db2 Instalación y migración\)](#)

[REBIND PACKAGE subcomando \(DSN\) \(Comandos de Db2 \)](#)

[BIND PACKAGE subcomando \(DSN\) \(Comandos de Db2 \)](#)

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2 \)](#)

## Revinculación de un paquete

Debe volver a vincular un paquete cuando realiza cambios que afectan al paquete pero que no implican cambios en las sentencias SQL. Por ejemplo, si crea un índice nuevo, debe volver a vincular el paquete. Si cambia el SQL, debe utilizar el mandato BIND PACKAGE con la opción ACTION(REPLACE).

### Antes de empezar

Para un paquete de activación, utilice el subcomando REBIND TRIGGER PACAKGE. Para obtener más información, consulte “[Paquetes desencadenantes](#)” en la página 172.

### Procedimiento

Utilice el subcomando REBIND PACKAGE.

Puede cambiar cualquiera de las opciones de encuadernación de un paquete cuando lo vuelva a encuadernar.

La siguiente tabla aclara qué paquetes se vinculan, dependiendo de cómo especifique *el id. de colección* (coll-id), *el id. de paquete* (pkg-id) y *el id. de versión* (ver-id) en el subcomando REBIND PACKAGE.

REENLazar PAQUETE no se aplica a paquetes para los que no tienes el privilegio de ENLazar. Un asterisco (\*) utilizado como identificador de colecciones, paquetes o versiones no se aplica a los paquetes en sitios remotos.

*Tabla 147. Comportamiento de la especificación REBIND PACKAGE.* «Todas» significa todas las colecciones, paquetes o versiones en el servidor local Db2 para las que el ID de autorización que emite el comando tiene el privilegio BIND.

Entrada	Colecciones afectadas	Paquetes afectados	Versiónes afectadas
*	Todos	Todos	Todos
*.*.(*)	Todos	Todos	Todos
*.*	Todos	Todos	Todos
*.*.(ver-id)	Todos	Todos	ver-id
*.*.()	Todos	Todos	serie vacía
coll-id.*	coll-id	Todos	Todos
coll-id.*.(*)	coll-id	Todos	Todos
coll-id.*.(ver-id)	coll-id	Todos	ver-id
coll-id.*.()	coll-id	Todos	serie vacía
coll-id.pkg-id.(*)	coll-id	pkg-id	Todos
coll-id.pkg-id	coll-id	pkg-id	serie vacía
coll-id.pkg-id.()	coll-id	pkg-id	serie vacía
coll-id.pkg-id.(ver-id)	coll-id	pkg-id	ver-id
*.pkg-id.(*)	Todos	pkg-id	Todos
*.pkg-id	Todos	pkg-id	serie vacía
*.pkg-id.()	Todos	pkg-id	serie vacía
*.pkg-id.(ver-id)	Todos	pkg-id	ver-id

## **ejemplos**

### **Ejemplo: reempaquetar un paquete en una ubicación remota**

El siguiente ejemplo muestra las opciones para reempaquetar un paquete en la ubicación remota. El nombre de la ubicación es SNTERSA. La colección es GROUP1, el ID del paquete es PROGA y el ID de la versión es V1. Los tipos de conexión mostrados en el subcomando REBIND sustituyen a los tipos de conexión especificados en el subcomando BIND original.

```
REBIND PACKAGE(SNTERSA.GROUP1.PROGA.(V1)) ENABLE(CICS,REMOTE)
```

### **Ejemplo: Reencuadernación de todos los paquetes locales**

Puede utilizar el asterisco en el subcomando REBIND para paquetes locales, pero no para paquetes en sitios remotos. Cualquiera de los siguientes comandos vuelve a vincular todas las versiones de todos los paquetes en todas las colecciones, en el sistema local Db2 , para el que tiene el privilegio BIND.

```
REBIND PACKAGE (*)
REBIND PACKAGE (*.*)
REBIND PACKAGE (*.*.*)
```

### **Ejemplo: Reencuadernación de todas las versiones de todos los paquetes locales**

Cualquiera de los siguientes comandos vuelve a vincular todas las versiones de todos los paquetes en la colección local LEDGER para los que tiene el privilegio BIND.

```
REBIND PACKAGE (LEDGER.*)
```

```
REBIND PACKAGE (LEDGER.*.*)
```

### **Ejemplo: Reencuadernación de paquetes locales en todas las colecciones**

Cualquiera de los siguientes comandos vuelve a vincular la versión de cadena vacía del paquete DEBIT en todas las colecciones, en el sistema local Db2 , para el que tiene el privilegio BIND.

```
REBIND PACKAGE (*.DEBIT)
```

```
REBIND PACKAGE (*.DEBIT.())
```

## **Tareas relacionadas**

[Reutilización y comparación de las vías de acceso durante la vinculación y la revinculación \(Db2 Performance\)](#)

## **Referencia relacionada**

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2 \)](#)

[REBIND PACKAGE subcomando \(DSN\) \(Comandos de Db2 \)](#)

## **Introducción gradual de los reenvíos de paquetes**

Db2 , con la fase de reasignación activada, puede reasignar un paquete al mismo tiempo que lo ejecuta. Una operación de reimpresión crea una nueva copia del paquete. Cuando finaliza la operación de reenlace, las nuevas hebras pueden utilizar la nueva copia de paquete inmediatamente y las hebras existentes pueden continuar utilizando la copia que estaba en uso antes del reenlace (la copia de eliminación gradual) sin interrupciones.

### **FL 505**

Con *rebind phase-in*, una operación REBIND PACKAGE genera una nueva copia, mientras que los hilos existentes continúan ejecutando la copia actual del paquete, que se convierte en la copia eliminada. Cuando el mandato REBIND confirma la nueva copia, dicha copia se convierte en la copia actual y está inmediatamente disponible para la siguiente ejecución mediante una nueva hebra. Las hebras que existían antes de REBIND también pueden utilizar la nueva copia actual cuando liberan la copia de eliminación gradual (basada en la opción RELEASE(COMMIT) o RELEASE(DEALLOCATION)).

**Consejo:** APAR PH28693 (enero de 2021) mejora la concurrencia para los comandos REBIND en Db2 12 en el nivel de función 505 o superior. Con este APAR, un comando REBIND ahora siempre obtiene un bloqueo U, lo que permite que las transacciones posteriores que están ejecutando un paquete se ejecuten en paralelo. Para más información, consulte [Mejora de los tiempos de ejecución de transacciones y concurrencia para REBIND PACKAGE](#).

Db2 puede generar hasta 14 copias para la fase transitoria de nuevas copias de paquetes en REBIND. Solo una copia es la actual y su ID de copia está en la tabla de catálogo SYSPACKAGE. Todas las copias de eliminación, y las copias originales y anteriores, se almacenan en SYSPACKCOPY y en otras tablas de catálogo hasta que se suprimen. Sin embargo, el copyID 3 es solo para uso interno y no se almacena en las tablas de catálogo SYSPACKCOPY o SYSPACKAGE.

En las ejecuciones posteriores del mandato REBIND PACKAGE, Db2 detecta cuándo se puede suprimir de forma segura una copia de eliminación gradual y su ID de copia se puede reutilizar. El ID de copia máximo es 16.

Cuando todos los ID de copia disponibles están en uso, es posible que un mandato REBIND posterior no se ejecute correctamente y emita el mensaje DSNT500I, con el código de razón 00E30307. Este mandato anómalo indica una hebra que impide que se reutilicen una o varias copias de eliminación gradual.

Puede utilizar la opción PLANMGMTSCOPE(PHASEOUT) en el submandato FREE PACKAGE para liberar las copias de eliminadas gradualmente no utilizadas, que se pueden crear cuando se revincule un paquete. Se recomienda liberar las copias del paquete de eliminación gradual para reducir el espacio en el directorio y el catálogo de Db2. El submandato liberará las copias de eliminación gradual que no utilizan actualmente una hebra de ejecución. El valor de la columna SYSPACKCOPY.TIMESTAMP se puede utilizar para determinar cuándo se ha eliminado la copia de una copia. Además, la opción PLANMGMTSCOPE (INACTIVE) en el submandato FREE PACKAGE también se ocupa de las copias eliminadas gradualmente.

La fase transitoria de revinculación está soportada para las siguientes opciones:

- APREUSE(NONE) PLANMGMT(EXTENDED)
- APREUSE(WARN) PLANMGMT(EXTENDED) APREUSESOURCE(CURRENT)
- APREUSE(ERROR) PLANMGMT(EXTENDED) APREUSESOURCE(CURRENT)
- El paquete no es un paquete generado para una rutina de desencadenante o SQL, como un procedimiento o una función definida por el usuario.

Cuando se especifica PLANMGMTSCOPE(PHASEOUT), las copias del paquete eliminado gradualmente se liberan, independientemente de si también se especifica la opción INVALIDONLY(YES).

## Conceptos relacionados

[Copias de paquetes para la gestión de planes \(Rendimiento de Db2\)](#)

### Revinculación automática

*Las reasociaciones automáticas* (a veces llamadas "autobindings") se producen cuando un usuario autorizado ejecuta un paquete o plan y las estructuras de tiempo de ejecución del plan o paquete no se pueden utilizar. Esta situación se da normalmente cuando se modifican los atributos de los datos de los que depende el paquete o plan, o si se modifica el entorno en el que se ejecuta el paquete o plan.

### Tareas relacionadas

#### Revinculación de aplicaciones

Debe volver a enlazar las aplicaciones para cambiar las opciones de enlace. También es necesario volver a enlazar las aplicaciones cuando realice cambios que afecten al plan o al paquete, como por ejemplo la creación de un índice, pero no haya cambiado las sentencias SQL.

### Referencia relacionada

[REBIND PACKAGE subcomando \(DSN\) \(Comandos de Db2 \)](#)

## Volver a encuadrinar un plan

Debe volver a vincular un plan cuando realice un cambio en uno de los atributos del plan, como la lista de paquetes.

## Procedimiento

Utilice el subcomando REBIND PLAN.

Puede cambiar cualquiera de las opciones de enlace para ese plan.

Cuando vuelva a vincular un plan, utilice la palabra clave PKLIST para reemplazar cualquier lista de paquetes especificada anteriormente. Omite la palabra clave PKLIST para utilizar la lista de paquetes anterior para la reasignación. Utilice la palabra clave NOPKLIST para eliminar cualquier lista de paquetes que se haya especificado cuando el plan se vinculó anteriormente.

### ejemplos

#### Ejemplo

El siguiente comando vuelve a vincular PLANA y cambia la lista de paquetes:

```
REBIND PLAN(PLANA) PKLIST(GROUP1.*) MEMBER(ABC)
```

#### Ejemplo

El siguiente comando vuelve a vincular el plan y suprime toda la lista de paquetes:

```
REBIND PLAN(PLANA) NOPKLIST
```

### Referencia relacionada

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2 \)](#)

[REBIND PLAN \(Db2 \)](#)

## Reencuadernación de listas de planes y paquetes

En algunas situaciones, es necesario volver a vincular un conjunto de planes o paquetes que no se pueden describir mediante asteriscos. Por ejemplo, si finaliza una operación de reasociación, puede generar un subcomando de reasociación para cada objeto que no se haya asociado.

### Acerca de esta tarea

Una situación en la que esta técnica resulta útil es para completar una operación de reencuadernación que ha finalizado por falta de recursos. Una reasignación para muchos objetos, como el PAQUETE DE REASIGNACIÓN (\*) para un ID con autoridad SYSADM, finaliza si un recurso necesario deja de estar disponible. Como resultado, algunos objetos se devuelven correctamente y otros no. Si repite el subcomando, Db2 intenta volver a vincular todos los objetos de nuevo. Pero si genera un subcomando de reasignación para cada objeto que no se reasignó y emite esos subcomandos, Db2 no repite ningún trabajo que ya se haya realizado y no es probable que se quede sin recursos.

Para una descripción de la técnica y varios ejemplos de su uso, consulte [“Programa de muestra para crear subcomandos REBIND para listas de planes y paquetes”](#) en la página 953.

## Generación de listas de comandos REBIND

Para generar una lista de subcomandos REBIND para un conjunto de paquetes que no se pueden describir, utilice asteriscos y la información del catálogo Db2 . A continuación, puede emitir la lista de subcomandos a través de DSN.

### Acerca de esta tarea

La siguiente lista es un resumen de los procedimientos para el PAQUETE REBIND:

1. Utilice DSNTIAUL para generar los subcomandos REBIND PACKAGE para los paquetes seleccionados.
2. Utilice DSNTEDIT CLIST para eliminar los espacios en blanco superfluos de los subcomandos REBIND PACKAGE.
3. Utilice los comandos de edición TSO para añadir comandos DSN al conjunto de datos secuenciales.

4. Utilice DSN para ejecutar los subcomandos REBIND PACKAGE para los paquetes seleccionados.

## Programa de muestra para crear subcomandos REBIND para listas de planes y paquetes

Si no puede utilizar asteriscos para identificar una lista de paquetes o planes que desea reasignar, es posible que pueda crear automáticamente los subcomandos REBIND necesarios utilizando el programa de ejemplo DSNTIAUL.

Una situación en la que esta técnica podría ser útil es cuando un recurso deja de estar disponible durante una reasignación de muchos planes o paquetes. Db2 normalmente finaliza la reasignación y no reasigna los planes o paquetes restantes. Más adelante, sin embargo, es posible que desee reasignar únicamente los objetos que quedan por reasignar. Puede crear subcomandos REBIND para los planes o paquetes restantes utilizando DSNTIAUL para seleccionar los planes o paquetes del catálogo Db2 y crear los subcomandos REBIND. A continuación, puede enviar los subcomandos a través del procesador de comandos DSN, como de costumbre.

Es posible que primero tengas que editar el resultado de DSNTIAUL para que DSN pueda aceptarlo como entrada. El CLIST DSNTEDIT puede realizar gran parte de esa tarea por usted.

Esta sección contiene los siguientes temas:

- [“Generación de listas de comandos REBIND” en la página 952](#)
- [“Ejemplos de sentencias SELECT para generar comandos REBIND” en la página 953](#)
- [“JCL de ejemplo para ejecutar listas de mandatos REBIND” en la página 955](#)

## Ejemplos de sentencias SELECT para generar comandos REBIND

Puede seleccionar planes o paquetes específicos para reasignar y concatenar la sintaxis del subcomando REBIND alrededor de los nombres de los planes o paquetes. También puede convertir una cadena de longitud variable en una cadena de longitud fija y añadir espacios en blanco adicionales a los subcomandos REBIND PLAN y REBIND PACKAGE, de modo que el procesador de comandos DSN pueda aceptar la longitud del registro como entrada válida.

**Creación de subcomandos REBIND :** Los ejemplos que siguen ilustran las siguientes técnicas:

- Usar SELECT para seleccionar paquetes o planes específicos que se van a reenviar
- Uso del operador CONCAT para concatenar la sintaxis del subcomando REBIND alrededor de los nombres de planes o paquetes
- Uso de la función SUBSTR para convertir una cadena de longitud variable en una cadena de longitud fija
- Añadir espacios en blanco adicionales a los subcomandos REBIND PLAN y REBIND PACKAGE, para que el procesador de comandos DSN pueda aceptar la longitud del registro como entrada válida

**Si la instrucción SELECT devuelve filas,** DSNTIAUL genera subcomandos REBIND para los planes o paquetes identificados en las filas devueltas. Ponga esos subcomandos en un conjunto de datos secuencial, donde luego podrá editarlos.

Para los subcomandos REBIND PACKAGE, elimine los espacios en blanco superfluos en el nombre del paquete, utilizando los comandos de edición TSO o el CLIST DSNTEDIT de Db2 .

Para los subcomandos REBIND PLAN y REBIND PACKAGE, agregue el comando DSN que necesita la instrucción como primera línea en el conjunto de datos secuenciales, y agregue END como última línea, utilizando comandos de edición TSO. Cuando haya editado el conjunto de datos secuenciales, puede ejecutarlo para volver a vincular los planes o paquetes seleccionados.

**Si la instrucción SELECT no devuelve ninguna fila que cumpla los requisitos,** DSNTIAUL no genera subcomandos REBIND.

Los ejemplos de este tema generan subcomandos REBIND que funcionan en Db2 for z/OS Db2 12. Es posible que tenga que modificar los ejemplos para versiones anteriores de Db2 que no permiten la misma sintaxis.

**Ejemplo: REASIGNAR todos los planes sin cancelar debido a la falta de recursos.**

```
SELECT SUBSTR('REBIND PLAN('CONCAT NAME
 CONCAT')',1,45)
FROM SYSIBM.SYSPLAN;
```

**Ejemplo: VUELVA A ENLazar todas las versiones de todos los paquetes sin terminar debido a la falta de recursos.**

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.
 CONCAT NAME CONCAT').(*))',1,55)
FROM SYSIBM.SYSPACKAGE;
```

**Ejemplo: VUELVA A ENCUADERNAR todos los planes encuadrernados antes de una fecha y hora determinadas.**

```
SELECT SUBSTR('REBIND PLAN('CONCAT NAME
 CONCAT')',1,45)
FROM SYSIBM.SYSPLAN
WHERE BINDDATE <= 'aammdd' OR
 (BINDDATE <= 'aammdd' AND
 BINDTIME <= 'hhmmssth');
```

donde *aammdd* representa la parte de fecha y *hhmmssth* representa la parte de hora de la cadena de marca de tiempo.

Si la fecha especificada es posterior al año 2000, debe incluir otra condición que incluya planes que se hayan vinculado antes del año 2000:

```
WHERE
 BINDDATE >= '830101' OR
 BINDDATE <= 'aammdd' OR
 (BINDDATE <= 'aammdd' AND
 BINDTIME <= 'hhmmssth');
```

**Ejemplo: REENLACE todas las versiones de todos los paquetes enlazados antes de una fecha y hora determinadas.**

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.
 CONCAT NAME CONCAT').(*))',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE BINDTIME <= 'timestamp';
```

donde *timestamp* es una cadena de marca de tiempo ISO.

**Ejemplo: VUELVA A ENLACAR todos los planes enlazados desde una fecha y hora determinadas.**

```
SELECT SUBSTR('REBIND PLAN('CONCAT NAME
 CONCAT')',1,45)
FROM SYSIBM.SYSPLAN
WHERE BINDDATE >= 'aammdd' AND
 BINDTIME >= 'hhmmssth';
```

donde *aammdd* representa la parte de fecha y *hhmmssth* representa la parte de hora de la cadena de marca de tiempo.

**Ejemplo: REENLACE todas las versiones de todos los paquetes enlazados desde una fecha y hora determinadas.**

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID
 CONCAT'.CONCAT NAME
 CONCAT'.(*))',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE BINDTIME >= 'timestamp';
```

donde *timestamp* es una cadena de marca de tiempo ISO.

**Ejemplo: VUELVA A ENLACAR todos los planes enlazados dentro de un intervalo de fecha y hora determinado.**

```
SELECT SUBSTR('REBIND PLAN('CONCAT NAME
CONCAT')',1,45)
FROM SYSIBM.SYSPLAN
WHERE
(BINDDATE >= 'yymmdd' AND
BINDTIME >= 'hhmmss') AND
BINDDATE <= 'yymmdd' AND
BINDTIME <= 'hhmmss');
```

donde *aammdd* representa la parte de fecha y *hhmmss* representa la parte de hora de la cadena de marca de tiempo.

**Ejemplo: VUELVA A ENCUADERNAR todas las versiones de todos los paquetes encuadrados dentro de un intervalo de fecha y hora determinado.**

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.
CONCAT NAME CONCAT').(*)',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE BINDTIME >= 'timestamp1' AND
BINDTIME <= 'timestamp2';
```

donde *timestamp1* y *timestamp2* son cadenas de marca de tiempo ISO.

**Ejemplo: VUELVA A ENMARCAR todas las versiones no válidas de todos los paquetes.**

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.
CONCAT NAME CONCAT').(*)',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE VALID = 'N';
```

**Ejemplo: VUELVA A ENCUADERNAR todos los planos encuadrados con el nivel de ESTABILIDAD DEL CURSOR ISOLACIÓN.**

```
SELECT SUBSTR('REBIND PLAN('CONCAT NAME
CONCAT')',1,45)
FROM SYSIBM.SYSPLAN
WHERE ISOLATION = 'S';
```

**Ejemplo: Vuelva a enlazar todas las versiones de todos los paquetes que permitan el paralelismo de la CPU y/o E/S.**

```
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.
CONCAT NAME CONCAT').(*)',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE DEGREE='ANY';
```

## JCL de ejemplo para ejecutar listas de mandatos REBIND

Puede utilizar JCL para volver a enlazar todas las versiones de todos los paquetes enlazados dentro de un periodo de fecha y hora especificado.

Debe especificar la fecha y el periodo de tiempo para los que desea que se vuelvan a enlazar los paquetes en una cláusula WHERE de la instrucción SELECT que contiene el comando REBIND. En el siguiente ejemplo, la cláusula WHERE tiene el siguiente aspecto:

```
WHERE BINDTIME >= 'YYYY-MM-DD-hh.mm.ss' AND
BINDTIME <= 'YYYY-MM-DD-hh.mm.ss'
```

La fecha y el período de tiempo tienen el siguiente formato:

**YYYY**

El año de cuatro dígitos. Por ejemplo: 2008.

**MM**

El mes de dos dígitos, que puede ser un valor entre 01 y 12.

**DD**

El día de dos dígitos, que puede ser un valor entre 01 y 31.

**hh**

La hora de dos dígitos, que puede ser un valor entre 01 y 24.

**mm.**

El minuto de dos dígitos, que puede ser un valor entre 00 y 59.

**ss**

El segundo de dos dígitos, que puede ser un valor entre 00 y 59.

```
//REBINDS JOB MSGLEVEL=(1,1),CLASS=A,MSGCLASS=A,USER=SYSADM,
// REGION=1024K
//*****
//SETUP EXEC PGM=IKJEFT01
//SYSTSPPRT DD SYSOUT=*

//SYSTSIN DD *
// DSN SYSTEM(DSN)
RUN PROGRAM(DSNTIAUL) PLAN(DSNTIBC1) PARM('SQL') -
LIB('DSN1210.RUNLIB.LOAD')
END
//SYSPRINT DD SYSOUT=*

//SYSUDUMP DD SYSOUT=*

//SYSPUNCH DD SYSOUT=*

//SYSREC00 DD DSN=SYSADM.SYSTSIN.DATA,
// UNIT=SYSDA,DISP=SHR

//*****
/*
/* GENER= '<SUBCOMMANDS TO REBIND ALL PACKAGES BOUND IN YYYY
/*
//*****
//SYSIN DD *
SELECT SUBSTR('REBIND PACKAGE('CONCAT COLLID CONCAT'.
CONCAT NAME CONCAT'.(*) ',1,55)
FROM SYSIBM.SYSPACKAGE
WHERE BINDTIME >= 'YYYY-MM-DD-hh.mm.ss' AND
 BINDTIME <= 'YYYY-MM-DD-hh.mm.ss';
/*
//*****
/* STRIP THE BLANKS OUT OF THE REBIND SUBCOMMANDS
/*
//*****
//STRIP EXEC PGM=IKJEFT01
//SYSPROC DD DSN=SYSADM.DSNCLIST,DISP=SHR
//SYSTSPPRT DD SYSOUT=*

//SYSPRINT DD SYSOUT=*

//SYSOUT DD SYSOUT=*

//SYSTSIN DD *
// DSNTEDIT SYSADM.SYSTSIN.DATA
//SYSIN DD DUMMY
/*
//*****
/* PUT IN THE DSN COMMAND STATEMENTS
/*
//*****
//EDIT EXEC PGM=IKJEFT01
//SYSTSPPRT DD SYSOUT=*

//SYSTSIN DD *
// EDIT 'SYSADM.SYSTSIN.DATA' DATA NONUM
TOP
INSERT DSN SYSTEM(DSN)
BOTTOM
INSERT END
TOP
LIST * 99999
END SAVE
/*

//*****
/* EXECUTE THE REBIND PACKAGE SUBCOMMANDS THROUGH DSN
/*
//*****
```

```

//LOCAL EXEC PGM=IKJEFT01
//DBRMLIB DD DSN=DSN1210.DBRMLIB.DATA,
// DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD DSN=SYSADM.SYSTSIN.DATA,
// UNIT=SYSDA,DISP=SHR
/*

```

El siguiente ejemplo muestra una muestra de JCL para reencuadernar todos los planes encuadrados sin especificar la palabra clave DEGREE en BIND con DEGREE(ANY).

```

//REBINDS JOB MSGLEVEL=(1,1),CLASS=A,MSGCLASS=A,USER=SYSADM,
// REGION=1024K
//***** *****/
//SETUP EXEC TSOBATCH
//SYSPRINT DD SYSOUT=*
//SYSPUNCH DD SYSOUT=*
//SYSREC00 DD DSN=SYSADM.SYSTSIN.DATA,
// UNIT=SYSDA,DISP=SHR
//***** *****/
///*
//** REBIND ALL PLANS THAT WERE BOUND WITHOUT SPECIFYING THE DEGREE
//** KEYWORD ON BIND WITH DEGREE(ANY)
///*
//***** *****/
//SYSTSIN DD *
 DSN S(DSN)
 RUN PROGRAM(DSNTIAUL) PLAN(DSNTIBC1) PARM('SQL')
 END
//SYSIN DD *
SELECT SUBSTR('REBIND PLAN('CONCAT NAME
 CONCAT') DEGREE(ANY)' ,1,45)
 FROM SYSIBM.SYSPLAN
 WHERE DEGREE = ' ';
/*
//***** *****/
///*
//** PUT IN THE DSN COMMAND STATEMENTS
///*
//***** *****/
//EDIT EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 EDIT 'SYSADM.SYSTSIN.DATA' DATA NONUM
 TOP
 INSERT DSN S(DSN)
 BOTTOM
 INSERT END
 TOP
 LIST * 99999
 END SAVE
/*
//***** *****/
///*
//** EXECUTE THE REBIND SUBCOMMANDS THROUGH DSN
///*
//***** *****/
//REBIND EXEC PGM=IKJEFT01
//STEPLIB DD DSN=SYSADM.TESTLIB,DISP=SHR
// DD DSN=DSN1210.SDSNLOAD,DISP=SHR
//DBRMLIB DD DSN=SYSADM.DBRMLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSTSIN DD DSN=SYSADM.SYSTSIN.DATA,DISP=SHR
//SYSIN DD DUMMY
/*

```

## Revinculación automática

*Las reasociaciones automáticas* (a veces llamadas "autobindings") se producen cuando un usuario autorizado ejecuta un paquete o plan y las estructuras de tiempo de ejecución del plan o paquete no se pueden utilizar. Esta situación se da normalmente cuando se modifican los atributos de los datos de los que depende el paquete o plan, o si se modifica el entorno en el que se ejecuta el paquete o plan.

Para obtener una lista de acciones que podrían hacer que Db2 marque los paquetes como no válidos, consulte [“Cambios que invalidan paquetes” en la página 16](#).

En la mayoría de los casos, Db2 marca un paquete que debe ser automáticamente rebotado como *inválido* estableciendo VALID='N' en las tablas de catálogo SYSIBM.SYSPLAN y SYSIBM.SYSPACKAGE.

Si falla una reasociación automática, Db2 marca un paquete como *inoperativo* en la columna OPERATIVE de las tablas de catálogo SYSIBM.SYSPLAN y SYSIBM.SYSPACKAGE. Sin embargo, si la fase de enlace automático falla para un paquete que se invalida a nivel de extracto, se utiliza OPERATIVE='R' solo en la tabla SYSPACKAGE.

## Controles para enlaces automáticos

Db2 utiliza enlaces automáticos solo cuando el parámetro del subsistema ABIND está establecido en YES o COEXIST (Db2 12 utiliza el mismo comportamiento para ambas configuraciones). Si ABIND está establecido en NO cuando se ejecuta un paquete no válido, Db2 devuelve un error. Para obtener detalles, consulte [AUTO BIND \(parámetro del subsistemaABIND \) \( Db2 Instalación y migración\)](#).

También puede utilizar tablas de límites de recursos para controlar los enlaces automáticos. Para obtener detalles, consulte [Restricción de operaciones de vinculación \(Db2 Performance\)](#).

## Opciones de vinculación para vinculaciones automáticas

En general, Db2 utiliza las mismas opciones de enlace del proceso de enlace más reciente para los enlaces automáticos. Existen las siguientes excepciones:

- Si una opción ya no es compatible, el proceso de opción de reasignación automática sustituye una opción compatible.
- Si una opción no tiene un valor existente, se utiliza la opción de enlace predeterminada.
- El valor de reasignación automática para APCOMPARE es NINGUNO.
- El valor de reasignación automática para APREUSE es WARN, y el valor de reasignación automática para APREUSESOURCE es CURRENT.
- Si no existe ningún valor para la opción de enlace APPLCOMPAT, se utiliza el parámetro del subsistema APPLCOMPAT.
- Si no existe ningún valor para la opción de enlace DESCSTAT, se utiliza el parámetro del subsistema DESCSTAT.

## Se vincula automáticamente con las copias del paquete

Si un paquete tiene copias anteriores u originales como resultado de la reasignación con las opciones PLANMGMT(BASIC) o PLANMGMT(EXTENDED) o de tener el parámetro del subsistema PLANMGMT establecido en BASIC o EXTENDED, esas copias no se ven afectadas por la reasignación automática. La reimpresión automática reemplaza solo la copia actual.

Puede darse una situación en la que la reasociación automática haga que la copia anterior u original tenga una versión de Db2 más reciente que la copia actual. Supongamos que la copia A es la copia actual y la copia B es la copia anterior. La copia A está en una versión anterior y compatible para paquetes de Db2 , pero la copia B está en una versión de Db2 anterior a la versión mínima compatible. Cuando cambias los paquetes para que la copia B se convierta en la copia actual y ejecutas la copia B, Db2 vuelve a vincular automáticamente la copia B. Ahora, la copia B está en una versión más reciente de Db2 que la copia A.

## Cuando fallan los enlaces automáticos

Cuando falla un enlace automático, Db2 emite el mensaje DSNT500I a la consola con el motivo '00E30305 ' x, el tipo de recurso '804 ' x y el nombre de recurso *collection.package*. (version).

Si se especificó EXPLAIN(YES) para la operación de reasociación anterior, el parámetro del subsistema ABEXP controla si Db2 captura la información EXPLAIN durante las reasociaciones automáticas. Para

obtener detalles, consulte [EXPLAIN PROCESSING](#) (parámetro del subsistema ABEXP) (Db2 Instalación y migración). Los reasignados automáticos fallan para la mayoría de los errores EXPLAIN.

Si se produce una vinculación automática mientras se ejecuta en modo ACCESS(MAINT), la vinculación automática se ejecuta bajo el id de autorización de SYSOPR. Si SYSOPR no está definido como un SYSOPR de instalación, el enlace automático falla.

### Conceptos relacionados

[Vínculos automáticos en coexistencia \(Db2 Installation and Migration\)](#)

[Incompatibilidades de SQL y aplicaciones entre releases \(Novedades de DB2 para z/OS\)](#)

### Tareas relacionadas

[Vuelva a vincular los planes y paquetes antiguos en Db2 11 para evitar vinculaciones automáticas perjudiciales en Db2 12 \(Instalación y migración Db2\)](#)

### Referencia relacionada

[BIND PACKAGE subcomando \(DSN\) \(Comandos de Db2\)](#)

[BIND PLAN subcomando \(DSN\) \(Comandos de Db2\)](#)

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2\)](#)

### Información relacionada

[DSNT500I \(Db2 Messages\)](#)

[00E30305 \(Db2 Codes\)](#)

## Especificación de las reglas que se aplican a un comportamiento SQL en tiempo de ejecución

Puede especificar si las reglas de Db2 o las reglas SQL estándar se aplican a un comportamiento SQL en tiempo de ejecución.

### Acerca de esta tarea

SQLRULES no solo especifica las reglas bajo las cuales se ejecuta una instrucción CONNECT de tipo 2, sino que también establece el valor inicial del registro especial CURRENT RULES cuando el servidor de la base de datos es el sistema local de Db2 . Cuando el servidor no es el sistema local Db2 , el valor inicial de CURRENT RULES es DB2. Después de vincular un plan, puede cambiar el valor en CURRENT RULES en un programa de aplicación utilizando la instrucción SET CURRENT RULES.

CURRENT RULES determina las reglas SQL, Db2 o estándar SQL, que se aplican al comportamiento de SQL en tiempo de ejecución. Por ejemplo, el valor en CURRENT RULES afecta al comportamiento de la definición de restricciones de verificación al emitir la instrucción ALTER TABLE en una tabla poblada:

- Si **CURRENT RULES tiene un valor de STD** y no hay filas existentes en la tabla que violen la restricción de verificación, Db2 añade la restricción a la definición de la tabla. De lo contrario, se produce un error y Db2 no añade la restricción de verificación a la definición de la tabla.

Si la tabla contiene datos y ya está en estado de verificación pendiente, la instrucción ALTER TABLE falla.

- Si **CURRENT RULES tiene un valor de DB2**, Db2 añade la restricción a la definición de la tabla, aplaza la aplicación de las restricciones de comprobación y coloca el espacio de tabla o la partición en estado CHECK-pending.

Puede utilizar la sentencia SET CURRENT RULES para controlar la acción que realiza la sentencia ALTER TABLE. Suponiendo que el valor de CURRENT RULES es inicialmente STD, las siguientes sentencias SQL cambian las reglas SQL a DB2, añaden una restricción de verificación, aplazan la validación de esa restricción, colocan la tabla en estado CHECK-pending y restauran las reglas a STD.

```
EXEC SQL
 SET CURRENT RULES = 'DB2';
EXEC SQL
 ALTER TABLE DSN8C10.EMP
 ADD CONSTRAINT C1 CHECK (BONUS <= 1000.0);
```

```
EXEC SQL
 SET CURRENT RULES = 'STD';
```

Visite “[Restricciones de comprobación](#)” en la página 137 para obtener información sobre las restricciones de cheques.

También puede utilizar CURRENT RULES en las asignaciones de variables de host. Por ejemplo, si desea almacenar el valor del registro especial CURRENT RULES en un momento determinado, puede asignar el valor a una variable host, como en la siguiente declaración:

```
SET :XRULE = CURRENT RULES;
```

También puede utilizar REGLAS ACTUALES como argumento de una condición de búsqueda. Por ejemplo, la siguiente sentencia recupera filas en las que la columna " COL1 " contiene el mismo valor que el registro especial CURRENT RULES.

```
SELECT * FROM SAMPTBL WHERE COL1 = CURRENT RULES;
```

## Conjuntos de datos de entrada y salida para trabajos por lotes DL/I

Los trabajos por lotes DL/I requieren un conjunto de datos de entrada con el nombre de definición de datos DDITV02 y un conjunto de datos de salida con el nombre de definición de datos DDOTV02.

### Db2 Entrada por lotes DL/I:

Antes de poder ejecutar un trabajo por lotes DL/I, debe proporcionar valores para una serie de parámetros de entrada. Los parámetros de entrada son posicionales y están delimitados por comas.

Puede especificar valores para los siguientes parámetros utilizando un conjunto de datos de la interfaz de programación de aplicaciones ( DDITV02, API) o un miembro del subsistema:

```
SSN,LIT,ESMT,RTT,REO,CRC
```

Puede especificar valores para los siguientes parámetros **solo** en un conjunto de datos de tipo " DDITV02 ":

```
CONNECTION_NAME,PLAN,PROG
```

Si utiliza el conjunto de datos DDITV02 y especifica un miembro del subsistema, los valores de la instrucción DD DDITV02 anulan los valores del miembro del subsistema especificado. Si no proporciona ninguno de los dos, Db2 finaliza de forma anómala el programa de aplicación con el código de error del sistema X'04E' y un código de motivo único en el registro 15.

DDITV02 es el nombre DD para un conjunto de datos que tiene opciones DCB de LRECL=80 y RECFM=F o FB.

Un miembro de subsistema es un miembro de la IMS biblioteca de procedimientos. Su nombre se deriva de la concatenación del valor del parámetro SSM con el valor del parámetro IMSID. Usted especifica el parámetro SSM y el parámetro IMSID cuando invoca el procedimiento DLIBATCH, que inicia el entorno de procesamiento por lotes DL/I.

Los significados de los parámetros de entrada son:

### NSS

Especifica el nombre del subsistema de la e Db2 . Este valor es necesario. Debe especificar un nombre para poder conectarse a Db2.

El valor del SSN puede tener de uno a cuatro caracteres.

Si el valor del parámetro SSN es el nombre de un subsistema activo en el grupo de intercambio de datos, la aplicación se conecta a ese subsistema. Si el valor del parámetro SSN no es el nombre de un subsistema activo, sino el nombre de un archivo adjunto de grupo, la aplicación se adjunta a un subsistema de intercambio de datos ( Db2 ) activo en el grupo de intercambio de datos.

**LIT**

Especifica un token de interfaz de idioma. Db2 requiere un token de interfaz de idioma para enrutar las instrucciones SQL cuando se opera en el entorno en línea IMS. Debido a que un programa de aplicación por lotes solo puede conectarse a un sistema de gestión de inventario (Db2), Db2 no utiliza el valor LIT.

El valor LIT puede tener de cero a cuatro caracteres de longitud.

**Recomendación :** Especifique el valor LIT como SYS1.

Puede omitir el valor LIT introduciendo SSN,,ESMT.

**ESMT**

Especifica el nombre del módulo de inicialización de Db2 , DSNMIN10. Este valor es necesario.

El valor ESMT debe tener ocho caracteres.

**RTT**

Especifica la tabla de traducción de recursos. Esta tarea es opcional.

El RTT puede tener de cero a ocho caracteres.

**REO**

Especifica la opción de error de región. Esta opción determina qué hacer si Db2 no está operativo o el plan no está disponible. Las tres opciones son:

- *R*, el valor predeterminado, da como resultado un código de retorno SQL para el programa de aplicación. El SQLCODE más común emitido en este caso es -923 (SQLSTATE '57015').
- *Q* da como resultado un error en el entorno por lotes; sin embargo, en el entorno en línea, este valor vuelve a colocar el mensaje de entrada en la cola.
- *Un error* provoca un fallo tanto en el entorno por lotes como en el entorno en línea.

Si el programa de aplicación utiliza la llamada XRST y si se requiere una recuperación coordinada en la llamada XRST, se ignora REO. En ese caso, el programa de aplicación se cierra de forma anormal si Db2 no está operativo.

El valor REO puede tener de cero a un carácter de longitud.

**CRC**

Especifica el carácter de reconocimiento de comandos. Debido a que los comandos de Db2 no son compatibles con el entorno de lotes DL/I, el carácter de reconocimiento de comandos no se utiliza en este momento.

El valor CRC puede tener de cero a un carácter de longitud.

**CONNECTION\_NAME**

Representa el nombre del paso de trabajo que coordina las actividades de e Db2 . Esta tarea es opcional. Si no especifica esta opción, los nombres de conexión predeterminados son:

**Tipo de aplicación****Nombre de conexión predeterminado****Trabajo por lotes**

Nombre de trabajo

**tarea iniciada**

Nombre de la tarea iniciada

**Usuario de TSO**

ID de autorización de TSO

Si falla una tarea de actualización por lotes, debe utilizar una tarea independiente para reiniciar la tarea por lotes. El nombre de conexión utilizado en el trabajo de reinicio debe ser el mismo que el nombre utilizado en el trabajo por lotes que falló. De forma alternativa, si se utiliza el nombre de conexión predeterminado, la tarea de reinicio debe tener el mismo nombre de tarea que la tarea de actualización por lotes que falló.

Db2 requiere nombres de conexión únicos. Si dos aplicaciones intentan conectarse con el mismo nombre de conexión, el segundo programa de aplicación no puede conectarse a Db2.

El valor CONNECTION\_NAME puede tener entre uno y ocho caracteres.

#### PLAN

Especifica el nombre del plan de e Db2 . Esta tarea es opcional. Si no especifica el nombre del plan, el nombre del módulo del programa de aplicación se compara con la tabla de traducción de recursos opcional. Si la tabla de traducción de recursos tiene una coincidencia, el nombre traducido se utiliza como nombre del plan e Db2 . Si no existe ninguna coincidencia en la tabla de traducción de recursos, el nombre del módulo del programa de aplicación se utiliza como nombre del plan.

El valor del PLAN puede tener entre cero y ocho caracteres.

#### PROG

Especifica el nombre del programa de aplicación. Este valor es necesario. Identifica el programa de aplicación que se va a cargar y que recibirá el control.

El valor PROG puede tener entre uno y ocho caracteres.

**Ejemplo :** A continuación se muestra un ejemplo de los campos del registro:

```
DSN,SYS1,DSNMIN10,,R,-,BATCH001,DB2PLAN,PROGA
```

#### Db2 Salida por lotes DL/I:

En un entorno en línea IMS, Db2 envía mensajes de estado no solicitados al operador de terminal maestro (MTO) y registra en el registro la información de procesamiento y diagnóstico de indoubt IMS registro.

En un entorno por lotes, Db2 envía esta información al conjunto de datos de salida que se especifica en la sentencia DD ( DDOTV02 ). Asegúrese de que el conjunto de datos de salida tenga opciones DCB de RECFM=V o VB, LRECL=4092, y BLKSIZE de al menos LRECL + 4. Si falta el extracto de DD, Db2 emite el mensaje « IEC130I » y continúa el procesamiento sin ningún resultado.

Es posible que desee guardar e imprimir el conjunto de datos, ya que la información es útil para fines de diagnóstico. Puede utilizar el IMS módulo, DFSERA10, para imprimir los registros del conjunto de datos de longitud variable en formato hexadecimal y de caracteres.

#### Conceptos relacionados

[Envío de trabajo para procesar \(Db2 Data Sharing Planning and Administration\)](#)

## Procedimientos JCL proporcionados por Db2 para la preparación de una aplicación

Puede precompilar y preparar un programa de aplicación utilizando un procedimiento JCL proporcionado por Db2.

Db2 tiene un procedimiento JCL único para cada lenguaje soportado, con los valores predeterminados adecuados para iniciar el precompilador de Db2 y el compilador o ensamblador del lenguaje anfitrión. Los procedimientos están en *prefijo.SDSNSAMP* miembro DSNTIJMP, que instala los procedimientos.

*Tabla 148. Procedimientos para la precompilación de programas*

Idioma	Procedimiento	Invocación incluida en.
High-Level Assembler	DSNHASM	DSNTEJ2A
C	DSNHC	DSNTEJ2D
C++	DSNHCPP	DSNTEJ2E
C++	DSNHCPP22	DSNTEJ6V
Enterprise COBOL	DSNHICOB	DSNTEJ2C1
Fortran	DSNHFOR	DSNTEJ2F

Tabla 148. Procedimientos para la precompilación de programas (continuación)

Idioma	Procedimiento	Invocación incluida en.
PL/I	DSNHPLI	DSNTEJ2P
SQL	DSNHSQ	DSNTEJ63

**Notas:**

1. Debe personalizar estos programas para invocar los procedimientos que se enumeran en esta tabla.
2. Este procedimiento muestra cómo se puede preparar un programa orientado a objetos que consta de dos conjuntos de datos o miembros, los cuales contienen SQL.

Si utiliza el procesador de macros PL/I, no debe utilizar la instrucción PL/I \*PROCESS en el código fuente para pasar opciones al compilador PL/I. Puede especificar las opciones necesarias en el parámetro PARM.PLI = de la instrucción EXEC en el procedimiento DSNHPLI.

## JCL para incluir el código de interfaz adecuado al utilizar los procedimientos JCL suministrados por Db2

Para incluir el código de interfaz adecuado al enviar los procedimientos JCL, utilice una instrucción INCLUDE SYSLIB en su JCL de edición de enlaces. La declaración debe especificar el módulo de interfaz de idioma correcto para el entorno.

### TSO, lote

```
//LKED.SYSIN DD *
 INCLUDE SYSLIB(member)
/*
```

el *miembro* debe ser DSNELI o DSNULI, excepto para FORTRAN, en cuyo caso el *miembro* debe ser DSNHFT.

### IMS

```
//LKED.SYSIN DD *
 INCLUDE SYSLIB(DFSLI000)
 ENTRY (specification)
/*
```

DFSLI000 es el módulo para adjuntar lotes DL/I.

La *especificación* de ENTRADA varía en función del idioma de destino. Incluya uno de los siguientes:

DLITCBL, para aplicaciones COBOL

PLICALLA, para aplicaciones PL/I

El nombre del programa, para aplicaciones de lenguaje ensamblador.

**Recomendación:** Para aplicaciones COBOL, especifique el enlace PSB directamente en la instrucción PROCEDURE DIVISION en lugar de en un punto de entrada DLITCBL. Cuando especifique el enlace PSB directamente en la sentencia PROCEDURE DIVISION, puede omitir la especificación ENTRY o especificar el nombre del programa de aplicación en lugar del punto de entrada DLITCBL.

### CICS

```
//LKED.SYSIN DD *
 INCLUDE SYSLIB(member)
/*
```

el *miembro* debe ser DSNCLI o DSNULI.

## Conceptos relacionados

[“Interfaz de lenguaje universal \(DSNULLI\)” en la página 121](#)

El subcomponente de la interfaz de lenguaje universal (DSNULLI) determina el entorno en tiempo de ejecución y carga y se bifurca dinámicamente al modelo de interfaz de lenguaje adecuado.

## Tareas relacionadas

[Disponibilidad de la interfaz de lenguaje CAF \(DSNALI\)](#)

Antes de invocar el recurso de conexión de llamada (CAF), primero debe hacer que DSNALI esté disponible.

[Compilación y edición de enlaces de una aplicación](#)

Si utiliza el Db2 coprocessor, procesa las sentencias SQL a medida que compila su programa, y el siguiente paso es el enlace de edición del programa. El propósito del paso de edición de enlaces es producir un módulo de carga ejecutable.

## Adaptación de los procedimientos JCL suministrados por Db2, para la preparación de CICS programas

En lugar de utilizar los paneles de preparación del programa de Db2 para preparar su CICS programa, puede adaptar CICS -los procedimientos JCL suministrados para hacerlo. Para personalizar un CICS procedimiento, debe añadir algunos pasos y cambiar algunas declaraciones DD.

### Acerca de esta tarea

Realice los cambios necesarios para llevar a cabo las siguientes acciones:

- Procesar el programa con el precompilador Db2 .
- Encuadernar el plan de aplicación. Puede hacerlo en cualquier momento después de precompilar el programa. Puede vincular el programa en línea mediante los paneles de DB2I o como un paso por lotes en este u otro z/OS tarea.
- Incluya una declaración DD en el paso del editor de enlaces para acceder a la biblioteca de carga de Db2 .
- Asegúrese de que las sentencias de control del editor de enlaces contienen una sentencia INCLUDE para el módulo de interfaz de lenguaje de programación ( Db2 ).

El siguiente ejemplo ilustra los cambios necesarios. Este ejemplo asume el uso de un programa COBOL. Para cualquier otro lenguaje de programación, cambie el CICS nombre del procedimiento y las opciones del precompilador Db2 .

```
//TESTC01 JOB
//*
//*****DB2 PRECOMPILE THE COBOL PROGRAM*****
//(1) //PC EXEC PGM=DSNHPC,
//(1) // PARM='HOST(COB2),XREF,SOURCE,FLAG(I),APOST'
//(1) //STEPLIB DD DISP=SHR,DSN=prefix.SDSNEXIT
//(1) // DD DISP=SHR,DSN=prefix.SDSNLOAD
//(1) //DBRMLIB DD DISP=OLD,DSN=USER.DBRMLIB.DATA(TESTC01)
//(1) //SYSCIN DD DSN=&DSNHOUT,DISP=(MOD,PASS),UNIT=SYSDA,
//(1) // SPACE=(800,(500,500))
//(1) //SYSLIB DD DISP=SHR,DSN=USER.SRCLIB.DATA
//(1) //SYSPRINT DD SYSOUT=*
//(1) //SYSTEMR DD SYSOUT=*
//(1) //SYSUDUMP DD SYSOUT=*
//(1) //SYSUT1 DD SPACE=(800,(500,500),,ROUND),UNIT=SYSDA
//(1) //SYSUT2 DD SPACE=(800,(500,500),,ROUND),UNIT=SYSDA
//(1) //SYSIN DD DISP=SHR,DSN=USER.SRCLIB.DATA(TESTC01)
//(1) //*
//*****BIND THIS PROGRAM.*****
//(2) //BIND EXEC PGM=IKJEFT01,
//(2) // COND=((4,LT,PC))
```

```

(2) //STEPLIB DD DISP=SHR,DSN=prefix.SDSNEXIT
(2) // DD DISP=SHR,DSN=prefix.SDSNLOAD
(2) //DBRMLIB DD DISP=OLD,DSN=USER.DBRMLIB.DATA(TESTC01)
(2) //SYSPRINT DD SYSOUT=*
(2) //SYSTSPRT DD SYSOUT=*
(2) //SYSUDUMP DD SYSOUT=*
(2) //SYSTSIN DD *
(2) DSN S(DSN)
(2) BIND PLAN(TESTC01) MEMBER(TESTC01) ACTION(REP) RETAIN ISOLATION(CS)
(2) END
//*****COMPILE THE COBOL PROGRAM*****
//*****CICS EXEC DFHEITVL
(3) //CICS EXEC DFHEITVL
(4) //TRN.SYSIN DD DSN=&&DSNHOUT,DISP=(OLD,DELETE)
(5) //LKED.SYSLMOD DD DSN=USER.RUNLIB.LOAD
(6) //LKED.CICSLOAD DD DISP=SHR,DSN=prefix.SDFHLOAD
(7) //LKED.SYSIN DD *
(7) INCLUDE CICSLOAD(DSNCLI)
(7) NAME TESTC01(R)
//*****

```

El procedimiento tiene en cuenta estos pasos:

**Paso 1.** Precompilar el programa. La salida del precompilador de Db2 se convierte en la entrada del CICS traductor de lenguaje de comandos.

**Paso 2.** Encuadrinar el plan de aplicación.

**Paso 3.** Llame al CICS procedimiento para traducir, compilar y editar enlaces de un programa COBOL. Este procedimiento tiene varias opciones que debe tener en cuenta.

**Paso 4.** Reflejar una biblioteca de carga de aplicaciones en el nombre del conjunto de datos de la sentencia DD SYSLMOD. Debe incluir el nombre de esta biblioteca de carga en la instrucción DFHRPL DD del CICS jCL de tiempo de ejecución.

**Paso 5.** Nombre de la CICS biblioteca de carga que contiene el módulo DSNCLI.

**Paso 6.** Indique al editor de enlaces que incluya el CICS -Db2 módulo de interfaz de idioma (DSNCLI). En este ejemplo, el orden de las distintas secciones de control (CSECT) no es relevante porque la estructura del procedimiento satisface automáticamente cualquier requisito de orden.

Para más información sobre el procedimiento DFHEITVL, otros CICS procedimientos o CICS requisitos para los programas de aplicación, consulte el manual correspondiente CICS manual.

Si está preparando una solicitud especialmente grande o compleja, puede utilizar otro método de preparación. Por ejemplo, si su programa requiere cuatro de sus propias bibliotecas de inclusión de edición de enlaces, no puede preparar el programa con DB2I, porque DB2I limita el número de bibliotecas de inclusión a tres, más el lenguaje, IMS o CICS, y bibliotecas Db2 . Por lo tanto, necesitaría otro método de preparación. **Tenga cuidado de utilizar la interfaz de idioma correcta.**

#### Referencia relacionada

[Conjuntos de datos que utiliza el precompilador](#)

Cuando invoca el precompilador, necesita proporcionar conjuntos de datos que contienen entrada para el precompilador, como las sentencias de programación de host o sentencias SQL. También necesita proporcionar conjuntos de datos donde el precompilador pueda almacenar la salida, como el código de origen modificado o los mensajes de diagnóstico.

## Paneles de DB2I que se utilizan para la preparación del programa

DB2I contiene un conjunto de paneles que le permite preparar una aplicación para su ejecución.

La siguiente tabla describe cada uno de los paneles que necesita utilizar para preparar una solicitud.

Tabla 149. DB2I paneles utilizados para la preparación del programa

Nombre del panel	Descripción del panel
<a href="#">“Panel Preparación del programa de Db2” en la página 967</a>	Le permite elegir funciones específicas de preparación de programas para realizar. Para las funciones que elija, también puede mostrar los paneles asociados para especificar las opciones para realizar esas funciones.  Este panel también le permite cambiar los valores predeterminados de DB2I y realizar otras funciones de precompilación y preenlace.
<a href="#">“Panel de valores predeterminados 1 de DB2I” en la página 971</a>	Le permite cambiar muchos de los valores predeterminados del sistema que se establecen en el momento de la instalación de Db2 .
<a href="#">“Panel 2 de valores predeterminados de DB2I” en la página 973</a>	Le permite cambiar su declaración de trabajo predeterminada y establecer opciones COBOL adicionales.
<a href="#">“Panel Precompilar” en la página 974</a>	Le permite especificar valores para funciones de precompilación.  Puede acceder a este panel directamente desde el menú de opciones principales de DB2I o desde el panel de preparación del programa de Db2 . Si llega a este panel desde el panel Preparación del programa, muchos de los campos contienen valores de los paneles Principal y Precompilación.
<a href="#">“Panel Enlazar paquete” en la página 976</a>	Le permite cambiar muchas opciones cuando vincula un paquete.  Puede acceder a este panel directamente desde el menú de opciones principales de DB2I o desde el panel de preparación del programa de Db2 . Si llega a este panel desde el panel de Preparación del programa ( Db2 ), muchos de los campos contienen valores de los paneles Principal (Primary) y Precompilación (Precompile).
<a href="#">“Panel Enlazar plan” en la página 979</a>	Le permite cambiar las opciones cuando vincula un plan de aplicación.  Puede acceder a este panel directamente desde el menú de opciones principal de DB2I o como parte del proceso de preparación del programa. Este panel también sigue los paneles del paquete de enlace.
<a href="#">“Paneles Valores predeterminados para BIND PACKAGE y Valores predeterminados REBIND PACKAGE” en la página 982</a>	Permitirle cambiar los valores predeterminados para ENLAZAR o REENLAZAR PAQUETE o PLAN.
<a href="#">“Panel Tipos de conexión del sistema” en la página 986</a>	Le permite especificar un tipo de conexión del sistema.  Este panel se muestra si elige habilitar o deshabilitar las conexiones en los paneles Enlazar o Volver a enlazar paquete o Plan.
<a href="#">“Paneles para entrar listas de valores” en la página 988</a>	Permite introducir o modificar un número ilimitado de valores. Un panel de lista se parece a una sesión de edición de e ISPF s y le permite desplazarse y utilizar un conjunto limitado de comandos.

Tabla 149. DB2I paneles utilizados para la preparación del programa (continuación)

Nombre del panel	Descripción del panel
<a href="#">"Preparación del programa: Panel Compilar, Enlazar y Ejecutar" en la página 989</a>	Le permite realizar los dos últimos pasos del proceso de preparación del programa (compilar y editar vínculos). Este panel también le permite hacer la MACRO FASE PL/I para programas que requieren esta opción. Para los programas TSO, el panel también le permite ejecutar programas.

#### Referencia relacionada

[El menú de la opción primaria de DB2I \(Introducción a Db2 for z/OS\)](#)

[Paneles de DB2I que se utilizan para volver a vincular y liberar planes y paquetes](#)

Un conjunto de paneles de DB2I le permite enlazar, volver a enlazar o liberar paquetes.

## Panel Preparación del programa de Db2

El panel Preparación del programa de Db2 le permite elegir qué función de preparación del programa específica desea realizar.

Para las funciones que elija, también puede optar por mostrar los paneles asociados para especificar las opciones para realizar esas funciones. Algunas de las funciones que puede seleccionar son:

#### Precompilar

El panel de esta función le permite controlar el precompilador de Db2 .

#### Envolver un paquete

El panel de esta función le permite vincular el DBRM de su programa a un paquete y cambiar sus valores predeterminados para vincular los paquetes.

#### Enviar un plan

El panel de esta función le permite crear el plan de aplicación de su programa y cambiar sus valores predeterminados para vincular los planes.

#### Recopilar, vincular y ejecutar

El panel para estas funciones le permite controlar el compilador o ensamblador y el editor de enlaces.

**TSO y lote:** Para los programas TSO, puede utilizar los programas de preparación de programas para controlar el procesador de tiempo de ejecución del lenguaje host y el propio programa.

El panel de preparación del programa también le permite cambiar los valores predeterminados de DB2I y realizar otras funciones de precompilación y preenlace.

En el panel de preparación del programa ( Db2 ), que se muestra en la siguiente figura, introduzca el nombre del conjunto de datos del programa de origen (en este ejemplo se utiliza SAMPLEPG.COBOL ) y especifique las demás opciones que desee incluir. Cuando termine, pulse INTRO para ver el siguiente panel.

```

DSNEPP01 DB2 PROGRAM PREPARATION SSID: DSN
COMMAND ==>_

Enter the following:
1 INPUT DATA SET NAME ==> SAMPLEPG.COBOL
2 DATA SET NAME QUALIFIER ==> TEMP (For building data set names)
3 PREPARATION ENVIRONMENT ==> FOREGROUND (FOREGROUND, BACKGROUND, EDITJCL)
4 RUN TIME ENVIRONMENT ... ==> TSO (TSO, CAF, CICS, IMS, RRSAF)
5 OTHER DSNH OPTIONS ==>

Select functions: Display panel? (Optional DSNH keywords)
 Perform function?
6 CHANGE DEFAULTS ==> Y (Y/N) ==> N (Y/N)
7 PL/I MACRO PHASE ==> N (Y/N) ==> Y (Y/N)
8 PRECOMPILE ==> Y (Y/N) ==> N (Y/N)
9 CICS COMMAND TRANSLATION
10 BIND PACKAGE ==> Y (Y/N) ==> Y (Y/N)
11 BIND PLAN..... ==> Y (Y/N) ==> Y (Y/N)
12 COMPILE OR ASSEMBLE ==> Y (Y/N) ==> Y (Y/N)
13 PRELINK..... ==> N (Y/N) ==> N (Y/N)
14 LINK..... ==> N (Y/N) ==> Y (Y/N)
15 RUN..... ==> N (Y/N) ==> Y (Y/N)

```

Figura 46. El panel de preparación del programa "Db2" (Programa de Becas de la Fundación Tiffany)

A continuación se explican las funciones del panel de preparación del programa (Db2) y cómo completar los campos necesarios para iniciar la preparación del programa.

### 1 ENTRADA DE DATOS NOMBRE DEL CONJUNTO

Le permite especificar el nombre del conjunto de datos de entrada. El nombre del conjunto de datos de entrada puede ser un PDS o un conjunto de datos secuencial, y también puede incluir un nombre de miembro. Si no incluye el nombre del conjunto de datos entre comillas, un prefijo TSO estándar (ID de usuario) califica el nombre del conjunto de datos.

El nombre del conjunto de datos de entrada que especifique se utiliza para precompilar, vincular, editar enlaces y ejecutar el programa.

### 2 CALIFICADOR DEL NOMBRE DEL CONJUNTO DE DATOS

Le permite calificar nombres de conjuntos de datos temporales involucrados en el proceso de preparación del programa. Utilice cualquier cadena de caracteres de 1 a 8 caracteres que se ajuste a las convenciones de nomenclatura normales de TSO. (El valor predeterminado es TEMP.)

Para los programas que se preparan en segundo plano o que utilizan EDITJCL para la opción PREPARATION ENVIRONMENT (ENTORNO DE PREPARACIÓN), Db2 crea un conjunto de datos denominado *tsoprefix.qualifier.CNTL* para contener el JCL de preparación del programa. El nombre *tsoprefix* representa el prefijo que asigna TSO, y *el calificador* representa el valor que introduce en el campo CALIFICADOR DE NOMBRE DE CONJUNTO DE DATOS. Si ya existe un conjunto de datos con este nombre, Db2 lo elimina.

### 3. ENTORNO DE PREPARACIÓN

Le permite especificar si la preparación del programa se produce en primer plano o en segundo plano. También puede especificar EDITJCL, en cuyo caso podrá editar y luego enviar el trabajo. Utilice:

FOREGROUND para utilizar los valores que especifique en el panel de Preparación del programa y ejecutarlo inmediatamente.

ANTECEDENTES para crear y enviar un archivo que contenga un DSNH CLIST que se ejecute inmediatamente utilizando la instrucción de control JOB desde el panel de valores predeterminados de DB2I, o desde la salida SUBMIT de su sitio. El archivo se guarda.

EDITJCL para crear y abrir un archivo que contenga un DSNH CLIST en modo de edición. A continuación, puede enviar el CLIST o guardarlo.

### 4. ENTORNO DE EJECUCIÓN

Le permite especificar el entorno (TSO, CAF, CICS,, RRSAF) en el que se ejecuta su programa IMS, RRSAF) en el que se ejecuta su programa.

Todos los programas se preparan en TSO, pero pueden ejecutarse en cualquiera de los entornos. Si especifica CICS, IMS o RRSAF, debe establecer el campo RUN en NO porque no puede ejecutar dichos

programas desde el panel de Preparación de programas. Si establece el campo RUN en YES, solo puede especificar TSO o CAF.

(Los programas por lotes también se ejecutan en el programa TSO Terminal Monitor). Por lo tanto, debe especificar TSO en este campo para los programas por lotes)

## 5 OTRAS OPCIONES DE DSNH

Le permite especificar una lista de opciones DSNH que afectan al proceso de preparación del programa y que anulan las opciones especificadas en otros paneles. Si está utilizando CICS, estas pueden incluir opciones que deseé especificar al CICS traductor de comandos.

Si especifica opciones en este campo, sepárelas con comas. Puede continuar enumerando opciones en la siguiente línea, pero la longitud total de la lista de opciones no puede superar los 70 bytes.

Los campos 6 a 15 le permiten seleccionar la función a realizar y elegir si desea mostrar los paneles de DB2I a para las funciones que seleccione. Utilice S para SÍ o N para NO.

Si está dispuesto a aceptar valores predeterminados para todos los pasos, introduzca N en ¿Panel de visualización? para todos los demás paneles de preparación enumerados.

Para realizar cambios en los valores predeterminados, introduzca Y en ¿Panel de visualización? para cualquier panel que deseé ver. DB2I y, a continuación, mostrará cada uno de los paneles que solicite. Despues de que se muestren todos los paneles, Db2 procede con los pasos necesarios para preparar la ejecución de su programa.

Las variables para todas las funciones utilizadas durante la preparación del programa se mantienen separadas de las variables introducidas desde el menú de opciones principales (DB2I). Por ejemplo, las variables del plan de enlace que se introducen en el panel de preparación del programa se guardan por separado de las de cualquier panel de plan de enlace al que se acceda desde el menú de opciones principal.

## 6 CAMBIAR VALORES PREDETERMINADOS

Le permite especificar si desea cambiar los valores predeterminados de DB2I. ¿Introducir Y en el panel de visualización? junto a esta opción; de lo contrario, introduzca N. Como mínimo, debe especificar su identificador de subsistema y lenguaje de programación en el panel Valores predeterminados.

## 7 PL/I MACRO FASE

Le permite especificar si desea mostrar "el panel Preparación del programa: Compilar, vincular y ejecutar" para controlar la fase de macro PL/I introduciendo opciones PL/I en el campo OPCIONES de ese panel. Ese panel también muestra las opciones COMPILAR O ENSAMBLAR, ENLAZAR y EJECUTAR.

Este campo se aplica únicamente a los programas PL/I. Si su programa no es un programa PL/I o no utiliza el macroprocesador PL/I, especifique N en el campo Realizar función para esta opción, que establece el panel de visualización al valor predeterminado N.

## 8 PRECOMPIILACIÓN

Le permite especificar si desea mostrar el panel Precompilar. Para ver este panel, introduzca Y en el panel de visualización junto a esta opción; de lo contrario, introduzca N.

## 9 CICS TRADUCCIÓN DEL MANDO

Le permite especificar si desea utilizar el CICS traductor de comandos. Este campo se aplica únicamente a CICS programas únicamente.

**IMS y TSO:** Si ejecuta bajo TSO o IMS, ignore este paso; esto permite que el campo de la función Realizar tenga el valor predeterminado N.

**CICS:** Si está utilizando CICS y ha precompilado su programa, debe traducir su programa utilizando el CICS traductor de comandos.

El traductor de comandos no tiene un panel de configuración (DB2I) independiente. Puede especificar las opciones de traducción en el campo Otras opciones del panel Preparación del programa Db2 , o en su programa fuente si no es un programa ensamblador.

Debido a que especificó un CICS entorno de tiempo de ejecución, la columna de la función Ejecutar está predeterminada en S. La traducción de comandos se realiza automáticamente después de precompilar el programa.

## 10 ENVÍO DEL PAQUETE

Le permite especificar si desea mostrar el panel Enlazar paquete. Para verlo, introduce Y en el panel de visualización junto a esta opción; de lo contrario, introduzca N.

## 11 VINCULAR PLAN

Le permite especificar si desea mostrar el panel Enlazar plan. Para verlo, introduce Y en el panel de visualización junto a esta opción; de lo contrario, introduzca N.

## 12 COMPILAR O RECOPILAR

Le permite especificar si desea mostrar el panel "Preparación del programa: Compilar, vincular y ejecutar". Para ver este panel, introduzca Y en el panel de visualización junto a esta opción; de lo contrario, introduzca N.

## 13 PRELINK

Le permite utilizar la utilidad prelink para hacer que su programa C, C++ o Enterprise COBOL para z/OS programa reentrant. Esta utilidad concatena la información de inicialización en tiempo de compilación de una o más pilas de texto en una sola unidad de inicialización. Para utilizar la utilidad, introduzca Y en el panel de visualización junto a esta opción; de lo contrario, introduzca N. Si solicita este paso, también debe solicitar el paso de compilación y el paso de edición de enlaces.

## 14 ENLACE

Le permite especificar si desea mostrar el panel "Preparación del programa: Compilar, vincular y ejecutar". Para verlo, introduce Y en el panel de visualización junto a esta opción; de lo contrario, introduzca N. Si especifica S en el panel de visualización? para la opción COMPILEAR O ENSAMBLAR, no es necesario que realice ningún cambio en este campo; el panel que se muestra para COMPILEAR O ENSAMBLAR es el mismo que el panel que se muestra para ENLAZAR. Puede realizar los cambios que desee en el paso de edición de enlaces al mismo tiempo que realiza los cambios en el paso de compilación.

## 15 EJECUTAR

Le permite especificar si desea ejecutar su programa. La opción RUN solo está disponible si especifica TSO o CAF para RUN TIME ENVIRONMENT.

Si especifica S en el panel de visualización? para la opción COMPILEAR O ENSAMBLAR o ENLAZAR, puede especificar N en este campo, porque el panel que se muestra para COMPILEAR O ENSAMBLAR y para ENLAZAR es el mismo que el panel que se muestra para EJECUTAR.

**IMS y CICS:** IMS y CICS no pueden ejecutarse utilizando DB2I. Si utiliza IMS o CICS, utilice N en estos campos.

**TSO y lote:** Si está utilizando TSO y desea ejecutar su programa, debe introducir Y en la columna Realizar función junto a esta opción. También puede indicar que desea especificar opciones y valores que afecten al funcionamiento de su programa, introduciendo Y en la columna del panel de visualización.

Al pulsar INTRO, se accede al primer panel de la serie especificada; en este ejemplo, al panel de valores predeterminados de DB2I. Si, en cualquier momento de su progreso de un panel a otro, pulsa la tecla FIN, volverá a este primer panel, desde el que podrá cambiar sus especificaciones de procesamiento. ¿Hay asteriscos (\*) en el panel de visualización? la columna de las filas 7 a 14 indica qué paneles ya ha examinado. Puede volver a ver un panel escribiendo una Y sobre un asterisco.

### Referencia relacionada

#### Panel Enlazar paquete

El panel Vincular paquete es el primero de dos paneles DB2I que solicitan información sobre cómo desea vincular un paquete.

#### Panel Enlazar plan

El panel Vincular plan es el primero de dos paneles DB2I que solicitan información sobre cómo desea vincular un plan de aplicación.

#### Panel de valores predeterminados 1 de DB2I

El Panel 1 de valores predeterminados de DB2I le permite cambiar muchos de los valores predeterminados del sistema que se establecieron durante la instalación de Db2.

Paneles Valores predeterminados para BIND PACKAGE y Valores predeterminados REBIND PACKAGE  
Estos paneles DB2I le permiten cambiar los valores predeterminados para las opciones BIND PACKAGE y REBIND PACKAGE.

Paneles Valores predeterminados para BIND PLAN y Valores predeterminados REBIND PLAN  
Estos paneles DB2I le permiten cambiar los valores predeterminados para las opciones BIND PLAN y REBIND PLAN.

#### Panel Precompilar

Tras definir los valores predeterminados de DB2I, puede precompilar su aplicación. Puede alcanzar el panel Precompilar especificándolo como parte del proceso de preparación del programa desde el panel Db2 Program Preparation. También puede acceder a él directamente desde el Menú Opciones primarias de DB2I.

#### Preparación del programa: Panel Compilar, Enlazar y Ejecutar

El panel Compilar, Enlazar y Ejecutar le permite realizar los dos últimos pasos en el proceso de preparación del programa (compilar y editar enlace). Este panel también le permite realizar la PL/I MACRO PHASE para programas que requieren esta opción.

#### Procedimiento de comando DSNH (TSO CLIST) (Comandos de Db2)

Preenlace de una aplicación (z/OS Language Environment Programming Guide)

## Panel de valores predeterminados 1 de DB2I

El Panel 1 de valores predeterminados de DB2I le permite cambiar muchos de los valores predeterminados del sistema que se establecieron durante la instalación de Db2.

La siguiente figura muestra los campos que afectan al procesamiento de los demás paneles de DB2I.

```
DSNEOP01 DB2I DEFAULTS PANEL 1
COMMAND ==>_

Change defaults as desired:

 1 DB2 NAME ==> DSN (Subsystem identifier)
 2 DB2 CONNECTION RETRIES ==> 0 (How many retries for DB2 connection)
 3 APPLICATION LANGUAGE ==> IBMCOB (ASM, C, CPP, IBMCOB, FORTRAN, PLI)
 4 LINES/PAGE OF LISTING ==> 60 (A number from 5 to 999)
 5 MESSAGE LEVEL ==> I (Information, Warning, Error, Severe)
 6 SQL STRING DELIMITER ==> DEFAULT (DEFAULT, ' or ")
 7 DECIMAL POINT ==> . (., or ,)
 8 STOP IF RETURN CODE >= ==> 8 (Lowest terminating return code)
 9 NUMBER OF ROWS ==> 20 (For ISPF Tables)
10 AS USER ==> (User ID to associate with trusted connection)
```

Figura 47. Panel de valores predeterminados 1 de DB2I

A continuación se explican los campos del Panel de valores predeterminados de DB2I 1.

### 1 NOMBRE DE Db2

Le permite especificar el subsistema de Db2 que procesa sus solicitudes de DB2I. Si especifica un subsistema de Db2 diferente, su identificador se muestra en el campo SSID (identificador de subsistema) situado en la parte superior derecha de la pantalla. El valor predeterminado es DSN.

### 2 REINTENTOS DE CONEXIÓN DE Db2

Le permite especificar el número de veces adicionales que se intentará conectar a Db2, si Db2 no está activo cuando el programa emite el comando DSN. El proceso de preparación del programa no utiliza esta opción.

Utilice un número del 0 al 120. El valor predeterminado es 0. Se intentan las conexiones a intervalos de 30 segundos.

### **3 IDIOMA DE LA APLICACIÓN**

Le permite especificar el lenguaje de programación predeterminado para su programa de aplicación. Puede especificar cualquiera de los siguientes idiomas:

#### **ASM**

Para High Level Assembler / z/OS

#### **C**

Para lenguaje C

#### **CPP**

Para C++

#### **IBMCOB**

Para Enterprise COBOL para z/OS. Esta opción es el valor predeterminado.

#### **FORTRAN**

Para VS Fortran

#### **PLI**

para PL/I

Si especifica IBMCOB, Db2 le pedirá más valores predeterminados de COBOL en el panel DSNEOP02. Consulte “[Panel 2 de valores predeterminados de DB2I](#)” en la página 973.

No puede especificar FORTRAN para IMS o CICS programas.

### **4 LÍNEAS/PÁGINA DE ANUNCIO**

Le permite especificar el número de líneas que se imprimirán en cada página del listado o de la salida SPUFI. El valor predeterminado es 60.

### **5 NIVEL DE MENSAJES**

Le permite especificar el nivel más bajo de mensaje que se le devolverá durante la fase BIND del proceso de preparación. Utilice:

#### **I**

Para todos los mensajes de información, advertencia, error y error grave

#### **W**

Para mensajes de advertencia, error y error grave

#### **E**

Para mensajes de error y error grave

#### **S**

Solo para mensajes de error graves

### **6 DELIMITADOR DE CADENA SQL**

Le permite especificar el símbolo utilizado para delimitar una cadena en instrucciones SQL en programas COBOL. Esta opción solo es válida cuando el idioma de la aplicación es IBMCOB Utilice:

#### **DEFAULT**

Para utilizar el valor predeterminado definido en el momento de la instalación

#### **'**

Para un apóstrofe

#### **"**

Para una marca de cotización

### **7 PUNTO DECIMAL**

Le permite especificar cómo su programa fuente de idioma anfitrión representa los separadores decimales y cómo SPUFI muestra los separadores decimales en su salida. Utilice una coma (,) o un punto (.). El valor predeterminado es un punto (.).

### **8 DETENERSE SI EL CÓDIGO DE RETORNO >=**

Le permite especificar el valor más pequeño del código de retorno (de precompilación, compilación, edición de enlaces o enlace) que impedirá la ejecución de pasos posteriores. Utilice:

#### **4**

Para detenerse en advertencias y errores más graves.

**8**

Para detener los errores y los errores más graves. El valor por omisión es 8.

## 9 NÚMERO DE FILAS

Le permite especificar el número predeterminado de filas de entrada que se generarán en la visualización inicial de los paneles de ISPF. El número de filas con entradas que no están en blanco determina el número de filas que aparecen en las visualizaciones posteriores.

## 10 COMO USUARIO

Le permite especificar un ID de usuario para asociarlo con la conexión de confianza para la sesión actual de DB2I.

Db2 establece la conexión de confianza para el usuario que usted especifique si se cumplen las siguientes condiciones:

- El ID de autorización principal que obtiene Db2 después de ejecutar la salida de conexión se le permite utilizar la conexión de confianza sin autenticación.
- La etiqueta de seguridad, si se define implícita o explícitamente en el contexto de confianza para el usuario, se define en RACF para el usuario.

Después de que Db2 establezca la conexión de confianza, el ID de autorización principal, cualquier ID de autorización secundaria, cualquier función y cualquier etiqueta de seguridad que esté asociada con el ID de usuario especificado en el campo COMO USUARIO se utilizarán para la conexión de confianza. Db2 utiliza esta etiqueta de seguridad para verificar la seguridad multinivel para el usuario.

Si el ID de autorización principal asociado al ID de usuario especificado en el campo COMO USUARIO no está autorizado a utilizar la conexión de confianza o requiere información de autenticación, la solicitud de conexión falla. Si Db2 no puede verificar la etiqueta de seguridad, la solicitud de conexión también falla.

El valor que introduzca en este campo se conservará únicamente durante la sesión de DB2I. El campo se restablece en blanco cuando sale de DB2I.

Supongamos que el lenguaje de programación predeterminado es PL/I y que el número predeterminado de líneas por página del listado del programa es 60. Tu programa está en COBOL, por lo que quieres cambiar el campo 3, LENGUAJE DE APLICACIÓN. También desea imprimir 80 líneas por página, por lo que debe cambiar también el campo 4, LÍNEAS/PÁGINA DE LISTADO. Figura 47 en la página 971 muestra las entradas que realiza en el Panel de valores predeterminados 1 de DB2I para realizar estos cambios. En este caso, al pulsar INTRO se accede a Db2 Panel de valores predeterminados 2.

## Panel 2 de valores predeterminados de DB2I

Después de pulsar Intro en el Panel 1 Predeterminados de DB2I, se visualiza el Panel 2 Predeterminados de DB2I. Si elige IBMCOB como lenguaje en el Panel 1 Predeterminados de DB2I, se visualizan tres campos. De lo contrario, solo se visualiza el primer campo.

La siguiente figura muestra el panel de valores predeterminados de DB2I 2 cuando se selecciona IBMCOB.

```

DSNEOP02 DB2I DEFAULTS PANEL 2
COMMAND ==>_

Change defaults as desired:

1 DB2I JOB STATEMENT: (Optional if your site has a SUBMIT exit)
==> //USRTO01A JOB (ACCOUNT), 'NAME'
==> //*
==> //*
==> //*

COBOL DEFAULTS: (For IBMCOB)
2 COBOL STRING DELIMITER ==> DEFAULT (DEFAULT, ' or ")
3 DBCS SYMBOL FOR DCLGEN ==> G (G/N - Character in PIC clause)

```

Figura 48. Panel 2 de valores predeterminados de DB2I

## **1 DECLARACIÓN DE TAREAS DE DB2I**

Le permite cambiar su declaración de trabajo predeterminada. Especifique una sentencia de control de trabajos y, opcionalmente, una sentencia JOBLIB para utilizarla en segundo plano o en el entorno de preparación del programa EDITJCL. Utilice una sentencia JOBLIB para especificar las bibliotecas de tiempo de ejecución que requiere su aplicación. Si su programa tiene una rutina de salida SUBMIT, Db2 utiliza esa rutina. Si esa rutina crea una declaración de control de trabajo, puede dejar este campo en blanco.

## **2 DELIMITADOR DE CADENA COBOL**

Le permite especificar el símbolo utilizado para delimitar una cadena en una instrucción COBOL en una aplicación COBOL. Utilice:

### **DEFAULT**

Para utilizar el valor predeterminado definido en el momento de la instalación

'

Para un apóstrofe

"

Para una marca de cotización

Deje este campo en blanco para aceptar el valor predeterminado.

## **3 SÍMBOLO DBCS PARA DCLGEN**

Le permite introducir G (por defecto) o N, para especificar si DCLGEN genera una cláusula de imagen que tiene la forma PIC G(n) DISPLAY-1 o PIC N(n).

Deje este campo en blanco para aceptar el valor predeterminado.

Al pulsar INTRO, se accede al siguiente panel especificado en el panel Preparación del programa de Db2 , en este caso, al panel Precompilación.

## **Panel Precompilar**

Tras definir los valores predeterminados de DB2I, puede precompilar su aplicación. Puede alcanzar el panel Precompilar especificándolo como parte del proceso de preparación del programa desde el panel Db2 Program Preparation. También puede acceder a él directamente desde el Menú Opciones primarias de DB2I.

La forma en que elija llegar al panel determina los valores predeterminados de los campos que contiene. La siguiente figura muestra el panel de Precompilación.

```
DSNETP01 PRECOMPILE SSID: DSN
COMMAND ==>_
Enter precompiler data sets:
1 INPUT DATA SET ==> SAMPLEPG.COBOL
2 INCLUDE LIBRARY ... ==> SRCLIB.DATA

3 DSNAME QUALIFIER .. ==> TEMP (For building data set names)
4 DBRM DATA SET ==>

Enter processing options as desired:
5 WHERE TO PRECOMPILE ==> FOREGROUND (BACKGROUND, or EDITJCL)
6 VERSION ==> (Blank, VERSION, or AUTO)
7 OTHER OPTIONS ==>
```

*Figura 49. El panel de Precompilación*

A continuación se explican las funciones del panel de precompilación y cómo introducir los campos para prepararse para la precompilación.

## **1 CONFIGURACIÓN DE DATOS DE ENTRADA**

Le permite especificar el nombre del conjunto de datos del programa de origen y las sentencias SQL que se van a precompilar.

Si ha llegado a este panel a través del panel de Preparación del programa de Db2 , este campo contiene el nombre del conjunto de datos especificado allí. Puede anularla en este panel.

Si ha llegado a este panel directamente desde el menú de opciones principales de DB2I, debe introducir el nombre del conjunto de datos del programa que desea precompilar. El nombre del conjunto de datos puede incluir un nombre de miembro. Si no incluye el nombre del conjunto de datos con apóstrofes, un prefijo TSO estándar (ID de usuario) califica el nombre del conjunto de datos.

## 2 INCLUIR BIBLIOTECA

Le permite introducir el nombre de una biblioteca que contenga miembros que el precompilador debe incluir. Estos miembros pueden contener resultados de DCLGEN. Si no incluye el nombre entre comillas, un prefijo TSO estándar (ID de usuario) califica el nombre.

Puede solicitar bibliotecas INCLUDE adicionales introduciendo parámetros DSNH CLIST del formulario PnLIB (*dsname*), donde *n* es 2, 3 o 4) en el campo OTRAS OPCIONES de este panel o en el campo OTRAS OPCIONES DSNH del panel Preparación del programa.

## 3 CALIFICADOR DE NOMBRE DE DOMINIO

Le permite especificar una cadena de caracteres que califique los nombres de conjuntos de datos temporales durante la precompilación. Utilice cualquier cadena de caracteres de 1 a 8 caracteres de longitud que se ajuste a las convenciones de nomenclatura normales de TSO.

Si ha llegado a este panel a través del panel de Preparación del programa de Db2 , este campo contiene el calificador del nombre del conjunto de datos especificado allí. Puede anularla en este panel.

Si ha llegado a este panel desde el menú de opciones principal de DB2I, puede especificar un CALIFICADOR DE NOMBRE DE SERVICIO o dejar que el campo tome su valor predeterminado, TEMP.

**IMS y TSO:** Para los IMS y TSO, Db2 almacena las instrucciones de origen precompiladas (para pasarlas al compilador o al paso de ensamblaje) en un conjunto de datos denominado *tsoprefix.qualifier.suffix*. Un conjunto de datos llamado *tsoprefix.qualifier.PCLIST* contiene el listado de impresión del precompilador.

Para los programas preparados en segundo plano o que utilizan la opción EDITJCL del ENTORNO DE PREPARACIÓN (en el panel de Preparación de programas de Db2 ), un conjunto de datos llamado *tsoprefix.qualifier.CNTL* contiene el JCL de preparación del programa.

En estos ejemplos, *tsoprefix* representa el prefijo que asigna TSO, a menudo el mismo que el ID de autorización. *el calificador* representa el valor introducido en el campo DSNAME QUALIFIER. *el sufijo* representa el nombre de salida, que es uno de los siguientes: COBOL, FORTRAN, C, PLI, ASM, DECK, CICSIN, OBJ o DATA. En el panel de precompilación que se muestra arriba, el conjunto de datos *tsoprefix.TEMP.COBO*L contiene las instrucciones de origen precompiladas, y *tsoprefix.TEMP.PCLIST* contiene el listado de impresión del precompilador. Si ya existen conjuntos de datos con estos nombres, Db2 los elimina.

**CICS :** Para CICS programas, el conjunto de datos *tsoprefix.qualifier.suffix* recibe las instrucciones de origen precompiladas en preparación para CICS traducción de comandos.

Si no planea hacer CICS traducir el comando, las sentencias de origen en *tsoprefix.qualifier.suffix*, están listas para compilarse. El conjunto de datos *tsoprefix.qualifier.PCLIST* contiene el listado de impresión del precompilador.

Cuando el precompilador completa su trabajo, el control pasa al CICS traductor de comandos. Como no hay panel para el traductor, la traducción se realiza automáticamente. El conjunto de datos *tsoprefix.qualifier.CXLIST* contiene el resultado del traductor de comandos.

## 4 CONJUNTO DE DATOS DBRM

Le permite nombrar el conjunto de datos de la biblioteca DBRM para la salida del precompilador. El conjunto de datos también puede incluir un nombre de miembro.

Cuando llegue a este panel, el campo estará en blanco. Sin embargo, al pulsar INTRO, el valor contenido en el campo DSNAME QUALIFIER del panel, concatenado con DBRM, especifica el conjunto de datos DBRM: *qualifier.DBRM*.

Puede introducir otro nombre de conjunto de datos en este campo solo si asigna y cataloga el conjunto de datos antes de hacerlo. Esto es así incluso si el nombre del conjunto de datos que introduce se corresponde con el valor predeterminado de este campo.

El precompilador envía el código fuente modificado al conjunto de datos *qualifier.host*, donde *host* es el idioma especificado en el campo IDIOMA DE APLICACIÓN del panel DB2I Defaults 1.

## 5 DÓNDE PRECOMPILAR

Le permite indicar si desea precompilar en primer plano o en segundo plano. También puede especificar EDITJCL, en cuyo caso podrá editar y luego enviar el trabajo.

Si ha llegado a este panel desde el panel Preparación del programa de Db2 , el campo contiene el entorno de preparación especificado allí. Puede anular ese valor si lo desea.

Si ha llegado a este panel directamente desde el menú de opciones principal de DB2I, puede especificar un entorno de procesamiento o dejar que este campo tome su valor predeterminado. Utilice:

FOREGROUND para precompilar inmediatamente el programa con los valores que especifique en estos paneles.

ANTECEDENTES para crear y enviar inmediatamente para ejecutar un archivo que contenga un DSNH CLIST utilizando la instrucción de control JOB desde el Panel 2 de Defaults de DB2I o la salida SUBMIT de su sitio. El archivo se guarda.

EDITJCL para crear y abrir un archivo que contenga un DSNH CLIST en modo de edición. A continuación, puede enviar el CLIST o guardarla.

## 6 VERSIÓN

Le permite especificar la versión del programa y su DBRM. Si la versión contiene el número máximo de caracteres permitidos (64), debe introducir cada carácter sin espacios intermedios de una línea a la siguiente. Este campo es opcional.

## 7 OTRAS OPCIONES

Le permite introducir cualquier opción que acepte el DSNH CLIST, lo que le da un mayor control sobre su programa. Las opciones de DSNH que especifique en este campo anulan las opciones especificadas en otros paneles. La lista de opciones puede continuar en la siguiente línea, pero la longitud total de la lista no puede ser superior a 70 bytes.

### Referencia relacionada

[Procedimiento de comando DSNH \(TSO CLIST\) \(Comandos de Db2 \)](#)

## Panel Enlazar paquete

El panel Vincular paquete es el primero de dos paneles DB2I que solicitan información sobre cómo desea vincular un paquete.

Puede acceder al panel del paquete de vinculación directamente desde el menú de opciones principales de DB2I, o como parte del proceso de preparación del programa. Si se entra en el panel de Paquete de enlace desde el panel de Preparación del programa, muchas de las entradas del Paquete de enlace contienen valores de los paneles Primario y Precompilación. [Figura 50 en la página 977 muestra el panel Enlazar paquete.](#)

```

DSNEBP07 BIND PACKAGE SSID: DSN

COMMAND ==>_

Specify output location and collection names:
 1 LOCATION NAME ==> (Defaults to local)
 2 COLLECTION-ID ==> > (Required)

Specify package source (DBRM or COPY):
 3 DBRM: COPY: ==> DBRM (Specify DBRM or COPY)
 4 MEMBER or COLLECTION-ID ==> >
 5 PASSWORD or PACKAGE-ID .. ==> >
 6 LIBRARY or VERSION ==>
 (Blank, or COPY version-id)
 7 -- OPTIONS ==> (COMPOSITE or COMMAND)

Enter options as desired:
 8 CHANGE CURRENT DEFAULTS? ==> NO (NO or YES)
 9 ENABLE/DISABLE CONNECTIONS? ==> NO (NO or YES)
10 OWNER OF PACKAGE (AUTHID).. ==> > (Leave blank for primary ID)
11 QUALIFIER ==> > (Leave blank for OWNER)
12 ACTION ON PACKAGE ==> REPLACE (ADD or REPLACE)
13 INCLUDE PATH? ==> NO (NO or YES)
14 REPLACE VERSION ==> (Replacement version-id)

```

*Figura 50. El panel del paquete Bind*

La siguiente información explica las funciones del panel Enlazar paquete y cómo llenar los campos necesarios para enlazar su programa.

## 1 NOMBRE DE LA UBICACIÓN

Le permite especificar el sistema al que vincular el paquete. Puede utilizar de 1 a 16 caracteres para especificar el nombre de la ubicación. El nombre de la ubicación debe estar definido en la tabla del catálogo SYSIBM.LOCATIONS. El valor predeterminado es el DBMS local.

## 2 IDENTIFICACIÓN DE LA COLECCIÓN

Le permite especificar la colección en la que se encuentra el paquete. Puede utilizar de 1 a 128 caracteres para especificar la colección, y el primer carácter debe ser alfabético. Este campo se puede desplazar.

### 3 DBRM: COPIA:

Le permite especificar si está creando un paquete nuevo (DBRM) o haciendo una copia de un paquete que ya existe (COPY). Utilice:

#### DBRM

Para crear un nuevo paquete. Debe especificar valores en los campos BIBLIOTECA, CONTRASEÑA y MIEMBRO.

#### COPiar

Para copiar un paquete existente. Debe especificar valores en los campos COLLECTION-ID y PACKAGE-ID. (El campo VERSIÓN es opcional)

## 4 IDENTIFICACIÓN DE MIEMBRO o DE COLECCIÓN

**MIEMBRO (para paquetes nuevos):** Si está creando un paquete nuevo, esta opción le permite especificar el DBRM que se va a vincular. Puede especificar un nombre de miembro de 1 a 128 caracteres. Este campo se puede desplazar. El nombre predeterminado depende del nombre del conjunto de datos de entrada.

- Si el conjunto de datos de entrada está particionado, el nombre predeterminado es el nombre de miembro del conjunto de datos de entrada especificado en el campo INPUT DATA SET NAME (NOMBRE DEL CONJUNTO DE DATOS DE ENTRADA) del panel de preparación del programa (Db2).
- Si el conjunto de datos de entrada es secuencial, el nombre predeterminado es el segundo calificador de este conjunto de datos de entrada.

**COLLECTION-ID (para copiar un paquete):** Si está copiando un paquete, esta opción especifica el ID de recogida que contiene el paquete original. Puede especificar un ID de colección de 1 a 128 caracteres, que debe ser diferente del ID de colección especificado en el campo ID DE PAQUETE. Este campo se puede desplazar.

## **5 CONTRASEÑA o ID DE PAQUETE**

**CONTRASEÑA (para paquetes nuevos ):** Si está creando un paquete nuevo, esto le permite introducir la contraseña de la biblioteca que indica en el campo BIBLIOTECA. Puede utilizar este campo solo si ha llegado al panel de vinculación de paquetes directamente desde el menú de opciones principales de Db2 . Este campo se puede desplazar.

**PACKAGE-ID (para copiar paquetes ):** Si está copiando un paquete, esta opción le permite especificar el nombre del paquete original. Puede introducir un ID de paquete de 1 a 128 caracteres. Este campo se puede desplazar.

## **6 BIBLIOTECA o VERSIÓN**

**BIBLIOTECA (para paquetes nuevos ):** Si está creando un paquete nuevo, esto le permite especificar los nombres de las bibliotecas que contienen los DBRM especificados en el campo MEMBER para el proceso de enlace. Las bibliotecas se buscan en el orden especificado y deben figurar en las tablas del catálogo.

**VERSIÓN (para copiar paquetes ):** Si está copiando un paquete, esta opción le permite especificar la versión del paquete original. Puede especificar un ID de versión de entre 1 y 64 caracteres.

## **7 OPCIONES**

Le permite especificar qué opciones de enlace utiliza Db2 cuando emite BIND PACKAGE con la opción COPY. Especifique:

- **COMPOSITE (predeterminado )** para hacer que Db2 utilice las opciones que especifique en el comando BIND PACKAGE. Para todas las demás opciones, Db2 utiliza las opciones del paquete copiado.
- **COMMAND** para hacer que Db2 utilice las opciones que especifique en el comando BIND PACKAGE. Para todas las demás opciones, Db2 utiliza los siguientes valores:
  - Para obtener una copia local de un paquete, Db2 utiliza los valores predeterminados para el subsistema local Db2 .
  - Para una copia remota de un paquete, Db2 utiliza los valores predeterminados para el servidor al que está vinculado el paquete.

## **8 ¿CAMBIAR LOS VALORES PREDETERMINADOS ACTUALES?**

Le permite especificar si desea cambiar los valores predeterminados actuales para los paquetes de encuadernación. Si introduce SÍ en este campo, verá el panel Valores predeterminados del paquete de enlace como siguiente paso. Puede introducir sus nuevas preferencias allí; para obtener instrucciones, consulte “[Paneles Valores predeterminados para BIND PACKAGE y Valores predeterminados REBIND PACKAGE](#)” en la página 982.

## **9 ¿ACTIVAR/DESACTIVAR CONEXIONES?**

Le permite especificar si desea habilitar y deshabilitar los tipos de conexiones del sistema para usar con este paquete. Esto solo es válido si el campo NOMBRE DE UBICACIÓN nombra su sistema de e Db2 a local.

Si se introduce SÍ en este campo, se mostrará un panel (que se muestra en [Figura 56 en la página 987](#)) que le permite especificar si varias conexiones del sistema son válidas para esta aplicación. Puede especificar nombres de conexión para identificar mejor las conexiones habilitadas dentro de un tipo de conexión. Un nombre de conexión solo es válido cuando también se especifica su tipo de conexión correspondiente.

La opción predeterminada habilita todos los tipos de conexión.

## **10. PROPIETARIO DEL PAQUETE (AUTHID)**

Le permite especificar el ID de autorización principal del propietario del nuevo paquete. Dicho ID es el nombre del propietario del paquete y el nombre asociado a todos los registros contables y de seguimiento generados por el paquete.

El propietario debe tener los privilegios necesarios para ejecutar las sentencias SQL contenidas en el paquete.

El valor predeterminado es el ID de autorización principal del proceso de enlace.

El campo se puede desplazar y la longitud máxima del campo es de 128.

## 11 REQUISITO

Le permite especificar el esquema predeterminado para tablas, vistas, índices y alias no cualificados. Puede especificar un nombre de esquema de 1 a 128 caracteres. El valor predeterminado es el ID de autorización del propietario del paquete. Este campo se puede desplazar.

## 12 ACCIÓN SOBRE EL PAQUETE

Le permite especificar si desea reemplazar un paquete existente o crear uno nuevo. Utilice:

**REPLACE (predeterminado)** para reemplazar el paquete nombrado en el campo PACKAGE-ID si ya existe, y agregarlo si no existe. (Utilice esta opción si va a cambiar el paquete porque han cambiado las instrucciones SQL del programa). Si solo cambia el entorno SQL pero no las sentencias SQL, puede utilizar REBIND PACKAGE

**ADD** para añadir el paquete nombrado en el campo PACKAGE-ID, solo si no existe ya.

## 13 ¿INCLUIR CAMINO?

Indica si proporcionará una lista de nombres de esquemas que Db2 busca cuando resuelve nombres de tipos distintos no cualificados, funciones definidas por el usuario y procedimientos almacenados en sentencias SQL. El valor predeterminado es NO. Si especifica SÍ, Db2 muestra un panel en el que se especifican los nombres de los esquemas que Db2 debe buscar.

## 14 SUSTITUIR VERSIÓN

Le permite especificar si desea reemplazar una versión específica de un paquete existente o crear uno nuevo. Si el paquete y la versión nombrados en los campos PACKAGE-ID y VERSION ya existen, debe especificar REPLACE. Puede especificar un ID de versión de entre 1 y 64 caracteres. El ID de versión predeterminado es el especificado en el campo VERSIÓN.

## Panel Enlazar plan

El panel Vincular plan es el primero de dos paneles DB2I que solicitan información sobre cómo desea vincular un plan de aplicación.

Al igual que el panel Precompilar, puede acceder al panel Enlazar plan directamente desde el menú de opciones principales de DB2I, o como parte del proceso de preparación del programa. Debe tener un plan de aplicación, incluso si vincula su aplicación a paquetes; este panel también sigue los paneles de Paquete vinculado.

Si entra en el panel Plan de enlace desde el panel Preparación del programa, muchas de las entradas del Plan de enlace contienen valores de los paneles Principal y Precompilación.

```
DSNEBP02 BIND PLAN SSID: DSN
COMMAND ==>_

Enter primary package list:
 1 LOCATION NAME ==> (Defaults to local)
 2 COLLECTION ID ==> > (Required)
 3 PACKAGE ID ==> (Package ID or *)
 4 ADDITIONAL PACKAGE LISTS .. ==> NO (YES to include more packages)

Enter options as desired:
 5 PLAN NAME ==> (Required to create a plan)
 6 CHANGE CURRENT DEFAULTS?.. ==> NO (NO or YES)
 7 ENABLE/DISABLE CONNECTIONS? ==> NO (NO or YES)
 8 OWNER OF PLAN (AUTHID)..... ==> > (Leave blank for your primary ID)
 9 QUALIFIER ==> > (For tables, views, and aliases)
10 CACHESIZE ==> 0 (Blank, or value 0-4096)
11 ACTION ON PLAN ==> REPLACE (REPLACE or ADD)
12 RETAIN EXECUTION AUTHORITY. ==> NO (YES to retain user list)
13 CURRENT SERVER ==> (Location name)
14 INCLUDE PATH? ==> NO (NO or YES)
```

Figura 51. El panel del Plan de vinculación

A continuación se explican las funciones del panel Enlazar plan y cómo llenar los campos necesarios para enlazar su programa.

## **1 NOMBRE DE LA UBICACIÓN**

Le permite especificar el sistema remoto al que está vinculado el paquete que se nombra en el campo ID DE PAQUETE. El nombre de la ubicación debe estar definido en la tabla del catálogo SYSIBM.LOCATIONS. El valor predeterminado es el DBMS local.

## **2 IDENTIFICACIÓN DE LA COLECCIÓN**

Le permite especificar la colección que incluye el paquete que se va a vincular al plan.

El campo se puede desplazar y la longitud máxima del campo es de 128.

## **3 ID DEL PAQUETE**

Le permite especificar el nombre del paquete que se va a vincular al plan.

## **4 LISTAS ADICIONALES DE PAQUETES**

Le permite incluir una lista de paquetes adicionales en el plan. Si especifica SÍ, se muestra un panel independiente, donde debe introducir la ubicación del paquete, el nombre de la recogida y el nombre del paquete para cada paquete que se incluya en el plan. Esta lista es opcional.

## **5 NOMBRE DEL PLAN**

Le permite nombrar el plan de aplicación que desea crear. Puede especificar un nombre de 1 a 8 caracteres, y el primer carácter debe ser alfabético. Si no hay errores, el proceso de enlace prepara el plan e introduce su descripción en la tabla EXPLAIN.

Si ha llegado a este panel a través del panel de Preparación del programa de Db2 , el valor predeterminado de este campo depende del valor que haya introducido en el campo INPUT DATA SET NAME de ese panel.

Si ha llegado a este panel directamente desde el menú de opciones principal de Db2 , debe incluir un nombre de plan si desea crear un plan de aplicación. El nombre predeterminado de este campo depende del conjunto de datos de entrada:

- Si el conjunto de datos de entrada está particionado, el nombre predeterminado es el nombre del miembro.
- Si el conjunto de datos de entrada es secuencial, el nombre predeterminado es el segundo calificador del nombre del conjunto de datos.

## **6 ¿CAMBIAR LOS VALORES PREDETERMINADOS ACTUALES?**

Le permite especificar si desea cambiar los valores predeterminados actuales para los planes vinculantes. Si introduce SÍ en este campo, verá el panel Valores predeterminados para el plan vinculado como siguiente paso. Allí puede introducir sus nuevas preferencias.

## **7 ¿ACTIVAR/DESACTIVAR CONEXIONES?**

Le permite especificar si desea habilitar y deshabilitar los tipos de conexiones del sistema para usar con este paquete. Esto solo es válido si el campo NOMBRE DE UBICACIÓN nombra su sistema de e Db2 a local.

Si se introduce SÍ en este campo, se mostrará un panel (que se muestra en [Figura 56 en la página 987](#)) que le permite especificar si varias conexiones del sistema son válidas para esta aplicación. Puede especificar nombres de conexión para identificar mejor las conexiones habilitadas dentro de un tipo de conexión. Un nombre de conexión solo es válido cuando también se especifica su tipo de conexión correspondiente.

La opción predeterminada habilita todos los tipos de conexión.

## **8 PROPIETARIO DEL PLAN (AUTHID)**

Le permite especificar el ID de autorización principal del propietario del nuevo plan. Dicho ID es el nombre del titular del plan y el nombre asociado a todos los registros contables y de seguimiento generados por el plan.

El propietario debe tener los privilegios necesarios para ejecutar las instrucciones SQL contenidas en el plan.

El campo se puede desplazar y la longitud máxima del campo es de 128.

## **9 REQUISITO**

Le permite especificar el esquema predeterminado para tablas, vistas y alias no cualificados. Puede especificar un nombre de esquema de 1 a 128 caracteres, que debe ajustarse a las reglas de los identificadores SQL. Si deja este campo en blanco, el calificador predeterminado es el ID de autorización del propietario del plan. Este campo se puede desplazar.

Le permite especificar el esquema predeterminado para tablas, vistas y alias no cualificados. Puede especificar un nombre de esquema de entre 1 y 8 caracteres, que debe ajustarse a las reglas de los identificadores SQL. Si deja este campo en blanco, el calificador predeterminado es el ID de autorización del propietario del plan.

## **10 TAMAÑO DE LA CAJA**

Le permite especificar el tamaño (en bytes) de la caché de autorización. Los valores válidos están en el rango de 0 a 4096. Los valores que no son múltiplos de 256 se redondean al siguiente múltiplo más alto de 256. Un valor de 0 indica que Db2 no utiliza caché de autorización. El valor predeterminado es de 1024.

Cada usuario simultáneo de un plan requiere 8 bytes de almacenamiento, con 32 bytes adicionales para gastos generales.

## **11 ACCIÓN EN EL PLAN**

Le permite especificar si se trata de un plan de aplicación nuevo o modificado. Utilice:

**REEMPLAZAR (opción predeterminada )** para reemplazar el plan nombrado en el campo NOMBRE DEL PLAN si ya existe, y añadir el plan si no existe.

**ANADIR** para añadir el plan nombrado en el campo NOMBRE DEL PLAN, solo si aún no existe.

## **12 CONSERVAR LA AUTORIDAD DE EJECUCIÓN**

Le permite elegir si los usuarios con autoridad para vincular o ejecutar el plan existente deben mantener dicha autoridad sobre el plan modificado. Esto solo se aplica cuando se reemplaza un plan existente.

Si la titularidad del plan cambia y usted especifica SÍ, el nuevo titular concede autoridad de VINCULACIÓN y EJECUCIÓN al anterior titular del plan.

Si la propiedad del plan cambia y no especifica SÍ, todos, excepto el nuevo propietario del plan, perderán la autoridad de EJECUTAR (pero no la autoridad de VINCULAR), y el nuevo propietario del plan concederá la autoridad de VINCULAR al propietario anterior del plan.

## **13 SERVIDOR ACTUAL**

Le permite especificar el servidor inicial para recibir y procesar instrucciones SQL en este plan. Puede especificar un nombre de 1 a 16 caracteres, que debe definir previamente en la tabla del catálogo SYSIBM.LOCATIONS.

Si especifica un servidor remoto, Db2 se conecta a ese servidor cuando se ejecuta la primera instrucción SQL. El valor predeterminado es el nombre del subsistema de Db2 local.

## **14 ¿INCLUIR CAMINO?**

Indica si proporcionará una lista de nombres de esquemas que Db2 busca cuando resuelve nombres de tipos distintos no cualificados, funciones definidas por el usuario y procedimientos almacenados en sentencias SQL. El valor predeterminado es NO. Si especifica SÍ, Db2 muestra un panel en el que se especifican los nombres de los esquemas que Db2 debe buscar.

Cuando termine de realizar cambios en este panel, pulse INTRO para ir al segundo de los paneles de preparación del programa, Preparación del programa: Compilar, vincular y ejecutar.

### **Tareas relacionadas**

[IDs de autorización de almacenamiento en memoria caché para planes \(Managing Security\)](#)

### **Referencia relacionada**

[Paneles Valores predeterminados para BIND PLAN y Valores predeterminados REBIND PLAN](#)

Estos paneles DB2I le permiten cambiar los valores predeterminados para las opciones BIND PLAN y REBIND PLAN.

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2 \)](#)

## Paneles Valores predeterminados para BIND PACKAGE y Valores predeterminados REBIND PACKAGE

Estos paneles DB2I le permiten cambiar los valores predeterminados para las opciones BIND PACKAGE y REBIND PACKAGE.

En el siguiente panel, introduzca nuevos valores predeterminados para la encuadernación de un paquete.

```
DSNEBP10 DEFAULTS FOR BIND PACKAGE SSID: DSN
COMMAND ===> -
----- Use the UP/DOWN keys to access all options -----
More: +
Change default options as necessary:
1 ISOLATION LEVEL ==> (CS, RR, RS, UR, or NC)
2 VALIDATION TIME ==> (RUN or BIND)
3 RESOURCE RELEASE TIME ... ==> (COMMIT, DEALLOCATE, or
 INHERITFROMPLAN)
4 EXPLAIN PATH SELECTION .. ==> (NO or YES)
5 DATA CURRENCY ==> (NO or YES)
6 PARALLEL DEGREE ==> (1 or ANY)
7 SQLERROR PROCESSING ==> (NOPACKAGE or CONTINUE)
8 REOPTIMIZE FOR INPUT VARS ==> (ALWAYS, NONE, ONCE, or AUTO)
9 DEFER PREPARE ==> (NO, YES, or INHERITFROMPLAN)
10 KEEP DYNAMIC SQL
 PAST COMMIT or ROLLBACK ==> (NO or YES)
11 APPLICATION ENCODING ... ==> (Blank, ASCII, EBCDIC,
 UNICODE, or ccsid)
12 OPTIMIZATION HINT ==> > (Blank or 'hint-id')
13 IMMEDIATE WRITE ==> (YES, NO, or INHERITFROMPLAN)
14 DYNAMIC RULES ==> (RUN, BIND, DEFINE, or INVOKE)
15 DBPROTOCOL ==> (blank, DRDA, or DRDABCF)
16 ACCESS PATH REUSE ==> (ERROR, NONE, or WARN)
17 ACCESS PATH COMPARISON .. ==> (ERROR, NONE, or WARN)
18 SYSTEM_TIME SENSITIVE ... ==> (blank, NO, or YES)
19 BUSINESS_TIME SENSITIVE . ==> (blank, NO, or YES)
20 ARCHIVE SENSITIVE ==> (blank, NO, or YES)
21 APPLICATION COMPATIBILITY ==> (blank, DB2 function level,
 V10R1, or V11R1)

PRESS: ENTER to continue UP/DOWN to scroll RETURN to EXIT
```

Figura 52. El panel Valores predeterminados para el paquete de encuadernación

En el siguiente panel, introduzca nuevos valores predeterminados para reempaquetar un paquete.

Salvo algunas excepciones menores, las opciones de este panel son las mismas que las opciones predeterminadas para reempaquetar un paquete. Sin embargo, los valores predeterminados para REBIND PACKAGE son diferentes de los que se muestran en la figura anterior, y puede especificar SAME en cualquier campo para especificar los valores utilizados la última vez que se vinculó el paquete. Para la reasignación, el valor predeterminado para todos los campos es IGUAL.

```

DSNEBP11 DEFAULTS FOR REBIND PACKAGE SSID: DSN
COMMAND ==> -
----- Use the UP/DOWN keys to access all options -----
More: +
Change default options as necessary:

 1 ISOLATION LEVEL ==> (SAME, CS, RR, RS, UR, or NC)
 2 PLAN VALIDATION TIME ... ==> (SAME, RUN, or BIND)
 3 RESOURCE RELEASE TIME ... ==> (SAME, DEALLOCATE, COMMIT,
 or INHERITFROMPLAN)
 4 EXPLAIN PATH SELECTION .. ==> (SAME, NO, or YES)
 5 DATA CURRENCY ==> (SAME, NO, or YES)
 6 PARALLEL DEGREE ==> (SAME, 1 or ANY)
 7 REOPTIMIZE FOR INPUT VARS ===> (SAME, ALWAYS, NONE, ONCE, AUTO)
 8 DEFER PREPARE ==> (SAME, NO, YES,
 or INHERITFROMPLAN)

 9 KEEP DYNAMIC SQL
 PAST COMMIT OR ROLLBACK. ==> (SAME, NO, or YES)
10 APPLICATION ENCODING ... ==> (SAME, Blank, ASCII, EBCDIC,
 UNICODE, or ccsid)
11 OPTIMIZATION HINT ==> > (Blank or 'hint-id')
12 IMMEDIATE WRITE ==> (SAME, NO, YES,
 or INHERITFROMPLAN)
13 DBPROTOCOL ==> (blank, DRDA, or DRDABCF)
14 DYNAMIC RULES ==> (SAME, RUN, BIND,
 DEFINERUN, DEFINEBIND,
 INVOKERUN or INVOKEBIND)
15 PLAN MANAGEMENT ==> (DEFAULT, BASIC, EXTENDED, OFF)
16 ACCESS PATH REUSE ==> (DEFAULT, ERROR, NONE, or WARN)
17 ACCESS PATH COMPARISON .. ==> (DEFAULT, ERROR, NONE, or WARN)
18 ACCESS PATH RETAIN DUPS . ==> (DEFAULT, NO, OR YES)
19 SYSTEM_TIME SENSITIVE ... ==> (SAME, NO, or YES)
20 BUSINESS_TIME SENSITIVE . ==> (SAME, NO, or YES)
21 ARCHIVE SENSITIVE ==> (SAME, NO, or YES)
22 APPLICATION COMPATIBILITY ==> (SAME, DB2 función level,
 V10R1, or V11R1)

```

PRESS: ENTER to continue    UP/DOWN to scroll    RETURN to EXIT

*Figura 53. El panel Valores predeterminados para el paquete de reasignación*

La siguiente tabla enumera los campos de los paneles Valores predeterminados para el paquete de encuadernación y Valores predeterminados para el paquete de re-encuadernación, y las opciones de encuadernación y re-encuadernación correspondientes.

*Tabla 150. Valores predeterminados para el paquete de encuadernación y valores predeterminados para el paquete de re-encuadernación campos de panel y opciones de encuadernación o re-encuadernación correspondientes*

Nombre de campo	Opción de encuadernación o re-encuadernación
COMPARACIÓN DE LA RUTA DE ACCESO	<a href="#">APCOMPARE</a>
CAMINO DE ACCESO CONSERVAR COPIAS	<a href="#">APRETAINUP</a>
REUTILIZACIÓN DE LA VÍA DE ACCESO	<a href="#">APREUSE</a>
compatibilidad de aplicaciones	<a href="#">APPLCOMPAT</a>
APPLICATION ENCODING	<a href="#">ENCODING</a>
ARCHIVO SENSIBLE	<a href="#">ARCHIVESENSITIVE</a>
HORARIO DE OFICINA SENSIBLE	<a href="#">SENSIBLE A LOS BRUJEROS</a>
moneda de datos	<a href="#">CURRENTDATA</a>
DBPROTOCOL	<a href="#">DBPROTOCOL</a>
APLAZAR PREPARAR	<a href="#">DEFER y NODEFER</a>
NORMAS DINÁMICAS	<a href="#">DYNAMICRULES</a>
EXPLICAR LA SELECCIÓN DE RUTA	<a href="#">EXPLAIN</a>

*Tabla 150. Valores predeterminados para el paquete de encuadernación y valores predeterminados para el paquete de re-encuadernación campos de panel y opciones de encuadernación o re-encuadernación correspondientes (continuación)*

<b>Nombre de campo</b>	<b>Opción de encuadernación o re-encuadernación</b>
ESCRITURA INMEDIATA	<a href="#">IMMEDWRITE</a>
ISOLATION LEVEL	<a href="#">ISOLATION</a>
MANTENER SQL DINÁMICO PASADO EL COMPROMISO O	<a href="#">KEEPDYNAMIC</a>
Sugerencia de optimización	<a href="#">OPTHINT</a>
GRADO PARALELO	<a href="#">DEGREE</a>
GESTIÓN DEL PLAN	<a href="#">PLANMGMT</a>
REOPTIMIZAR PARA VARIABLES DE ENTRADA	<a href="#">REOPT</a>
HORA DE LIBERACIÓN DE RECURSOS	<a href="#">RELEASE</a>
PROCESAMIENTO DE SQLERROR	<a href="#">SQLERROR</a>
SENSIBLE AL TIEMPO DEL SISTEMA	<a href="#">SYSTIMESENSITIVE</a>
HORA DE VALIDACIÓN y HORA DE VALIDACIÓN DEL PLAN	<a href="#">Validate</a>

### **Conceptos relacionados**

[Opciones de reglas dinámicas para sentencias de SQL dinámico](#)

La opción de vinculación DYNAMICRULES y el entorno de ejecución determinan las reglas para los atributos de SQL dinámico.

[Proceso paralelo \(Db2 Performance\)](#)

[Investigación del rendimiento de SQL utilizando EXPLAIN \(Db2 Performance\)](#)

### **Tareas relacionadas**

[Establecimiento del nivel de aislamiento de sentencias SQL en un programa REXX](#)

Los niveles de aislamiento especifican el comportamiento de bloqueo para sentencias SQL. Puede establecer el nivel de aislamiento de las sentencias SQL en su programa REXX en lectura repetible (RR), estabilidad de lectura (RS), estabilidad del cursor (CS) o lectura no confirmada (UR).

### **Referencia relacionada**

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2 \)](#)

## **Paneles Valores predeterminados para BIND PLAN y Valores predeterminados REBIND PLAN**

Estos paneles DB2I le permiten cambiar los valores predeterminados para las opciones BIND PLAN y REBIND PLAN.

En el siguiente panel, introduzca nuevos valores predeterminados para vincular un plan.

```

DSNEBP10 DEFAULTS FOR BIND PLAN SSID: DSN
COMMAND ===>

Change default options as necessary:

 1 ISOLATION LEVEL ==> RR (RR, RS, CS, or UR)
 2 VALIDATION TIME ==> RUN (RUN or BIND)
 3 RESOURCE RELEASE TIME ... ==> COMMIT (COMMIT, DEALLOCATE, or
 INHERITFROMPLAN)

 4 EXPLAIN PATH SELECTION .. ==> NO (NO or YES)
 5 DATA CURRENCY ==> NO (NO or YES)
 6 PARALLEL DEGREE ==> 1 (1 or ANY)
 7 RESOURCE ACQUISITION TIME ==> USE (USE or ALLOCATE)
 8 REOPTIMIZE FOR INPUT VARS ==> NONE (ALWAYS, NONE, ONCE, AUTO)
 9 DEFER PREPARE ==> NO (NO, YES, INHERITFROMPLAN)

10 KEEP DYNAMIC SQL
 PAST COMMIT OR ROLLBACK.. ==> NO (NO or YES)
11 APPLICATION ENCODING ... ==> (Blank, ASCII, EBCDIC, UNICODE,
 or ccsid)

12 OPTIMIZATION HINT ==> > (Blank or 'hint-id')
13 IMMEDIATE WRITE ==> NO (NO, YES, INHERITFROMPLAN)
14 DYNAMIC RULES ==> RUN (RUN or BIND)
15 SQLRULES..... ==> DB2 (DB2 or STD)
16 DISCONNECT ==> EXPLICIT (EXPLICIT, AUTOMATIC,
 or CONDITIONAL)
17 PROGRAM AUTHORIZATION ... ==> DISABLE (DISABLE, ENABLE)

```

Figura 54. El panel Valores predeterminados para el plan de vinculación

En el siguiente panel, introduzca nuevos valores predeterminados para volver a vincular un plan.

```

DSNEBP11 DEFAULTS FOR REBIND PLAN SSID: DSN
COMMAND ===>

Change default options as necessary:

 1 ISOLATION LEVEL ==> SAME (SAME, RR, RS, CS, or UR)
 2 PLAN VALIDATION TIME ==> SAME (SAME, RUN, or BIND)
 3 RESOURCE RELEASE TIME ... ==> SAME (SAME, DEALLOCATE, COMMIT,
 or INHERITFROMPLAN)

 4 EXPLAIN PATH SELECTION .. ==> SAME (SAME, NO, or YES)
 5 DATA CURRENCY ==> SAME (SAME, NO, or YES)
 6 PARALLEL DEGREE ==> SAME (SAME, 1 or ANY)
 7 REOPTIMIZE FOR INPUT VARS ==> SAME (SAME, ALWAYS, NONE, ONCE, AUTO)
 8 DEFER PREPARE ==> SAME (SAME, NO, YES,
 or INHERITFROMPLAN)

 9 KEEP DYNAMIC SQL
 PAST COMMIT OR ROLLBACK.. ==> SAME (SAME, NO, or YES)
10 APPLICATION ENCODING ... ==> SAME (SAME, Blank, ASCII, EBCDIC,
 UNICODE, or ccsid)

11 OPTIMIZATION HINT ==> > (SAME, 'hint-id')
12 IMMEDIATE WRITE ==> SAME (SAME, NO, YES,
 or INHERITFROMPLAN)
13 SQLRULES ==> SAME (SAME, DB2 or STD)
14 DYNAMIC RULES ==> SAME (SAME, RUN, or BIND)
15 RESOURCE ACQUISITION TIME ==> SAME (SAME, ALLOCATE, or USE)
16 DISCONNECT ==> SAME (SAME, EXPLICIT, AUTOMATIC,
 or CONDITIONAL)
17 PROGRAM AUTHORIZATION ... ==> SAME (SAME, DISABLE, ENABLE)

```

Figura 55. El panel Valores predeterminados para el plan de reasignación

La siguiente tabla enumera los campos de los paquetes Defaults for Bind y Defaults for Rebind, y las opciones de encuadernación y reencuadernación correspondientes.

Tabla 151. Campos de los paneles Valores predeterminados para el plan de encuadernación y Valores predeterminados para el plan de re-encuadernación y opciones de encuadernación o re-encuadernación correspondientes

Nombre de campo	Opción de encuadernación o re-encuadernación
APPLICATION ENCODING	<a href="#">ENCODING</a>
moneda de datos	<a href="#">CURRENTDATA</a>

Tabla 151. Campos de los paneles Valores predeterminados para el plan de encuadernación y Valores predeterminados para el plan de re-encuadernación y opciones de encuadernación o re-encuadernación correspondientes (continuación)

Nombre de campo	Opción de encuadernación o re-encuadernación
DBPROTOCOL	<a href="#">DBPROTOCOL</a>
APLAZAR PREPARAR	<a href="#">DEFER y NODEFER</a>
DISCONNECT	<a href="#">DISCONNECT</a>
NORMAS DINÁMICAS	<a href="#">DYNAMICRULES</a>
EXPLICAR LA SELECCIÓN DE RUTA	<a href="#">EXPLAIN</a>
ESCRITURA INMEDIATA	<a href="#">IMMEDWRITE</a>
ISOLATION LEVEL	<a href="#">ISOLATION</a>
SQL DINÁMICO PASADO CONFIRMADO O REVERSIÓN	<a href="#">KEEPDYNAMIC</a>
Sugerencia de optimización	<a href="#">OPTHINT</a>
GRADO PARALELO	<a href="#">DEGREE</a>
AUTORIZACIÓN DEL PROGRAMA	<a href="#">PROGAUTH</a>
REOPTIMIZAR PARA VARIABLES DE ENTRADA	<a href="#">REOPT</a>
TIEMPO DE ADQUISICIÓN DE RECURSOS	<a href="#">ACQUIRE</a>
HORA DE PUBLICACIÓN DE RECURSOS	<a href="#">RELEASE</a>
HORA DE VALIDACIÓN y HORA DE VALIDACIÓN DEL PLAN	<a href="#">Validate</a>

### Conceptos relacionados

[Opciones de reglas dinámicas para sentencias de SQL dinámico](#)

La opción de vinculación DYNAMICRULES y el entorno de ejecución determinan las reglas para los atributos de SQL dinámico.

[Proceso paralelo \(Db2 Performance\)](#)

[Investigación del rendimiento de SQL utilizando EXPLAIN \(Db2 Performance\)](#)

### Tareas relacionadas

[IDs de autorización de almacenamiento en memoria caché para planes \(Managing Security\)](#)

[Establecimiento del nivel de aislamiento de sentencias SQL en un programa REXX](#)

Los niveles de aislamiento especifican el comportamiento de bloqueo para sentencias SQL. Puede establecer el nivel de aislamiento de las sentencias SQL en su programa REXX en lectura repetible (RR), estabilidad de lectura (RS), estabilidad del cursor (CS) o lectura no confirmada (UR).

[Especificación de las reglas que se aplican a un comportamiento SQL en tiempo de ejecución](#)

Puede especificar si las reglas de Db2 o las reglas SQL estándar se aplican a un comportamiento SQL en tiempo de ejecución.

## Panel Tipos de conexión del sistema

El panel Tipos de conexión del sistema le permite especificar qué tipos de conexiones pueden utilizar un plan o paquete.

Este panel se muestra si introduce SÍ en ¿ACTIVAR/DESACTIVAR CONEXIONES? en los paneles de Paquete o Plan de Encuadernación o Reencuadernación. Para el panel Paquete de encuadernación o reencuadernación, la opción REMOTO no se muestra como en el siguiente panel.

```

DSNEBP13 SYSTEM CONNECTION TYPES FOR BIND ...
SSID: DSN
COMMAND ===>

Select system connection types to be Enabled/Disabled:

1 ENABLE ALL CONNECTION TYPES? ===> (* to enable all types)
or
2 ENABLE/DISABLE SPECIFIC CONNECTION TYPES ===> (E/D)

BATCH ==> (Y/N) SPECIFY CONNECTION NAMES?
DB2CALL ==> (Y/N)
RRSAF ==> (Y/N)
CICS ==> (Y/N) ==> N (Y/N)
IMS ==> (Y/N)
DLIBATCH ==> (Y/N) ==> N (Y/N)
IMSBMP ==> (Y/N) ==> N (Y/N)
IMSMPP ==> (Y/N) ==> N (Y/N)
REMOTE ==> (Y/N)

```

*Figura 56. El panel Tipos de conexión del sistema*

Para habilitar o deshabilitar los tipos de conexión (es decir, permitir o impedir que la conexión ejecute el paquete o plan), introduzca la siguiente información.

### **1 ¿ACTIVAR TODOS LOS TIPOS DE CONEXIÓN?**

Le permite introducir un asterisco (\*) para habilitar todas las conexiones. Después de esa entrada, puede ignorar el resto del panel.

### **2 HABILITAR/DESHABILITAR TIPOS DE CONEXIÓN ESPECÍFICOS**

Le permite especificar una lista de tipos para habilitar o deshabilitar; no puede habilitar algunos tipos y deshabilitar otros en la misma operación. Si enumera los tipos que desea habilitar, introduzca E; esto deshabilita todos los demás tipos de conexión. Si enumera los tipos que desea desactivar, introduzca D; eso habilita todos los demás tipos de conexión.

Para cada tipo de conexión que aparece a continuación, introduzca S (sí) si está en su lista, N (no) si no lo está. Los tipos de conexión son:

- **BATCH** para una conexión TSO
- **DB2CALL** para una conexión CAF
- **RRSAF** para una conexión RRSAF
- **CICS** para una CICS conexión
- **IMS** para todas las IMS conexiones: DLIBATCH, IMSBMP e IMSMPP
- **DLIBATCH** para una conexión DL/I Batch Support Facility
- **IMSBMP** para una IMS conexión a una región BMP
- **IMSMPP** para una IMS conexión a una región MPP o IFP
- **REMOTO** para todas las ubicaciones remotas o sin ubicaciones remotas

Para cada tipo de conexión que tenga una segunda flecha, en ¿ESPECIFICAR NOMBRES DE CONEXIÓN?, introduzca Y si desea enumerar nombres de conexión específicos de ese tipo. Deje N (el valor predeterminado) si no lo hace. Si utiliza Y en cualquiera de esos campos, verá otro panel en el que puede introducir los nombres de conexión.

Si utiliza el comando DISPLAY en TSO en este panel, puede determinar lo que tiene actualmente definido como "habilitado" o "deshabilitado" en su biblioteca DSNSPFT de ISPF (miembro DSNCONNS). La información no refleja el estado actual del catálogo de Db2 .

Si escribe DISPLAY ENABLED en la línea de comandos, obtendrá los nombres de conexión que están actualmente habilitados para sus tipos de conexión TSO. Por ejemplo:

```
Display OF ALL connection name(s) to be ENABLED
```

CONNECTION	SUBSYSTEM
CICS1	ENABLED
CICS2	ENABLED
CICS3	ENABLED
CICS4	ENABLED

DLI1	ENABLED
DLI2	ENABLED
DLI3	ENABLED
DLI4	ENABLED
DLI5	ENABLED

#### **Referencia relacionada**

## Paneles para entrar listas de valores

Algunos campos en paneles DB2I están asociados a palabras claves de mandato que aceptan varios valores. Estos campos le llevan a un panel de lista que le permite entrar o modificar varios valores.

Opciones BIND y REBIND para paquetes, planes y servicios (comandos Db2)

## **Paneles para entrar listas de valores**

Algunos campos en paneles DB2I están asociados a palabras claves de mandato que aceptan varios valores. Estos campos le llevan a un panel de lista que le permite entrar o modificar varios valores.

Un panel de lista se parece a una sesión de edición de e ISPF y le permite desplazarse y utilizar un conjunto limitado de comandos.

El formato de cada panel de lista varía, dependiendo del contenido y el propósito del panel. La siguiente figura muestra una muestra genérica de un panel de lista:

*panelid*            *Specific subcommand function*            SSID: DSN  
COMMAND ==>\_                                                    SCROLL ==>  
  
*Subcommand operand values:*  
  
CMD  
" " " "        *value*   ...  
" " " "        *value*   ...  
" " " "  
" " " "  
" " " "  
" " " "  
" " " "

*Figura 57. Ejemplo genérico de un panel de lista de "DB2I"*

Todos los paneles de listas le permiten introducir comandos limitados en dos lugares:

- En la línea de comandos del sistema, con el prefijo =====>
  - En un área de comando especial, identificada por "===="

En la línea de comandos del sistema, puede utilizar:

END

Guarda todas las variables introducidas, sale de la tabla y continúa el proceso.

CANCELAR

Descarta todas las variables introducidas, finaliza el procesamiento y vuelve al panel anterior.

## **GUARDAR**

Guarda todas las variables introducidas y permanece en la tabla.

En el área de comandos especiales, puede utilizar:

*Posada*

Inserte  $nn$  líneas después de esta.

Pnn

Elimine esta línea y las siguientes  $nn$  líneas.

Rnn

Ripete esta línea  $nn$  veces.

El valor predeterminado para  $nn$  es 1.

Cuando termine con un panel de lista, especifique END para guardar los valores del panel actual y continuar con el procesamiento.

## Preparación del programa: Panel Compilar, Enlazar y Ejecutar

El panel Compilar, Enlazar y Ejecutar le permite realizar los dos últimos pasos en el proceso de preparación del programa (compilar y editar enlace). Este panel también le permite realizar la PL/I MACRO PHASE para programas que requieren esta opción.

Para los programas TSO, el panel también le permite ejecutar programas.

```
DSNEPP02 PROGRAM PREP: COMPILE, PRELINK, LINK, AND RUN SSID: DSN
COMMAND ===>_

Enter compiler or assembler options:
1 INCLUDE LIBRARY ===> SRCLIB.DATA
2 INCLUDE LIBRARY ===>
3 OPTIONS ===> NUM, OPTIMIZE, ADV

Enter linkage editor options:
4 INCLUDE LIBRARY ===> SAMPLIB.COBOL
5 INCLUDE LIBRARY ===>
6 INCLUDE LIBRARY ===>
7 LOAD LIBRARY .. ===> RUNLIB.LOAD
8 PRELINK OPTIONS ===>
9 LINK OPTIONS... ===>
Enter run options:
10 PARAMETERS ===> D01, D02, D03/
11 SYSIN DATA SET ===> TERM
12 SYSPRINT DS ... ===> TERM
```

Figura 58. El panel Preparación del programa: Compilar, vincular y ejecutar

### 1,2 INCLUIR BIBLIOTECA

Le permite especificar hasta dos bibliotecas que contengan miembros para que el compilador las incluya. Los miembros también pueden ser generados desde DCLGEN. Puede dejar estos campos en blanco. No hay ningún valor predeterminado.

### 3 OPCIONES

Le permite especificar las opciones de compilador, ensamblador o procesador de macros PL/I. También puede introducir una lista de opciones de compilador o ensamblador separando las entradas con comas, espacios en blanco o ambos. Puede dejar estos campos en blanco. No hay ningún valor predeterminado.

### 4,5,6 INCLUIR BIBLIOTECA

Le permite introducir los nombres de hasta tres bibliotecas que contengan miembros para que el editor de enlaces los incluya. Puede dejar estos campos en blanco. No hay ningún valor predeterminado.

### 7 CARGAR BIBLIOTECA

Le permite especificar el nombre de la biblioteca que contendrá el módulo de carga. El valor predeterminado es RUNLIB.LOAD.

Si la biblioteca de carga especificada es un PDS y el conjunto de datos de entrada es un PDS, el nombre del miembro especificado en el campo NOMBRE DEL CONJUNTO DE DATOS DE ENTRADA del panel de preparación del programa es el nombre del módulo de carga. Si el conjunto de datos de entrada es secuencial, el segundo calificador del conjunto de datos de entrada es el nombre del módulo de carga.

Debe completar este campo si solicita ENLAZAR o EJECUTAR en el panel de Preparación del programa.

### 8 OPCIONES DE PREVISUALIZACIÓN

Le permite introducir una lista de opciones de preenlazador. Separa los elementos de la lista con comas, espacios en blanco o ambos. Puede dejar este campo en blanco. No hay ningún valor predeterminado.

La utilidad de preenlace solo se aplica a programas que utilizan C, C++ y Enterprise COBOL para z/OS.

## **9 OPCIONES DE ENLACE**

Le permite introducir una lista de opciones de edición de enlaces. Separa los elementos de la lista con comas, espacios en blanco o ambos.

Para preparar un programa que utilice direccionamiento de 31 bits y se ejecute por encima de la línea de 16 megabytes, especifique las siguientes opciones de edición de enlace: AMODE=31, RMODE=ANY.

## **10 PARÁMETROS**

Le permite especificar una lista de parámetros que desea pasar al procesador de tiempo de ejecución de su lenguaje host o a su aplicación. Separa los elementos de la lista con comas, espacios en blanco o ambos. Puede dejar este campo en blanco.

Si está preparando un IMS o CICS programa, debe dejar este campo en blanco; no puede utilizar DB2I para ejecutar IMS y programas CICS programas.

Utilice una barra (/) para separar las opciones de su procesador de tiempo de ejecución de las de su programa.

- Para PL/I y Fortran, los parámetros del procesador de tiempo de ejecución deben aparecer a la izquierda de la barra, y los parámetros de la aplicación deben aparecer a la derecha.

```
run time processor parameters / application parameters
```

- Para COBOL, invierta este orden. Los parámetros del procesador de tiempo de ejecución deben aparecer a la derecha de la barra, y los parámetros de la aplicación deben aparecer a la izquierda.
- Para ensamblador y C, no hay un entorno de tiempo de ejecución compatible, y no es necesario utilizar una barra para pasar parámetros al programa de aplicación.

## **11 CONJUNTO DE DATOS SYSIN**

Le permite especificar el nombre de un conjunto de datos SYSIN (o en Fortran, FT05F001) para su programa de aplicación, si necesita uno. Si no incluye el nombre del conjunto de datos entre apóstrofes, se le añadirá un prefijo (ID de usuario) y un sufijo TSO estándar. El valor predeterminado de este campo es TERM.

Si está preparando un IMS o CICS programa, debe dejar este campo en blanco; no puede utilizar DB2I para ejecutar IMS y programas CICS programas.

## **12 SYSPRINT DS**

Le permite especificar los nombres de un conjunto de datos SYSPRINT (o en Fortran, FT06F001) para su programa de aplicación, si necesita uno. Si no incluye el nombre del conjunto de datos entre apóstrofes, se le añadirá un prefijo (ID de usuario) y un sufijo TSO estándar. El valor predeterminado de este campo es TERM.

Si está preparando un IMS o CICS programa, debe dejar este campo en blanco; no puede utilizar DB2I para ejecutar IMS y programas CICS programas.

Su aplicación podría necesitar otros conjuntos de datos además de SYSIN y SYSPRINT. Si es así, recuerde catalogarlos y asignarlos antes de ejecutar el programa.

Cuando pulsa INTRO después de introducir valores en este panel, Db2 compila y edita la aplicación. Si especificó en el panel de preparación del programa Db2 que desea ejecutar la aplicación, Db2 también ejecuta la aplicación.

### **Referencia relacionada**

[Preenlace de una aplicación \(z/OS Language Environment Programming Guide\)](#)

## **Paneles de DB2I que se utilizan para volver a enlazar y liberar planes y paquetes**

Un conjunto de paneles de DB2I le permite enlazar, volver a enlazar o liberar paquetes.

La siguiente tabla describe paneles adicionales que puede utilizar para reasignar y liberar paquetes y planes. También describe el panel Ejecutar, que puede utilizar para ejecutar programas de aplicación que ya han sido preparados.

*Tabla 152. DB2I paneles utilizados para reasignar y liberar planes y paquetes y utilizados para ejecutar programas de aplicación*

Panel	Descripción del panel
<a href="#">"Panel Enlazar/Volver a enlazar/liberar selección" en la página 991</a>	El panel ENCUADERNAR/REENCUADERNAR/GRATIS le permite seleccionar el proceso ENCUADERNAR, REENCUADERNAR o GRATIS, PLAN, PAQUETE o PAQUETE DE ACTIVACIÓN que necesite.
<a href="#">"Volver a vincular panel de paquete" en la página 992</a>	El panel Volver a encuadrinar paquete le permite cambiar las opciones cuando vuelve a encuadrinar un paquete.
<a href="#">"Reenlazar Panel de activación del paquete" en la página 994</a>	El panel Volver a vincular paquete de activadores le permite cambiar las opciones cuando vuelve a vincular un paquete de activadores.
<a href="#">"Volver a vincular el panel del plan" en la página 997</a>	El panel Volver a vincular plan le permite cambiar las opciones al volver a vincular un plan de aplicación.
<a href="#">"Panel de paquete gratuito" en la página 998</a>	El panel Paquete gratuito le permite cambiar las opciones cuando libera un paquete.
<a href="#">"Panel Liberar plan" en la página 999</a>	El panel Plan gratuito le permite cambiar las opciones cuando libera un plan de aplicación.
<a href="#">"Panel Ejecutar de DB2I" en la página 1002</a>	El panel Ejecutar le permite iniciar un programa de aplicación. Debe utilizar este panel si ya ha preparado el programa y solo desea ejecutarlo.  También puede ejecutar un programa utilizando el panel "Preparación del programa: Compilar, Preenlazar, Enlazar y Ejecutar".

#### Referencia relacionada

[Paneles de DB2I que se utilizan para la preparación de programas](#)

DB2I contiene un conjunto de paneles que le permite preparar una aplicación para su ejecución.

[El menú de la opción primaria de DB2I \(Introducción a Db2 for z/OS\)](#)

## Panel Enlazar/Volver a enlazar/liberar selección

El panel Enlazar/Volver a enlazar/Selección libre permite elegir si desea enlazar, volver a enlazar o seleccionar de manera libre planes y paquetes.

```
DSNEBP01 BIND/REBIND/FREE SSID: DSN
COMMAND ===>_

Select one of the following and press ENTER:

 1 BIND PLAN (Add or replace an application plan)
 2 REBIND PLAN (Rebind existing application plan or plans)
 3 FREE PLAN (Erase application plan or plans)
 4 BIND PACKAGE (Add or replace a package)
 5 REBIND PACKAGE (Rebind existing package or packages)
 6 REBIND TRIGGER PACKAGE (Rebind existing package or packages)
 7 FREE PACKAGE (Erase a package or packages)
```

*Figura 59. El panel de selección Encuadernar/Reencuadernar/Gratis*

Este panel le permite seleccionar el proceso que necesita.

#### **1. VINCULAR PLAN**

Le permite crear un plan de aplicación. Debe tener un plan de aplicación para asignar recursos de Db2 a y admitir solicitudes de SQL durante el tiempo de ejecución. Si selecciona esta opción, se mostrará el panel Enlazar plan. Para obtener más información, consulte “[Panel Enlazar plan](#)” en la página 979.

#### **2 PLAN DE REEMBOLSO**

Le permite reconstruir un plan de aplicación cuando los cambios en él afectan al plan, pero las sentencias SQL del programa son las mismas. Por ejemplo, debe volver a vincular cuando cambie las autorizaciones, cree un nuevo índice que utilice el plan o utilice RUNSTATS. Si selecciona esta opción, se mostrará el panel Volver a vincular plan. Para obtener más información, consulte “[Volver a vincular el panel del plan](#)” en la página 997.

#### **3 PLAN GRATIS**

Le permite eliminar planes de Db2. Si selecciona esta opción, se mostrará el panel Plan gratuito. Para obtener más información, consulte “[Panel Liberar plan](#)” en la página 999.

#### **4. ENVÍO DEL PAQUETE**

Le permite crear un paquete. Si selecciona esta opción, se mostrará el panel Enlazar paquete. Para obtener más información, consulte “[Panel Enlazar paquete](#)” en la página 976.

#### **5 PAQUETE DE REENVÍO**

Le permite reconstruir un paquete cuando los cambios que se le hacen afectan al paquete, pero las sentencias SQL del programa son las mismas. Por ejemplo, debe volver a vincular cuando cambie las autorizaciones, cree un nuevo índice que utilice el paquete o utilice RUNSTATS. Si selecciona esta opción, se mostrará el panel Volver a vincular paquete. Para obtener más información, consulte “[Volver a vincular panel de paquete](#)” en la página 992.

#### **6 PAQUETE DE ACTIVACIÓN DE REBIND**

Le permite reconstruir un paquete de activación cuando necesita cambiar las opciones del paquete. Cuando ejecutas CREATE TRIGGER, Db2 vincula un paquete de activación utilizando un conjunto de opciones predeterminadas. Puede utilizar el PAQUETE DE ACTIVACIÓN DE REBIND para cambiar esas opciones. Por ejemplo, puede utilizar REBIND TRIGGER PACKAGE para cambiar el nivel de aislamiento del paquete de activación. Si selecciona esta opción, se mostrará el panel Paquete de activación de reasignación. Para obtener más información, consulte “[Reenlazar Panel de activación del paquete](#)” en la página 994.

#### **PAQUETE GRATUITO 7**

Le permite eliminar una versión específica de un paquete, todas las versiones de un paquete o colecciones enteras de paquetes de Db2. Si selecciona esta opción, se mostrará el panel Paquete gratuito. Para obtener más información, consulte “[Panel de paquete gratuito](#)” en la página 998.

## **Volver a vincular panel de paquete**

El panel Volver a vincular paquete es el primero de los dos paneles que se utilizan para volver a vincular un paquete. Este panel le permite especificar opciones para reempaquetar el paquete.

La siguiente figura muestra las opciones del paquete de reimpresión.

```

DSNEBP08 REBIND PACKAGE SSID: DSN
COMMAND ==>_

1 Rebind all local packages ==> (* to rebind all packages)

or
 Enter package name(s) to be rebound:
 2 LOCATION NAME ==> (Defaults to local)
 3 COLLECTION-ID ==> > (Required)
 4 PACKAGE-ID ==> > (Required)
 5 VERSION-ID ==>

 6 ADDITIONAL PACKAGES? ==> (*, Blank, (), or version-id)
 (Yes to include more packages)

Enter options as desired ==>
 7 CHANGE CURRENT DEFAULTS?... ==> (NO or YES)
 8 OWNER OF PACKAGE (AUTHID)... ==> > (SAME, new OWNER)
 9 QUALIFIER ==> > (SAME, new QUALIFIER)
 10 ENABLE/DISABLE CONNECTIONS? ==> (NO or YES)
 11 INCLUDE PATH? ==> (SAME, DEFAULT, or YES)

```

Figura 60. El panel Paquete de reasignación

Este panel le permite elegir opciones para reempaquetar un paquete.

#### **1 Vuelva a encuadrinar todos los paquetes locales**

Le permite volver a vincular todos los paquetes en el DBMS local. Para ello, coloque un asterisco (\*) en este campo; de lo contrario, déjelo en blanco.

#### **2 NOMBRE DE LA UBICACIÓN**

Le permite especificar dónde vincular el paquete. Si especifica un nombre de ubicación, debe utilizar entre 1 y 16 caracteres, y debe haberlo definido en la tabla del catálogo SYSIBM.LOCATIONS.

#### **3 IDENTIFICACIÓN DE LA COLECCIÓN**

Le permite especificar la recogida del paquete para volver a empaquetar. Debe especificar un ID de colección de 1 a 128 caracteres, o un asterisco (\*) para volver a vincular todas las colecciones en el sistema local de Db2 . No puede utilizar el asterisco para volver a vincular una colección remota. Este campo se puede desplazar.

#### **4 ID DE PAQUETE**

Le permite especificar el nombre del paquete que desea reenviar. Debe especificar un ID de paquete de 1 a 8 caracteres, o un asterisco (\*), para volver a vincular todos los paquetes de las colecciones especificadas en el sistema local de Db2 . No puede utilizar el asterisco para volver a vincular un paquete remoto.

El campo se puede desplazar y la longitud máxima del campo es de 128.

#### **5 IDENTIFICACIÓN DE VERSIÓN**

Le permite especificar la versión del paquete que desea volver a vincular. Debe especificar un ID de versión de 1 a 64 caracteres, o un asterisco (\*) para volver a vincular todas las versiones en las colecciones y paquetes especificados en el sistema de Db2 local. No puede utilizar el asterisco para volver a vincular una versión remota.

#### **6 ¿PAQUETES ADICIONALES?**

Le permite indicar si desea nombrar más paquetes para volver a vincular. Utilice SÍ para especificar más paquetes en un panel adicional, descrito en “[Paneles para entrar listas de valores](#)” en la página 988. El valor predeterminado es NO.

#### **7 ¿CAMBIAR LOS VALORES PREDETERMINADOS ACTUALES?**

Le permite indicar si desea cambiar los valores predeterminados de encuadernación. Utilice:

**NO (predeterminado )** para conservar los valores predeterminados de enlace del paquete anterior.  
**SÍ** para cambiar los valores predeterminados de encuadernación del paquete anterior. Para obtener información sobre los valores predeterminados para los paquetes vinculantes, consulte “[Paneles Valores predeterminados para BIND PACKAGE y Valores predeterminados REBIND PACKAGE](#)” en la página 982.

## **8. PROPIETARIO DEL PAQUETE (AUTHID)**

Le permite cambiar el ID de autorización del propietario del paquete. El propietario debe tener los privilegios necesarios para ejecutar las sentencias SQL del paquete. El propietario predeterminado del paquete es el propietario existente.

El campo se puede desplazar y la longitud máxima del campo es de 128.

## **9 REQUISITO**

Le permite especificar el esquema predeterminado para todos los nombres de tablas, vistas, índices y alias no cualificados del paquete. Puede especificar un nombre de esquema de 1 a 8 caracteres, que debe ajustarse a las reglas del identificador corto SQL. El valor predeterminado es el nombre del calificador existente.

El campo se puede desplazar y la longitud máxima del campo es de 128.

## **10 ¿ACTIVAR/DEACTIVAR CONEXIONES?**

Le permite especificar si desea habilitar y deshabilitar los tipos de conexiones del sistema para usar con este paquete. Esto solo es válido si el campo NOMBRE DE UBICACIÓN nombra su sistema de e Db2 a local.

Si se introduce SÍ en este campo, se mostrará un panel (que se muestra en [Figura 56 en la página 987](#)) que le permite especificar si varias conexiones del sistema son válidas para esta aplicación.

Los valores predeterminados son los utilizados para el paquete anterior.

## **11 ¿INCLUIR CAMINO?**

Indica cuál de las siguientes acciones desea realizar:

- Solicitar que Db2 utilice los mismos nombres de esquema que cuando el paquete estaba destinado a resolver nombres de tipos distintos no cualificados, funciones definidas por el usuario y procedimientos almacenados en sentencias SQL. Elija IGUAL para realizar esta acción. Este es el valor predeterminado.
- Proporcione una lista de nombres de esquemas que busca Db2 cuando resuelve nombres de tipos distintos no cualificados, funciones definidas por el usuario y procedimientos almacenados en sentencias SQL. Elija SÍ para realizar esta acción.
- Solicitar que Db2 restablezca el Vía SQL a SYSIBM, SYSFUN, SYSPROC y el propietario del paquete. Elija DEFAULT para realizar esta acción.

Si especifica SÍ, Db2 muestra un panel en el que se especifican los nombres de los esquemas que Db2 debe buscar.

### **Referencia relacionada**

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2 \)](#)

## **Reenlazar Panel de activación del paquete**

El panel Paquete de activación de reasignación especifica las opciones para reasignar un paquete de activación.

La siguiente figura muestra esas opciones.

```

DSNEBP19 REBIND TRIGGER PACKAGE SSID: DSN
COMMAND ==>_

1 Rebind all trigger packages ==> (* to rebind all packages)

or
Enter trigger package name(s) to be rebound:
2 LOCATION NAME ==> (Defaults to local)
3 COLLECTION-ID (SCHEMA NAME) ==> > (Required)
4 PACKAGE-ID (TRIGGER NAME)... ==> > (Required)

Enter options as desired ==>
5 ISOLATION LEVEL ==> SAME (SAME, RR, RS, CS, UR, or NC)
6 RESOURCE RELEASE TIME ==> SAME (SAME, DEALLOCATE, or COMMIT)
7 EXPLAIN PATH SELECTION ==> SAME (SAME, NO, or YES)
8 DATA CURRENCY ==> SAME (SAME, NO, or YES)
9 IMMEDIATE WRITE OPTION ==> SAME (SAME, NO, YES)
10 PLAN MANAGEMENT ==> DEFAULT (DEFAULT, BASIC, EXTENDED, OFF)
11 ACCESS PATH REUSE ==> DEFAULT (DEFAULT, ERROR, NONE, or WARN)
12 ACCESS PATH COMPARISON ==> DEFAULT (DEFAULT, ERROR, NONE, or WARN)
13 ACCESS PATH RETAIN DUPS ... ==> DEFAULT (DEFAULT, NO, or YES)
14 SYSTEM_TIME SENSITIVE ==> SAME (SAME, NO, or YES)
15 BUSINESS_TIME SENSITIVE ... ==> SAME (SAME, NO, or YES)
16 ARCHIVE SENSITIVE ==> SAME (SAME, NO, or YES)
17 APPLICATION COMPATIBILITY . ==> SAME (SAME, DB2 function level
 V10R1, V11R1)

```

*Figura 61. El panel Paquete de activación de reasignación*

Este panel le permite elegir opciones para reestructurar un paquete de activación.

### **1. Reencuadernar todos los paquetes de gatillos**

Le permite volver a vincular todos los paquetes en el DBMS local. Para ello, coloque un asterisco (\*) en este campo; de lo contrario, déjelo en blanco.

### **2 NOMBRE DE LA UBICACIÓN**

Le permite especificar dónde vincular el paquete de activación. Si especifica un nombre de ubicación, debe utilizar de 1 a 16 caracteres y debe haberlo definido en la tabla de catálogo SYSIBM.LOCATIONS.

### **3 COLLECTION-ID (NOMBRE DEL ESQUEMA)**

Le permite especificar la colección del paquete desencadenante que se va a volver a vincular. Debe especificar un ID de colección de 1 a 128 caracteres, o un asterisco (\*) para volver a vincular todas las colecciones en el sistema local de Db2 . No puede utilizar el asterisco para volver a vincular una colección remota. Este campo se puede desplazar.

### **4 ID DE PAQUETE**

Le permite especificar el nombre del paquete desencadenante que se va a volver a vincular. Debe especificar un ID de paquete de 1 a 128 caracteres, o un asterisco (\*) para volver a vincular todos los paquetes de activación en las colecciones especificadas en el sistema local Db2 . No puede utilizar el asterisco para volver a vincular un paquete de activación remota. Este campo se puede desplazar.

### **5 NIVEL DE AISLAMIENTO**

Le permite especificar hasta qué punto aislar su aplicación de los efectos de otras aplicaciones en ejecución. El valor predeterminado es el valor utilizado para el paquete de activación anterior.

### **6 HORA DE PUBLICACIÓN DE LOS RECURSOS**

Le permite especificar COMMIT o DEALLOCATE para indicar cuándo liberar bloqueos en recursos. El valor predeterminado es el utilizado para el antiguo paquete de activación.

### **7 EXPLICAR LA SELECCIÓN DE RUTA**

Le permite especificar SÍ o NO para obtener información EXPLICADA sobre cómo se ejecutan las sentencias SQL en el paquete. El valor predeterminado es el valor utilizado para el paquete de activación anterior.

El proceso de enlace inserta información en la tabla *owner.PLAN\_TABLE*, donde *owner* es el ID de autorización del propietario del plan o paquete. Si ha definido *owner.DSN\_STATEMNT\_TABLE*, Db2 también inserta información sobre el coste de la ejecución de la declaración en esa tabla. Si especifica SÍ en este campo y VINCULAR en el campo TIEMPO DE VALIDACIÓN, y si no define correctamente PLAN\_TABLE, el vínculo falla.

## **8 MONEDA DE LOS DATOS**

Le permite especificar SÍ o NO si necesita moneda de datos para cursosres ambiguos abiertos en ubicaciones remotas. El valor predeterminado es el valor utilizado para el paquete de activación anterior.

Los datos están actualizados si los datos dentro de la estructura de host son idénticos a los datos dentro de la tabla base. Los datos están siempre actualizados para el procesamiento local.

## **9 OPCIÓN DE ESCRITURA INMEDIATA**

Especifica cuándo escribe Db2 los cambios para las páginas actualizadas dependientes del grupo de búferes. Este campo solo se aplica a un entorno de intercambio de datos. Los valores que puede especificar son:

### **SAME**

Elija el valor de ESCRITURA INMEDIATA que especificó al vincular el paquete de activación. SAME es el valor predeterminado.

### **NEE**

Escriba los cambios en la fase 1 del proceso de confirmación o antes. Si la transacción se revierte más tarde, escriba los cambios adicionales causados por la reversión al final del proceso de cancelación.

PH1 es equivalente a NO.

### **YES**

Escribir los cambios inmediatamente después de que se actualicen las páginas dependientes del grupo de búferes.

## **10 GESTIÓN DEL PLAN**

Especifica la opción PLANMGMT que se utilizará para volver a enlazar el desencadenador. DEFAULT (predeterminado) significa tomar la configuración predeterminada para esta opción al volver a vincular para el paquete de activación anterior.

## **11 REUTILIZACIÓN DE LA VÍA DE ACCESO**

Especifica la opción APREUSE que se utilizará para volver a enlazar el disparador. DEFAULT (predeterminado) significa tomar la configuración predeterminada para esta opción al volver a vincular para el paquete de activación anterior.

## **12 COMPARACIÓN DE LA RUTA DE ACCESO**

Especifica la opción APCOMPARE que se utilizará para volver a enlazar el disparador. DEFAULT (predeterminado) significa tomar la configuración predeterminada para esta opción al volver a vincular para el paquete de activación anterior.

## **13 ACCESO A LA RUTA CONSERVAR DUPs**

Especifica la opción APRETAINDUP que se utilizará para volver a enlazar el disparador. DEFAULT (predeterminado) significa tomar la configuración predeterminada para esta opción al volver a vincular para el paquete de activación anterior.

## **14 SENSIBLE AL TIEMPO DEL SISTEMA**

Especifica la opción SENSIBLE AL SISTEMA que se utilizará para volver a enlazar el disparador. SAME significa tomar la configuración anterior para esta opción al volver a vincular el paquete de activación antiguo.

## **15 HORARIO\_COMERCIAL SENSIBLE**

Especifica la opción BUSTIMESENSITIVE que se utilizará para reasignar el disparador. SAME significa tomar la configuración anterior para esta opción al volver a vincular el paquete de activación antiguo.

## **16 ARCHIVO SENSIBLE**

Especifica la opción ARCHIVESENSITIVE que se utilizará para volver a enlazar el disparador. SAME significa tomar la configuración anterior para esta opción al volver a vincular el paquete de activación antiguo.

## **17 COMPATIBILIDAD DE APLICACIONES**

Especifica la opción APPLCOMPAT que se utilizará para volver a enlazar el disparador. SAME significa tomar la configuración anterior para esta opción al volver a vincular el paquete de activación antiguo.

## Referencia relacionada

Opciones BIND y REBIND para paquetes, planes y servicios (comandos Db2)

## Volver a vincular el panel del plan

El panel Volver a vincular plan es el primero de los dos paneles que se utilizan para volver a vincular un plan. Este panel le permite especificar opciones para reestructurar el plan.

La siguiente figura muestra las opciones del plan de reasignación.

```
DSNEBP03 REBIND PLAN SSID: DSN
COMMAND ==>_

Enter plan name(s) to be rebound:
1 PLAN NAME ==> (* to rebind all plans)
2 ADDITIONAL PLANS? ==> NO (Yes to include more plans)

Enter options as desired:
3 CHANGE CURRENT DEFAULTS?.... ==> NO (NO or YES)
4 OWNER OF PLAN (AUTHID).... ==> SAME > (SAME, new OWNER)
5 QUALIFIER ==> SAME > (SAME, new QUALIFIER)
6 CACHESIZE ==> SAME (SAME, or value 0-4096)
7 ENABLE/DISABLE CONNECTIONS? ==> NO (NO or YES)
8 INCLUDE PACKAGE LIST?..... ==> SAME (SAME, NO, or YES)
9 CURRENT SERVER ==> (Location name)
10 INCLUDE PATH? ==> SAME (SAME, DEFAULT, or YES)
```

Figura 62. El panel Plan de reasignación

Este panel le permite especificar opciones para reestructurar su plan.

### 1 NOMBRE DEL PLAN

Le permite nombrar el plan de aplicación que desea volver a vincular. Puede especificar un nombre de 1 a 8 caracteres, y el primer carácter debe ser alfabético. No comience el nombre con DSN, porque podría crear conflictos de nombres con Db2. Si no hay errores, el proceso de enlace prepara el plan e introduce su descripción en la tabla EXPLAIN.

Si deja este campo en blanco, el proceso de enlace se produce pero no genera ningún plan.

### 2 PLANES ADICIONALES?

Le permite indicar si desea nombrar más planes para volver a vincular. Utilice SÍ para especificar más planes en un panel adicional, descrito en ["Paneles para entrar listas de valores"](#) en la página 988. El valor predeterminado es NO.

### 3 ¿CAMBIAR LOS VALORES PREDETERMINADOS ACTUALES?

Le permite indicar si desea cambiar los valores predeterminados de encuadernación. Utilice:

**NO (predeterminado)** para conservar los valores predeterminados vinculantes del plan anterior.  
**SÍ** para cambiar los valores predeterminados de encuadernación del plan anterior.

### 4. TITULAR DEL PLAN (AUTHID)

Le permite cambiar el ID de autorización del titular del plan. El propietario debe tener los privilegios necesarios para ejecutar las sentencias SQL en el plan. El propietario del plan existente es el propietario predeterminado.

El campo se puede desplazar y la longitud máxima del campo es de 128.

### 5 REQUISITO

Le permite especificar el esquema predeterminado para todos los nombres de tablas, vistas, índices y alias no cualificados en el plan. Puede especificar un nombre de esquema de 1 a 128 caracteres, que debe ajustarse a las reglas del identificador SQL. El valor predeterminado es el ID de autorización.

Este campo se puede desplazar.

### 6 TAMAÑO DE LA CAJA

Le permite especificar el tamaño (en bytes) de la caché de autorización. Los valores válidos están en el rango de 0 a 4096. Los valores que no son múltiplos de 256 se redondean al siguiente múltiplo más

alto de 256. Un valor de 0 indica que Db2 no utiliza caché de autorización. El valor predeterminado es el tamaño de caché especificado para el plan anterior.

Cada usuario simultáneo de un plan requiere 8 bytes de almacenamiento, con 32 bytes adicionales para gastos generales.

#### **7 ¿ACTIVAR/DESACTIVAR CONEXIONES?**

Le permite especificar si desea habilitar y deshabilitar los tipos de conexiones del sistema para utilizarlos con este plan. Esto solo es válido para la reasignación en su sistema local de Db2 .

Si se introduce SÍ en este campo, se mostrará un panel (que se muestra en [Figura 56 en la página 987](#)) que le permite especificar si varias conexiones del sistema son válidas para esta aplicación.

Los valores predeterminados son los utilizados para el plan anterior.

#### **8 ¿INCLUIR LISTA DE PAQUETES?**

Le permite incluir una lista de colecciones y paquetes en el plan. Si especifica SÍ, se muestra un panel independiente en el que debe introducir la ubicación del paquete, el nombre de la recogida y el nombre del paquete para cada paquete que se vaya a incluir en el plan (consulte “[Paneles para entrar listas de valores](#)” en la página 988). Este campo puede añadir una lista de paquetes a un plan que no tenía una, o reemplazar una lista de paquetes existente.

Puede especificar un nombre de ubicación de 1 a 16 caracteres, un ID de colección de 1 a 18 caracteres y un ID de paquete de 1 a 8 caracteres. Separe dos o más parámetros de la lista de paquetes con una coma. Si especifica un nombre de ubicación, debe estar en la tabla del catálogo SYSIBM.LOCATIONS. La ubicación predeterminada es la lista de paquetes utilizada para el plan anterior.

#### **9 SERVIDOR ACTUAL**

Le permite especificar el servidor inicial para recibir y procesar instrucciones SQL en este plan. Puede especificar un nombre de 1 a 16 caracteres, que debe definir previamente en la tabla del catálogo SYSIBM.LOCATIONS.

Si especifica un servidor remoto, Db2 se conecta a ese servidor cuando se ejecuta la primera instrucción SQL. El valor predeterminado es el nombre del subsistema de Db2 .

#### **10 ¿INCLUIR RUTA?**

Indica cuál de las siguientes acciones desea realizar:

- Solicitar que Db2 utilice los mismos nombres de esquema que cuando el plan se vinculó para resolver nombres de tipos distintos no cualificados, funciones definidas por el usuario y procedimientos almacenados en sentencias SQL. Elija IGUAL para realizar esta acción. Este es el valor predeterminado.
- Proporcione una lista de nombres de esquemas que busca Db2 cuando resuelve nombres de tipos distintos no cualificados, funciones definidas por el usuario y procedimientos almacenados en sentencias SQL. Elija SÍ para realizar esta acción.
- Solicitar que Db2 restablezca el Vía SQL a SYSIBM, SYSFUN, SYSPROC y el propietario del plan. Elija DEFAULT para realizar esta acción.

Si especifica SÍ, Db2 muestra un panel en el que se especifican los nombres de los esquemas que Db2 debe buscar.

#### **Referencia relacionada**

[Paneles Valores predeterminados para BIND PLAN y Valores predeterminados REBIND PLAN](#)  
Estos paneles DB2I le permiten cambiar los valores predeterminados para las opciones BIND PLAN y REBIND PLAN.

[Opciones BIND y REBIND para paquetes, planes y servicios \(comandos Db2 \)](#)

## **Panel de paquete gratuito**

El panel Liberar paquete de DB2I es el primero de dos paneles a través de los cuales puede especificar opciones para liberar un paquete de aplicación.

La siguiente figura muestra las opciones de paquetes gratuitos.

```
DSNEBP18 FREE PACKAGE SSID: DSN
COMMAND ==>_
1 Free ALL packages ==> (* to free authorized packages)
or
2 LOCATION NAME ==> (Defaults to local)
3 COLLECTION-ID ==> > (Required)
4 PACKAGE-ID ==> >(* to free all packages)
5 VERSION-ID ==>
6 ADDITIONAL PACKAGES?..... ==> (*, Blank, (), or version-id)
7 PLAN MANAGEMENT SCOPE ==> (Yes to include more packages)
 (ALL or INACTIVE)
```

Figura 63. El panel del paquete gratuito

Este panel le permite especificar opciones para borrar paquetes.

### 1 Envío GRATIS en TODOS los paquetes

Le permite liberar (borrar) todos los paquetes para los que tiene autorización o para los que tiene autoridad BINDAGENT. Para ello, coloque un asterisco (\*) en este campo; de lo contrario, déjelo en blanco.

### 2 NOMBRE DE LA UBICACIÓN

Le permite especificar el nombre de la ubicación del DBMS para liberar el paquete. Puede especificar un nombre de 1 a 16 caracteres.

### 3 IDENTIFICACIÓN DE LA COLECCIÓN

Le permite especificar la colección de la que desea eliminar paquetes de los que es propietario o tiene privilegios BINDAGENT. Puede especificar un nombre de 1 a 128 caracteres, o un asterisco (\*) para liberar todas las colecciones en el sistema de archivos de Internet local (Db2). No puede utilizar el asterisco para liberar una colección remota. Este campo se puede desplazar.

### 4 ID DE PAQUETE

Le permite especificar el nombre del paquete que desea liberar. Puede especificar un nombre de 1 a 128 caracteres, o un asterisco (\*) para liberar todos los paquetes de las colecciones especificadas en el sistema de correo local (Db2). No puede utilizar el asterisco para liberar un paquete remoto. El nombre que especifique debe estar en las tablas del catálogo de Db2. Este campo se puede desplazar.

### 5 IDENTIFICACIÓN DE VERSIÓN

Le permite especificar la versión del paquete que desea liberar. Puede especificar un identificador de 1 a 64 caracteres, o un asterisco (\*) para liberar todas las versiones de las colecciones y paquetes especificados en el sistema de Db2 local. No puede utilizar el asterisco para liberar una versión remota.

### 6 ¿PAQUETES ADICIONALES?

Le permite indicar si desea nombrar más paquetes para liberar. Utilice SÍ para especificar más paquetes en un panel adicional, descrito en ["Paneles para entrar listas de valores" en la página 988](#). El valor predeterminado es NO.

### 7 ÁMBITO DE LA GESTIÓN DEL PLAN

Especifica si Db2 libera todas las copias del paquete, o solo las copias anteriores y originales inactivas. Este valor corresponde a la opción PLANMGMTSCOPE. El valor predeterminado es Todo.

## Panel Liberar plan

El panel Plan gratuito DB2I es el primero de dos paneles a través de los cuales puede especificar opciones para liberar un plan de aplicación.

[Figura 64 en la página 1000](#) muestra las opciones de planes gratuitos.

```
DSNEBP09 FREE PLAN SSID: DSN
COMMAND ==>_
Enter plan name(s) to be freed:
1 PLAN NAME ==> (* to free all authorized plans)
2 ADDITIONAL PLANS? ==> (Yes to include more plans)
```

Figura 64. El panel del Plan gratuito

Este panel le permite especificar opciones para liberar planos.

#### **1 NOMBRE DEL PLAN**

Le permite nombrar el plan de aplicación que desea eliminar de Db2. Utilice un asterisco para liberar todos los planes para los que tiene autoridad BIND. Puede especificar un nombre de 1 a 8 caracteres, y el primer carácter debe ser alfabético.

Si hay errores, el proceso gratuito finaliza para ese plan y continúa con el siguiente.

#### **¿2 PLANES ADICIONALES?**

Le permite indicar si desea nombrar más planes para liberar. Utilice SÍ para especificar más planes en un panel adicional, descrito en “[Paneles para entrar listas de valores](#)” en la página 988. El valor predeterminado es NO.

# Capítulo 10. Ejecutar una aplicación en Db2 for z/OS

Puede ejecutar su aplicación después de haber procesado las sentencias SQL, compilado y editado el enlace de la aplicación y vinculado la aplicación.

## Acerca de esta tarea

En tiempo de ejecución, Db2 verifica que la información del plan de aplicación y sus paquetes asociados sea coherente con la información correspondiente en el catálogo de Db2 . Si se producen cambios destructivos, como DROP o REVOKE (ya sea en las estructuras de datos a las que accede su aplicación o en la autoridad del vinculador para acceder a esas estructuras de datos), Db2 vuelve a vincular automáticamente los paquetes o el plan según sea necesario.

**Establecimiento de un entorno de prueba :** Este tema describe cómo diseñar una estructura de datos de prueba y cómo llenar tablas con datos de prueba.

**CICS** Antes de ejecutar una aplicación, asegúrese de que se cumplen las dos condiciones siguientes:

- Las entradas correspondientes en las áreas de SNT y RACF áreas de control autorizan la ejecución de su aplicación.
- El programa y su código de transacción se definen en el CICS CSD.

El administrador del sistema es responsable de estas funciones.

## Procesador de mandatos de DSN

El procesador de mandatos de DSN es un procesador de mandatos de TSO que se ejecuta en primer plano de TSO o bajo TSO en un entorno de proceso por lotes iniciado por JES.

Utiliza la función de adjuntar archivos de TSO para acceder a Db2. El procesador de comandos DSN proporciona un método alternativo para ejecutar programas que acceden a Db2 en un entorno TSO.

Cuando ejecutas una aplicación utilizando el procesador de comandos DSN, esa aplicación puede ejecutarse en una conexión de confianza si Db2 encuentra un contexto de confianza coincidente.

Puede utilizar el procesador de comandos DSN implícitamente durante el desarrollo del programa para funciones como:

- Uso del generador de declaraciones (DCLGEN)
- Ejecutar los subcomandos BIND, REBIND y FREE en planes y paquetes de Db2 para su programa
- Utilizar SPUFI (procesador SQL que utiliza entrada de archivos) para probar algunas de las funciones SQL del programa

El procesador de comandos DSN se ejecuta con el programa de monitorización de terminales TSO (TMP). Debido a que el TMP se ejecuta en primer plano o en segundo plano, las aplicaciones DSN se ejecutan de forma interactiva o como trabajos por lotes.

El procesador de comandos DSN puede proporcionar estos servicios a un programa que se ejecute bajo él:

- Conexión automática a Db2
- Atención al cliente
- Traducción de códigos de devolución en mensajes de error

## Limitaciones del procesador de comandos DSN

Cuando se utilizan servicios DSN, la aplicación se ejecuta bajo el control de DSN. Debido a que TSO ejecuta la macro ATTACH para iniciar DSN, y DSN ejecuta la macro ATTACH para iniciar una parte de sí mismo, su aplicación obtiene el control que está dos niveles de tarea por debajo de TSO.

Porque su programa depende de DSN para gestionar su conexión a Db2:

- Si Db2 no está disponible, su aplicación no podrá iniciarse.
- Si se cancela Db2 , también se cancela su solicitud.
- Una aplicación solo puede utilizar un plan.

Si estas limitaciones son demasiado estrictas, considere la posibilidad de que su aplicación utilice la función de adjuntar llamadas o Resource Recovery Services attachment facility. Para obtener más información sobre estos servicios de envío de archivos adjuntos, consulte “[Recurso de conexión de llamada](#)” en la página 42 y “[Recurso de conexión de Resource Recovery Services](#)” en la página 73.

## Procesamiento del código de devolución DSN

Al final de una sesión DSN, register 15 contiene el valor más alto que se coloca allí por cualquier subcomando DSN que se utiliza en la sesión o por cualquier programa que se ejecuta por el subcomando RUN. Su entorno de tiempo de ejecución podría formatear ese valor como un código de retorno. Sin embargo, el valor no se origina en DSN.

### Conceptos relacionados

[Recurso de conexión de TSO \(Introducción a Db2 para z/OS\)](#)

### Referencia relacionada

[DSN comando \(TSO\) \( Db2 Commands\)](#)

### Información relacionada

[Tipos de comandos para Db2 for z/OS \( Db2 Commands\)](#)

## Panel Ejecutar de DB2I

El panel Ejecutar de DB2I le permite iniciar un programa de aplicación que puede contener sentencias SQL.

Solo puede acceder al panel de ejecución a través del menú de opciones principales de DB2I. Puede realizar la misma tarea utilizando "el panel Preparación del programa: Compilar, vincular y ejecutar". Debe utilizar este panel si ya ha preparado el programa y simplemente desea ejecutarlo. [Figura 65 en la página 1002](#) muestra las opciones de ejecución.

```
DSNERP01 RUN SSID: DSN
COMMAND ==>_

Enter the name of the program you want to run:
1 DATA SET NAME ==>
2 PASSWORD..... ==> (Required if data set is password protected)

Enter the following as desired:
3 PARAMETERS .. ==>
4 PLAN NAME ... ==> (Required if different from program name)
5 WHERE TO RUN ==> (BACKGROUND, FOREGROUND, or EDITJCL)
```

Figura 65. El panel Ejecutar

Este panel le permite ejecutar programas de aplicación existentes.

### 1 DATA SET NAME

Le permite especificar el nombre del conjunto de datos particionado que contiene el módulo de carga. Si el módulo se encuentra en un conjunto de datos que el sistema operativo puede encontrar, puede especificar solo el nombre del miembro. No hay ningún valor predeterminado.

Si no incluye el nombre entre comillas, se añadirá un prefijo TSO estándar (ID de usuario) y un sufijo (.LOAD).

### 2 CONTRASEÑA

Le permite especificar la contraseña del conjunto de datos si es necesario. El procesador RUN no comprueba si necesita una contraseña. Si no introduce una contraseña obligatoria, su programa no se ejecutará.

### 3 PARÁMETROS

Le permite especificar una lista de parámetros que desea pasar al procesador de tiempo de ejecución de su lenguaje host o a su aplicación. Debe separar los elementos de la lista con comas, espacios en blanco o ambos. Puede dejar este campo en blanco.

Utilice una barra (/) para separar las opciones de su procesador de tiempo de ejecución de las de su programa.

- Para PL/I y Fortran, los parámetros del procesador de tiempo de ejecución deben aparecer a la izquierda de la barra, y los parámetros de la aplicación deben aparecer a la derecha.

```
run time processor parameters / application parameters
```

- Para COBOL, invierta este orden. los parámetros del procesador de tiempo de ejecución deben aparecer a la derecha de la barra, y los parámetros de la aplicación deben aparecer a la izquierda.
- Para ensamblador y C, no hay un entorno de tiempo de ejecución compatible, y no es necesario utilizar la barra para pasar parámetros al programa de aplicación.

### 4 NOMBRE DEL PLAN

Le permite especificar el nombre del plan al que está vinculado el programa. El valor predeterminado es el nombre de miembro del programa.

### 5 DÓNDE CORRER

Le permite indicar si se ejecuta en primer plano o en segundo plano. También puede especificar EDITJCL, en cuyo caso podrá editar la instrucción de control de trabajo antes de ejecutar el programa. Utilice:

FOREGROUND para ejecutar inmediatamente el programa en primer plano con los valores especificados.

ANTECEDENTES para crear y enviar inmediatamente para ejecutar un archivo que contenga un DSNH CLIST utilizando la instrucción de control JOB desde el Panel 2 de Defaults de DB2I o la salida SUBMIT de su sitio. El programa se ejecuta en segundo plano.

EDITJCL para crear y abrir un archivo que contenga un DSNH CLIST en modo de edición. A continuación, puede enviar el CLIST o guardarla. El programa se ejecuta en segundo plano.

### Ejecución de procesadores de comandos

Para ejecutar un procesador de comandos (CP), utilice los siguientes comandos desde el indicador TSO ready o como TSO TMP:

```
DSN SYSTEM (Db2-subsystem-name)
RUN CP PLAN (plan-name)
```

El subcomando RUN le pide más información. Al final del procesador DSN, utilice el comando END.

## Ejecución de un programa en primer plano en TSO

Utilice el panel RUN (Ejecutar) de DB2I, para ejecutar un programa en primer plano TSO. También puede emitir el comando DSN, seguido del subcomando RUN de DSN.

### Acerca de esta tarea

Antes de ejecutar el programa, asegúrese de asignar los conjuntos de datos que su programa necesita.

El siguiente ejemplo muestra cómo iniciar una aplicación TSO en primer plano. El nombre de la aplicación es SAMPPGM, y ssid es el ID del sistema:

```
TSO Prompt: READY
Enter: DSN SYSTEM(ssid)
DSN Prompt: DSN
Enter: RUN PROGRAM(SAMPPGM) -
 PLAN(SAMPLAN) -
 LIB(SAMPPROJ.SAMPLIB) -
 PARMS('/D01 D02 D03')
: (Here the program runs and might prompt you for input)
DSN Prompt: DSN
```

<b>Enter:</b>	END
TSO Prompt:	READY

Esta secuencia también funciona en ISPF opción 6. Puede empaquetar esta secuencia en un CLIST. Db2 no admite el acceso a múltiples subsistemas de Db2 desde un único espacio de direcciones.

La palabra clave PARMS del subcomando RUN le permite pasar parámetros al procesador en tiempo de ejecución y a su programa de aplicación:

```
PARMS ('/D01, D02, D03')
```

La barra (/) indica que está pasando parámetros. Para algunos idiomas, se pasan parámetros y opciones de tiempo de ejecución en el formulario *PARMS('parámetros/opciones de tiempo de ejecución')*. Un ejemplo de la palabra clave PARMS podría ser:

```
PARMS ('D01, D02, D03/')
```

Consulte sus publicaciones en el idioma de destino para conocer la forma correcta de la opción PARMS.

## Ejecutar una aplicación REXX de e Db2

Usted ejecuta aplicaciones REXX de e Db2 e bajo TSO. No precompila, compila, edita enlaces ni vincula aplicaciones REXX d Db2 antes de ejecutarlas.

### Acerca de esta tarea

En un entorno por lotes, puede utilizar sentencias como estas para invocar la aplicación REXXPROG:

```
//RUNREXX EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSEXEC DD DISP=SHR,DSN=SYSADM.REXX.EXEC
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
%REXXPROG parameters
```

El conjunto de datos SYSEXEC contiene su aplicación REXX, y el conjunto de datos SYSTSIN contiene el comando que utiliza para invocar la aplicación.

## Invocar programas a través del Interactive System Productivity Facility

Puede utilizar ISPF para invocar programas que se conectan a Db2 a través de la función de adjuntar llamadas (CAF).

### Acerca de esta tarea

Los programas de gestión de conexiones de muestra de ISPF /CAF ( DSN8SPM y DSN8SCM ) aprovechan los servicios de ISPLINK SELECT, permitiendo que cada rutina establezca su propia conexión con Db2 y su propio hilo y plan.

Con la misma estructura modular que en el ejemplo anterior, el uso de CAF probablemente proporcione una mayor eficiencia al reducir el número de CLIST. Sin embargo, esto no significa que cualquier función de Db2 se ejecute más rápidamente.

**Desventajas:** En comparación con la estructura modular que utiliza DSN, la estructura que utiliza CAF probablemente requiera un programa más complejo, que a su vez podría requerir subrutinas de lenguaje ensamblador. Para obtener más información, consulte [“Recurso de conexión de llamada”](#) en la página 42.

## ISPF

Interactive System Productivity Facility ( ISPF ) le ayuda a construir y ejecutar diálogos. Db2 incluye una aplicación de muestra que ilustra cómo utilizar ISPF a través de la función de adjuntar llamadas (CAF).

Cada escenario tiene ventajas y desventajas en términos de eficiencia, facilidad de codificación, facilidad de mantenimiento y flexibilidad general.

## Uso de ISPF y el procesador de comandos DSN

Existen algunas restricciones sobre cómo establecer y cortar conexiones con Db2 en cualquier estructura. Si utiliza la opción PGM de SELECT ( ISPF ), ISPF pasa el control a su módulo de carga mediante la macro LINK; si utiliza CMD, ISPF pasa el control mediante la macro ATTACH.

El procesador de comandos DSN solo permite conexiones de bloques de control de tareas (TCB) individuales. Tenga cuidado de no cambiar el TCB después de la primera instrucción SQL. ISPF Los servicios SELECT cambian el TCB si ha iniciado DSN en ISPF, por lo que no puede utilizarlos para pasar el control de un módulo de carga a otro. En su lugar, utilice LINK, XCTL o LOAD.

La siguiente figura muestra los bloques de control de tareas que resultan de adjuntar el procesador de comandos DSN debajo de TSO o ISPF.

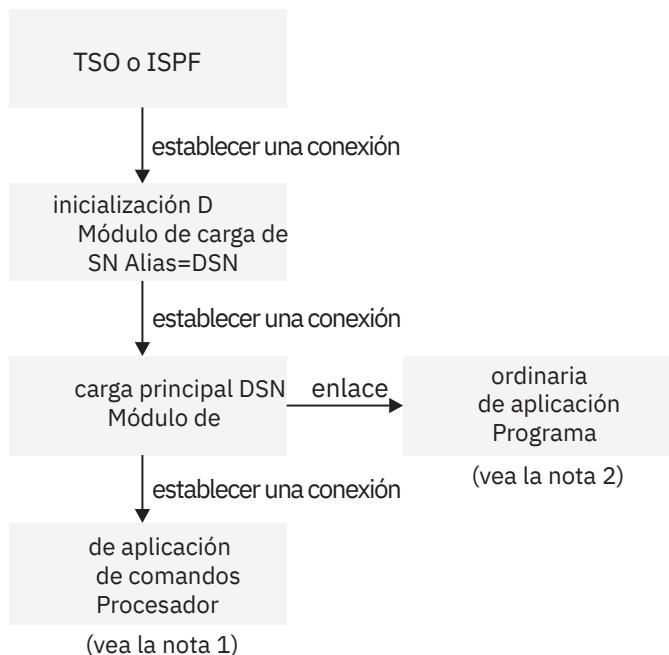


Figura 66. Estructura de tareas DSN

### Notas:

1. El comando RUN con la opción CP hace que DSN adjunte su programa y cree un nuevo TCB.
2. El comando RUN sin la opción CP hace que DSN se vincule a su programa.

Si estás en ISPF y ejecutas bajo DSN, puedes realizar un ISPLINK a otro programa, que llama a un CLIST. A su vez, el CLIST utiliza DSN y otra aplicación. Cada uno de estos usos de DSN crea una unidad de recuperación independiente (proceso o transacción) en Db2.

Todas estas unidades de trabajo DSN iniciadas no están relacionadas, en lo que respecta al aislamiento (bloqueo) y la recuperación (compromiso). Es posible entrar en un punto muerto con uno mismo; es decir, una unidad (DSN) puede solicitar un recurso serializado (una página de datos, por ejemplo) que otra unidad (DSN) posee de forma incompatible.

Un COMMIT en un programa se aplica solo a ese proceso. No hay ninguna facilidad para coordinar los procesos.

### Conceptos relacionados

[SQL dinámico y aplicación ISPF/CAF \(Db2 Installation and Migration\)](#)

[Impresión de opciones para los listados de la aplicación de muestra \(Db2 Installation and Migration\)](#)

[Ejemplos de aplicaciones suministrados con Db2 for z/OS](#)

Db2 proporciona ejemplos de aplicaciones para ayudarle con las técnicas de programación e Db2 es y las prácticas de codificación dentro de cada uno de los cuatro entornos: batch, TSO, IMS, y CICS. Las aplicaciones de ejemplo contienen varias aplicaciones que pueden aplicarse a la gestión de una empresa.

#### Procesador de mandatos de DSN

El procesador de mandatos de DSN es un procesador de mandatos de TSO que se ejecuta en primer plano de TSO o bajo TSO en un entorno de proceso por lotes iniciado por JES.

## Invocación de un único programa SQL a través de ISPF y DSN

Cuando invoque un único programa SQL a través de ISPF y DSN, primero debe invocar ISPF, que muestra los paneles de datos y selección. Cuando seleccionas el programa en el panel de selección, ISPF llama a un CLIST que ejecuta el programa.

#### Acerca de esta tarea

Un CLIST correspondiente podría contener:

```
DSN RUN PROGRAM(MYPROG) PLAN(MYPLAN)
END
```

La aplicación tiene un módulo de carga grande y un plan.

**Desventajas:** Para programas grandes de este tipo, se necesita un diseño más modular, que haga el plan más flexible y fácil de mantener. Si tiene un plan grande, debe volver a vincular todo el plan cada vez que cambie un módulo que incluya instrucciones SQL. Para lograr una construcción más modular cuando todas las partes del programa utilizan SQL, considere el uso de paquetes. Consulte [Capítulo 9, “Preparación de una aplicación para su ejecución en Db2 for z/OS”, en la página 891](#). No puede pasar el control a otro módulo de carga que haga llamadas SQL utilizando ISPLINK; en su lugar, debe utilizar LINK, XCTL o LOAD y BALR.

Si desea utilizar ISPLINK, llame a ISPF para ejecutarlo bajo DSN:

```
DSN RUN PROGRAM(ISPF) PLAN(MYPLAN)
END
```

A continuación, debe salir de ISPF antes de poder iniciar su solicitud.

Además, todo el programa depende de Db2; si Db2 no está en funcionamiento, ninguna parte del programa puede comenzar o seguir ejecutándose.

## Invocación de múltiples programas SQL a través de ISPF y DSN

Puede dividir una aplicación grande en varias funciones diferentes. Cada función se comunica a través de un conjunto común de variables compartidas, que está controlado por ISPF.

#### Acerca de esta tarea

Puede escribir algunas funciones como programas compilados y cargados por separado, otras como EXEC o CLIST. Puede iniciar cualquiera de esos programas o funciones a través del servicio SELECT de ISPF, y puede iniciarlos desde un programa, un CLIST o un panel de selección de ISPF.

Cuando utiliza el servicio SELECT de ISPF, puede especificar si ISPF debe crear un nuevo grupo de variables de ISPF antes de llamar a la función. También puede dividir una aplicación grande en varias partes independientes, cada una con su propio conjunto de variables e ISPF es.

Puede llamar a diferentes partes del programa de diferentes maneras. Por ejemplo, puede utilizar la opción PGM de ISPF SELECCIONAR:

```
PGM(program-name) PARM(parameters)
```

También puede utilizar la opción CMD:

CMD(*command*)

Para una parte que accede a Db2, el comando puede nombrar un CLIST que inicie DSN:

```
DSN
 RUN PROGRAM(PART1) PLAN(PLAN1) PARM(input from panel)
END
```

Dividir la aplicación en módulos separados la hace más flexible y fácil de mantener. Además, algunas de las aplicaciones pueden ser independientes de Db2; las partes de la aplicación que no llaman a Db2 pueden ejecutarse, incluso si Db2 no se está ejecutando. Una base de datos de Db2 o detenida no interfiere con las partes del programa que solo hacen referencia a otras bases de datos.

**Desventajas:** La aplicación modular, en general, tiene que trabajar más. Llama a varios CLIST, y cada uno debe ser localizado, cargado, analizado, interpretado y ejecutado. También establece y rompe conexiones con Db2 con más frecuencia que el módulo de carga único. Como resultado, es posible que pierda algo de eficiencia.

## Carga y ejecución de un programa por lotes

Se puede ejecutar un programa por lotes DL/I ejecutando el módulo DSNMTV01, que carga la aplicación, o ejecutando el programa de aplicación directamente.

### Procedimiento

Para ejecutar un programa que utilice Db2, necesita un plan de Db2 .

El proceso de enlace crea el plan de e Db2 . Db2 primero verifica si el paso de trabajo por lotes DL/I puede conectarse a Db2. Db2 , a continuación, comprueba si el programa de aplicación puede acceder a Db2 y exige la identificación del usuario de los trabajos por lotes que acceden a Db2.

Las dos formas de enviar solicitudes por lotes de DL/I a Db2 son:

- DSNMTV01 puede especificarse como el nombre del programa de aplicación para la región de lotes. Cuando se utiliza este método, el control se otorga a DSNMTV01. Cuando se establece el entorno de ejecución ( Db2 ), el control pasa al programa de aplicación.
- El nombre del programa de aplicación se puede especificar para la región del lote. Se da control a DSNMTV01 directamente para establecer el entorno del subsistema externo para Db2. Cuando se establece el entorno de ejecución de comandos ( Db2 ), el control se transfiere al programa de aplicación especificado en la región de lotes. Para lograrlo, en el procedimiento de inicio de la región por lotes en su JCL de aplicación, especifique la siguiente información:
  - MBR=*application-name*
  - SSM=Db2-subsystem-name

### ejemplos

#### Ejemplo: Envío de una solicitud por lotes de DL/I sin utilizar DSNMTV01

El JCL esqueleto del siguiente ejemplo ilustra un programa de aplicación COBOL, IVP8CP22, que se ejecuta utilizando el soporte de lotes DL/I de Db2 .

```
//TEPCTEST JOB 'USER=ADMF001',MSGCLASS=A,MSGLEVEL=(1,1),
// TIME=1440,CLASS=A,USER=SYSADM,PASSWORD=SYSADM
//*****
//BATCH EXEC DLIBATCH,PSB=IVP8CA,MBR=IVP8CP22,
// BKO=Y,DBRC=N,IRLM=N,SSM=SSDQ
//*****
//SYSPRINT DD SYSOUT=A
//REPORT DD SYSOUT=*
//G.DDOTV02 DD DSN=&TEMP,DISP=(NEW,PASS,DELETE),
// SPACE=(CYL,(10,1),RLSE),
// UNIT=SYSDA,DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092)
//G.DDITV02 DD *
SSDQ,SY51,DSNMIN10,,Q," ,DSNMTES1,,IVP8CP22
//G.SYSIN DD *
```

```

/*
//***** ALWAYS ATTEMPT TO PRINT OUT THE DDOTV02 DATA SET
//***** ****
//PRTLOG EXEC PGM=DFSER10,COND=EVEN
//STEPLIB DD DSN=IMS.RESLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSUT1 DD DSN=&TEMP,DISP=(OLD,DELETE)
//SYSIN DD *
CONTROL CNTL K=000,H=8000
OPTION PRINT
*/

```

### Ejemplo: Envío de una solicitud por lotes de DL/I mediante DSNMTV01

El siguiente ejemplo de JCL esqueleto ilustra un programa de aplicación COBOL, IVP8CP22, que se ejecuta utilizando el soporte de lotes DL/I de Db2.

- El primer paso utiliza el procedimiento estándar DLIBATCH IMS.
- El segundo paso muestra cómo utilizar el programa DFSERA10 IMS para imprimir el contenido del conjunto de datos de salida de DDOTV02.

```

//ISOCS04 JOB 3000,ISOIR,MSGLEVEL=(1,1),NOTIFY=ISOIR,
// MSGCLASS=T,CLASS=A
//JOBLIB DD DISP=SHR,
// DSN=prefix.SDSNLOAD
//***** ****
//** THE FOLLOWING STEP SUBMITS COBOL JOB IVP8CP22, WHICH UPDATES
//** BOTH DB2 AND DL/I DATABASES.
//**
//** ****
//UPDTE EXEC DLIBATCH,DBRC=Y,LOGT=SYSDA,COND=EVEN,
// MBR=DSNMTV01,PSB=IVP8CA,BKO=Y,IRLM=N
//G.STEPLIB DD
// DD DSN=prefix.SDSNLOAD,DISP=SHR
// DD DSN=prefix.RUNLIB.LOAD,DISP=SHR
// DD DSN=CEE.SCEERUN,DISP=SHR
// DD DSN=IMS.PGMLIB,DISP=SHR
//G.DDOTV02 DD DSN=&TEMP1,DISP=(NEW,PASS,DELETE),
// SPACE=(TRK,(1,1),RLSE),UNIT=SYSDA,
// DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092)
//G.DDITV02 DD *
// SSDQ,SYS1,DSNMIN10,,A,-,BATCH001,,IVP8CP22
/*
//***** ****
//*** ALWAYS ATTEMPT TO PRINT OUT THE DDOTV02 DATA SET ***
//***** ****
//STEP3 EXEC PGM=DFSER10,COND=EVEN
//STEPLIB DD DSN=IMS.RESLIB,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSNAME=&TEMP1,DISP=(OLD,DELETE)
//SYSIN DD *
CONTROL CNTL K=000,H=8000
OPTION PRINT
*/

```

### Conceptos relacionados

[Conjuntos de datos de entrada y salida para trabajos por lotes DL/I](#)

Los trabajos por lotes DL/I requieren un conjunto de datos de entrada con el nombre de definición de datos DDITV02 y un conjunto de datos de salida con el nombre de definición de datos DDOTV02.

## Autorización para ejecutar un programa de DL/I por lotes

Cuando una aplicación por lotes DL/I intenta ejecutar la primera instrucción SQL, Db2 comprueba si el ID de autorización tiene el privilegio EXECUTE para el plan. Db2 utiliza el mismo ID para las comprobaciones de autorización posteriores y también identifica los registros de la contabilidad y los rastros de rendimiento.

El ID de autorización principal es el valor del parámetro USER en la declaración de trabajo, si está disponible. Si ese parámetro no está disponible, el ID de autorización principal es el nombre de inicio de

sesión de TSO si se envía el trabajo. De lo contrario, el ID de autorización principal es el IMS Nombre del PSB. En ese caso, sin embargo, el ID no debe comenzar con la cadena "SYSADM" porque esta cadena hace que el trabajo finalice de forma anormal. El trabajo por lotes se rechaza si intenta cambiar el ID de autorización en una rutina de salida.

## Reiniciar un programa por lotes

Para reiniciar un programa por lotes que actualiza datos, primero ejecute la IMS Utilidad Batch Backout, seguido de un trabajo de reinicio que indique el último ID de punto de control correcto.

### Acerca de esta tarea

Para obtener pautas sobre cómo encontrar el último punto de control correcto, consulte [“Búsqueda del ID de punto de comprobación de lotes DL/I”](#) en la página 1010.

### Ejemplos

#### Ejemplo: Salida de lotes con JCL

El siguiente ejemplo de JCL esqueleto ilustra una reversión por lotes para PSB=IVP8CA.

```
//ISOCS04 JOB 3000,ISOIR,MSGLEVEL=(1,1),NOTIFY=ISOIR,
// MSGCLASS=T,CLASS=A
//*****
//* BACKOUT TO LAST CHKPT.
//* IF RC=0028 LOG WITH NO-UPDATE
//* *
//* *
//* *
//BACKOUT EXEC PGM=DFSRRC00,
// PARM='DLI,DFSBB000,IVP8CA,,Y,N,,Y',
// REGION=2600K,COND=EVEN
// *---> DBRC ON
//STEPLIB DD DSN=IMS.RESLIB,DISP=SHR
//IMS DD DSN=IMS.PSBLIB,DISP=SHR
// DD DSN=IMS.DBDLIB,DISP=SHR
//*
//* IMSLOGR DD data set is required
//* IEFRDER DD data set is required
//DFSVSAM DD *
OPTIONS,LTWA=YES
2048,7
1024,7
/*
//SYSIN DD DUMMY
/*
```

#### Ejemplo: reiniciar un trabajo por lotes DL/I con JCL

Los procedimientos operativos pueden reiniciar un paso de trabajo por lotes DL/I para un programa de aplicación utilizando IMS XRST y llamadas simbólicas CKP.

No se puede reiniciar un programa de aplicación BMP en un entorno de lotes DL/I de Db2 . No se accede a los registros simbólicos de los puntos de control, lo que provoca un IMSU0102.

Para reiniciar un trabajo por lotes que se haya terminado de forma anormal o prematura, busque el ID del punto de control del trabajo en el z/OS registro del sistema o en la lista SYSOUT del trabajo fallido. Antes de reiniciar el paso del trabajo, coloque el ID del punto de control en la opción CKPTID=value del procedimiento DLIBATCH, envíe el trabajo. Si se utiliza el nombre de conexión predeterminado (es decir, no especificó la opción de nombre de conexión en el conjunto de datos de entrada de DDTIV02 ), el nombre de trabajo del trabajo de reinicio debe ser el mismo que el del trabajo fallido. Consulte el siguiente ejemplo esquemático, en el que el último valor de ID de punto de control fue IVP80002:

```
//ISOCS04 JOB 3000,OJALA,MSGLEVEL=(1,1),NOTIFY=OJALA,
// MSGCLASS=T,CLASS=A
//*****
//* THE FOLLOWING STEP RESTARTS COBOL PROGRAM IVP8CP22, WHICH UPDATES
//* BOTH DB2 AND DL/I DATABASES, FROM CKPTID=IVP80002.
//*
```

```

//***** ****
//RSTRT EXEC DLIBATCH,DBRC=Y,COND=EVEN,LOGT=SYSDA,
// MBR=DSNMTV01,PSB=IVP8CA,BKO=Y,IRLM=N,CKPTID=IVP80002
//G.STEPLIB DD
// DD
// DD DSN=prefix.SDSNLOAD,DISP=SHR
// DD DSN=prefix.RUNLIB.LOAD,DISP=SHR
// DD DSN=SYS1.COB2LIB,DISP=SHR
// DD DSN=IMS.PGMLIB,DISP=SHR
//* other program libraries
//* G.IEFRDER data set required
//* G.IMSLOGR data set required
//G.DDOTV02 DD DSN=&TEMP2,DISP=(NEW,PASS,DELETE),
// SPACE=(TRK,(1,1),RLSE),UNIT=SYSDA,
// DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092)
//G.DDITV02 DD *
// DB2X,SY51,DSNMIN10,,A,-,BATCH001,,IVP8CP22
/*
//***** ****
//*** ALWAYS ATTEMPT TO PRINT OUT THE DDOTV02 DATA SET ***
//***** ****
//STEP8 EXEC PGM=DFSER10,COND=EVEN
//STEPLIB DD DSN=IMS.RESLIB,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSNAME=&TEMP2,DISP=(OLD,DELETE)
//SYSIN DD *
CONTROL CNTL K=000,H=8000
OPTION PRINT
/*
//

```

## Búsqueda del ID de punto de comprobación de lotes DL/I

Cuando un programa de aplicación emite una llamada IMS CHKP, IMS envía el ID del punto de control a la consola z/OS y la lista SYSOUT en el mensaje DFS0540I.

### Acerca de esta tarea

IMS también registra el ID del punto de control en el registro de tipo X'41' IMS. Las llamadas simbólicas CHKP también crean uno o más registros de tipo X'18' en el IMS registro. XRST utiliza los registros de tipo X'18' para reposicionar las bases de datos DL/I y devolver la información al programa de aplicación.

Durante el proceso de confirmación, el ID del punto de control del programa de aplicación se pasa a Db2. Si se produce un fallo durante el proceso de confirmación, creando una unidad de trabajo indubitable, Db2 recuerda el ID del punto de control. Puede utilizar las siguientes técnicas para encontrar el último ID de punto de control:

- Busque en el listado SYSOUT el paso del trabajo para encontrar el mensaje DFS0540I, que contiene los ID de los puntos de control que se emiten. Utilice el último ID de punto de control de la lista.
- Mira el z/OS registro de la consola para encontrar el mensaje DFS0540I que contiene el ID del punto de control que se emite para este programa por lotes. Utilice el último ID de punto de control de la lista.
- Enviar la IMS Utilidad Batch Backout para retroceder las bases de datos DL/I al último ID de punto de control (predeterminado). Cuando finaliza la retirada del lote, el mensaje DFS395I proporciona el último ID de punto de control válido IMS identificador de punto de control válido. Utilice este ID de punto de control al reiniciar.
- Al reiniciar Db2, ejecute el comando -DISPLAY THREAD(\*) TYPE(INDOUBT) para obtener una posible unidad de trabajo indudable (nombre de conexión e ID de punto de control). Si reinicias el programa de aplicación desde este ID de punto de control, el programa podría funcionar porque el punto de control está registrado en el IMS registro; sin embargo, el programa podría fallar con un IMSU102 de error del usuario porque IMS no terminó de registrar la información antes del fallo. En ese caso, reinicie el programa de aplicación desde el ID del punto de control anterior.

Db2 realiza una de dos acciones automáticamente cuando se reinicia, si el fallo ocurre fuera del período indudable: o bien retrocede la unidad de trabajo al punto de control anterior, o bien confirma los datos sin

ninguna ayuda. Si el operador emite el siguiente comando, no se muestra información sobre la unidad de trabajo:

```
-DISPLAY THREAD(*) TYPE(INDOUBT)
```

## Ejecución de procedimientos almacenados desde el Db2 command line processor

Como alternativa a llamar a un procedimiento almacenado desde un programa de aplicación, puede utilizar el Db2 command line processor para invocar procedimientos almacenados.

### Procedimiento

Para ejecutar un procedimiento almacenado desde el Db2 command line processor:

1. Invoque el Db2 command line processor y conéctese al subsistema Db2 adecuado. Para obtener más información sobre cómo realizar estas tareas, consulte [Db2 command line processor , \( Db2 Commands\)](#).
2. Especifique la instrucción CALL en el formulario que sea aceptable para el Db2 command line processor.

#### Tareas relacionadas

[Invocación de un procedimiento almacenado desde una aplicación](#)

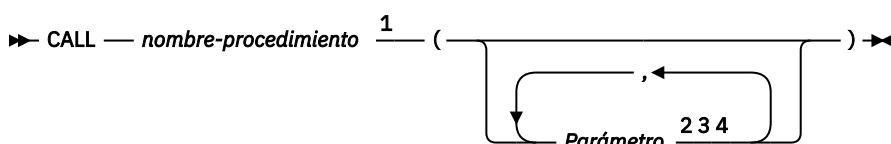
Para ejecutar un procedimiento almacenado, puede llamarlo desde un programa cliente o invocarlo desde el Db2 command line processor.

[Implementación de procedimientos almacenados de Db2 \(Db2 Administration Guide\)](#)

## Db2 command line processor Sentencia CALL

Utilice la instrucción CALL de la biblioteca de procedimientos ( Db2 command line processor ) para invocar procedimientos almacenados desde la biblioteca de funciones ( Db2 command line processor).

Utilice la siguiente sintaxis para la instrucción CALL (llamada) de la función Db2 command line processor .



Notas:

<sup>1</sup> Si especifica un nombre de procedimiento almacenado no cualificado, Db2 busca en la lista de esquemas del registro especial CURRENT PATH. Db2 busca en esta lista un procedimiento almacenado con el número especificado de parámetros de entrada y salida.

<sup>2</sup> Especifique un signo de interrogación (?) como marcador de posición para cada parámetro de salida.

<sup>3</sup> Para los parámetros de entrada no numéricos, BLOB o CLOB, escriba cada valor entre comillas simples (''). La excepción es si los datos son un valor BLOB o CLOB que se va a leer de un archivo. En ese caso, utilice la notación file://fully qualified file name.

<sup>4</sup> Especifique los parámetros de entrada y salida en el orden en que se especifican en la firma del procedimiento almacenado.

### ejemplos

#### Example1

Suponga que el procedimiento almacenado TEST.DEPT\_MEDIAN se creó con la siguiente instrucción:

```
CREATE PROCEDURE TEST.DEPT_MEDIAN
(IN DEPTNUMBER SMALLINT, OUT MEDIANSALARY INT)
```

Para invocar el procedimiento almacenado desde el Db2 command line processor, puede especificar la siguiente instrucción CALL:

```
CALL TEST.DEPT_MEDIAN(51, ?)
```

Supongamos que el procedimiento almacenado devuelve un valor de 25 000. Db2 command line processor:

```
Value of output parameters

Parameter Name : MEDIANSalary
Parameter Value : 25000
```

### Ejemplo 2

Supongamos que el procedimiento almacenado TEST.BLOBSP se define con un parámetro de entrada de tipo BLOB y un parámetro de salida. Puede invocar este procedimiento almacenado desde el Db2 command line processor con la siguiente declaración:

```
CALL TEST.BLOBSP(file:///tmp/photo.bmp,?)
```

Db2 command line processor lee el contenido de /tmp/photo.bmp como parámetro de entrada. De forma alternativa, puede invocar este procedimiento almacenado especificando el parámetro de entrada en la propia instrucción CALL, como en el siguiente ejemplo:

```
CALL TEST.BLOBSP('abcdef',?)
```

## Ejemplo de ejecución de una aplicación de batch Db2 en TSO

La mayoría de los programas de aplicación que están escritos para el entorno por lotes se ejecutan bajo el Programa de Monitor de Terminal (TMP) TSO en modo de fondo.

La siguiente figura muestra las instrucciones JCL que necesita para iniciar un trabajo de este tipo. La lista que sigue explica cada afirmación.

```
//jobname JOB USER=MY DB2ID
//GO EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DSN=prefix.SDSNEXIT,DISP=SHR
// DD DSN=prefix.SDSNLOAD,DISP=SHR
:
//SYSTSPRT DD SYSOUT=A
//SYSTSIN DD *
 DSN SYSTEM (ssid)
 RUN PROG (SAMPPGM) -
 PLAN (SAMPLAN) -
 LIB (SAMPPROJ.SAMPLIB) -
 PARMS ('/D01 D02 D03')
END
/*
```

- La opción TRABAJO identifica esto como una tarjeta de trabajo. La opción USUARIO especifica el ID de autorización de e Db2 s del usuario.
- La instrucción EXEC llama al programa TSO Terminal Monitor Program (TMP).
- La declaración STEPLIB especifica la biblioteca en la que residen los módulos de carga del procesador de comandos DSN y DSNHDECP o un módulo predeterminado de aplicación especificado por el usuario. También puede hacer referencia a las bibliotecas en las que residen las aplicaciones de usuario, las rutinas de salida y el módulo DSNHDECP personalizado. El módulo DSNHDECP personalizado se crea durante la instalación.
- Los extractos de DD posteriores definen archivos adicionales que necesita su programa.
- El comando DSN conecta la aplicación a un subsistema de Db2 e concreto.
- El subcomando RUN especifica el nombre del programa de aplicación que se va a ejecutar.
- La palabra clave PLAN especifica el nombre del plan.
- La palabra clave LIB especifica la biblioteca a la que debe acceder la aplicación.

- La palabra clave PARMS pasa parámetros al procesador en tiempo de ejecución y al programa de aplicación.
- END finaliza el procesador de comandos DSN.

## Notas de uso

- Mantenga cortos los pasos del trabajo DSN.
- **Recomendación:** No utilice DSN para llamar al procesador de comandos EXEC para ejecutar CLISTS que contengan sentencias ISPEXEC; los resultados son impredecibles.
- Si su programa falla o le da un código de retorno distinto de cero, DSN termina.
- Puede utilizar un nombre de archivo adjunto de grupo o de subgrupo en lugar de *un ssid* específico para conectarse a un miembro de un grupo de intercambio de datos.

## Tareas relacionadas

[Ejecución de programas de aplicación TSO \(Db2 Administration Guide\)](#)

## Referencia relacionada

[Ejecución de Terminal Monitor Program \(TSO/E Customization\)](#)

[DSN comando \(TSO\) \( Db2 Commands\)](#)

## Ejemplo de invocación de aplicaciones en un procedimiento de mandatos

Como alternativa a las llamadas en primer plano o por lotes a una aplicación, puede ejecutar una aplicación TSO o por lotes mediante un procedimiento de mandato (CLIST).

El siguiente CLIST llama a un programa de aplicación de Windows ( Db2 ) llamado MYPROG. *ssid* representa el nombre del subsistema de la red de área local ( Db2 ), o el nombre de la conexión de grupo o de la conexión de subgrupo.

```

PROC 0
 DSN SYSTEM(ssid) /* INVOCATION OF DSN FROM A CLIST */
 IF &LASTCC = 0 THEN /* INVOKE DB2 SUBSYSTEM ssid */
 DO /* BE SURE DSN COMMAND WAS SUCCESSFUL */
 DATA /* IF SO THEN DO DSN RUN SUBCOMMAND */
 RUN PROGRAM(MYPROG) /* ELSE OMIT THE FOLLOWING: */
 END /* THE RUN AND THE END ARE FOR DSN */
 ENDDATA /* */ */
 END
EXIT

```

### IMS : Para ejecutar un programa basado en mensajes

En primer lugar, asegúrese de que puede responder a las solicitudes interactivas de datos del programa y de que puede reconocer los resultados esperados. A continuación, introduzca el código de transacción asociado al programa. Los usuarios del código de transacción deben estar autorizados para ejecutar el programa.

### Para ejecutar un programa no basado en mensajes

#### CICS Para ejecutar un programa

En primer lugar, asegúrese de que las entradas correspondientes en las áreas SNT y RACF áreas de control permiten la autorización de ejecución para su aplicación. El administrador del sistema es responsable de estas funciones.

Enviar las declaraciones de control de trabajo que se necesitan para ejecutar el programa.

Además, asegúrese de definir CICS el código de transacción que se asigna a su programa y el programa en sí.

#### Hacer una nueva copia del programa

Emitir el comando NEWCOPY si CICS no se ha reinicializado desde la última vez que se vinculó y compiló el programa.

# Capítulo 11. Prueba y depuración de un programa de aplicación en Db2 for z/OS

Dependiendo de la situación, la prueba del programa de aplicación puede implicar la configuración de un entorno de prueba, la prueba de sentencias, la depuración del programa y la lectura de salida desde el precompilador.

## Tareas relacionadas

[Modelado de un entorno de producción en un subsistema de prueba \(Db2 Performance\)](#)

[Modelado de las estadísticas del sistema de producción en un subsistema de prueba \(Db2 Performance\)](#)

## Diseño de una estructura de datos de prueba

Cuando pruebe una aplicación que acceda a datos de Db2, debe tener datos de Db2 disponibles para la prueba. Para ello, puede crear tablas y vistas de prueba.

### Acerca de esta tarea

- **Vistas de prueba de tablas existentes** : si su aplicación no cambia un conjunto de datos e Db2 es y los datos existen en una o más tablas de nivel de producción, podría considerar el uso de una vista de tablas existentes.
- **Tablas de prueba** : Para crear una tabla de prueba, necesita una base de datos y espacio en la tabla. Hable con su DBA para asegurarse de que dispone de una base de datos y espacios de tablas.

Si los datos que desea cambiar ya existen en una tabla, considere utilizar la cláusula LIKE de CREATE TABLE. Si desea que otras personas además de usted sean propietarias de una tabla con fines de prueba, puede especificar un ID secundario como propietario de la tabla. Puede hacerlo con la instrucción SET CURRENT SQLID.

Si su ubicación tiene un sistema de pruebas independiente (Db2), puede crear las tablas y vistas de prueba en el sistema de pruebas. Esta información asume que usted realiza todas las pruebas en un sistema independiente y que la persona que creó las tablas y vistas de prueba tiene un ID de autorización de TEST. Los nombres de las tablas son TEST.EMP, PRUEBA.PROJ y PRUEBA.DEPT.

### Conceptos relacionados

[ID de autorización \(Gestión de la seguridad\)](#)

## Tareas relacionadas

[Modelado de un entorno de producción en un subsistema de prueba \(Db2 Performance\)](#)

[Modelado de las estadísticas del sistema de producción en un subsistema de prueba \(Db2 Performance\)](#)

### Referencia relacionada

[SET CURRENT SQLID declaración \(Db2 SQL\)](#)

## Analizar las necesidades de datos de la aplicación

Para diseñar pruebas de una aplicación, es necesario determinar el tipo de datos que utiliza la aplicación y cómo accede a ellos.

### Acerca de esta tarea

Esta información asume que usted realiza todas las pruebas en un sistema independiente y que la persona que creó las tablas y vistas de prueba tiene un ID de autorización de TEST. Los nombres de las tablas son TEST.EMP, PRUEBA.PROJ y PRUEBA.DEPT.

Para diseñar tablas y vistas de prueba, primero analice las necesidades de datos de su aplicación.

## Procedimiento

Para analizar las necesidades de datos de su aplicación:

1. Enumere los datos a los que accede su aplicación y describa cómo accede a cada elemento de datos.

Por ejemplo, supongamos que está probando una aplicación que accede a las tablas DSN8C10.EMP, DSN8C10.DEPT y DSN8C10.PROJ. Puede registrar la información sobre los datos como se muestra en [Tabla 153 en la página 1016](#).

Tabla 153. Descripción de los datos de la solicitud

Nombre de tabla o vista	¿Insertar filas?	¿Eliminar filas?	Nombre de columna	Tipo de datos	¿Actualizar acceso?
DSN8C10.EMP	Nee	Nee	EMPNO	CHAR(6)	Nee
			LASTNAME	VARCHAR (15)	Nee
			WORKDEPT	CHAR(3)	Sí
			PHONENO	CHAR(4)	Sí
			JOB	DECIMAL(3)	Sí
DSN8C10.DEPT	Nee	Nee	DEPTNO	CHAR(3)	Nee
			MGRNO	CHAR (6)	Nee
DSN8C10.PROY	Sí	Sí	PROJNO	CHAR(6)	No
			DEPTNO	CHAR(3)	Sí
			RESPEMP	CHAR(6)	Sí
			PRSTAFF	DECIMAL(5,2)	Sí
			PRSTDATE	DECIMAL(6)	Sí
			PRENDATE	DECIMAL(6)	Sí

2. Determinar las tablas y vistas de prueba que necesita para probar su aplicación.

Cree una tabla de prueba en su lista cuando se dé cualquiera de las siguientes condiciones:

- La aplicación modifica los datos de la tabla.
- Debe crear una vista basada en una tabla de prueba porque su aplicación modifica los datos de la vista.

Para continuar con el ejemplo, cree estas tablas de prueba:

- TEST.EMP, con el siguiente formato:

EMPNO	LASTNAME	WORKDEPT	PHONENO	JOB
:	:	:	:	:

- TEST.PROJ, con las mismas columnas y formato que DSN8C10.PROJ, porque la aplicación inserta filas en la tabla DSN8C10.PROJ.

Para apoyar el ejemplo, cree una vista de prueba de la tabla DSN8C10.DEPT.

- PRUEBA.DEPT ver, con el siguiente formato:

DEPTNO	MGRNO
:	:

Dado que la aplicación no modifica ningún dato de la tabla DEPT de DSN8C10, puede basar la vista en la propia tabla (en lugar de en una tabla de prueba). Sin embargo, un enfoque más seguro es tener un

conjunto completo de tablas de prueba y probar el programa a fondo utilizando únicamente datos de prueba.

## Autorización para tablas de prueba y aplicaciones

Antes de poder crear una tabla, debe estar autorizado para crear tablas y para utilizar el espacio de tabla en el que residirá la tabla. También debe tener autoridad para vincular y ejecutar programas que desee probar.

Su DBA puede concederle la autorización necesaria para crear y acceder a tablas, así como para vincular y ejecutar programas.

Si tiene la intención de utilizar tablas y vistas existentes (ya sea directamente o como base para una vista), necesita privilegios para acceder a esas tablas y vistas. Su DBA puede conceder esos privilegios.

Para crear una vista, debe tener autorización para cada tabla y vista en la que se base la vista. Entonces, tendrá los mismos privilegios sobre la vista que sobre las tablas y vistas en las que basó la vista. Antes de probar los ejemplos, pídale a su DBA que le conceda los privilegios para crear nuevas tablas y vistas y para acceder a las tablas existentes. Obtenga de su DBA los nombres de las tablas y vistas a las que está autorizado a acceder (así como los privilegios que tiene para cada tabla).

## Ejemplo de instrucciones SQL para crear una estructura de prueba completa

Debe crear un grupo de almacenamiento, una base de datos, un espacio de tabla y una tabla para utilizarlos como estructura de prueba para su aplicación SQL.

Las siguientes sentencias SQL muestran cómo crear una estructura de prueba completa para contener una pequeña tabla llamada SPUFINUM. La estructura de la prueba consta de:

- Un grupo de almacenamiento llamado SPUFISG
- Una base de datos llamada SPUFIDB
- Un espacio de tabla llamado SPUFITS en la base de datos SPUFIDB y que utiliza el grupo de almacenamiento SPUFISG
- Una tabla llamada SPUFINUM dentro del espacio de tablas SPUFITS

```
CREATE STOGROUP SPUFISG
 VOLUMES (user-volume-number)
 VCAT DSNCAT ;

CREATE DATABASE SPUFIDB ;

CREATE TABLESPACE SPUFITS
 IN SPUFIDB
 USING STOGROUP SPUFISG ;

CREATE TABLE SPUFINUM
 (XVAL CHAR(12) NOT NULL,
 ISFLOAT FLOAT,
 DEC30 DECIMAL(3,0),
 DEC31 DECIMAL(3,1),
 DEC32 DECIMAL(3,2),
 DEC33 DECIMAL(3,3),
 DEC10 DECIMAL(1,0),
 DEC11 DECIMAL(1,1),
 DEC150 DECIMAL(15,0),
 DEC151 DECIMAL(15,1),
 DEC1515 DECIMAL(15,15))
 IN SPUFIDB.SPUFITS ;
```

### Referencia relacionada

[CREATE DATABASE declaración \( Db2 SQL\)](#)

[CREATE STOGROUP declaración \( Db2 SQL\)](#)

[CREATE TABLE declaración \( Db2 SQL\)](#)

[CREATE TABLESPACE declaración \( Db2 SQL\)](#)

# Introducción de datos en las tablas de prueba

Para llenar tablas de prueba, utilice sentencias SQL INSERT o la utilidad LOAD.

## Acerca de esta tarea

Puede introducir datos de prueba en una tabla de varias maneras:

- **INSERTAR... VALUES** (una instrucción SQL) coloca una fila en una tabla cada vez que se ejecuta la instrucción.
- **INSERTAR... SELECT** (una instrucción SQL) obtiene datos de una tabla existente (basada en una cláusula SELECT) y los coloca en la tabla que se identifica en la instrucción INSERT.
- **MERGE** (una instrucción SQL) introduce nuevos datos en una tabla y actualiza los datos existentes.
- La utilidad LOAD obtiene datos de un archivo secuencial (un archivo no eDb2), los formatea para una tabla y los coloca en una tabla.
- El programa UNLOAD de ejemplo de la biblioteca de funciones de Oracle ( Db2 , DSNTIAUL) puede descargar datos de una tabla o vista y crear sentencias de control para la utilidad LOAD.
- La utilidad UNLOAD puede descargar datos de una tabla y crear instrucciones de control para la utilidad LOAD.
- La recuperación redirigida, la utilidad RECOVER con la opción FROM, puede recuperar datos de una tabla de producción a una tabla de prueba sin afectar a la disponibilidad de los datos y aplicaciones de producción. Los datos de producción se pueden recuperar en el objeto de prueba hasta un momento determinado o hasta el estado actual con consistencia transaccional. A continuación, se puede utilizar SQL o la utilidad UNLOAD en los datos de la tabla de prueba.

## Conceptos relacionados

[Ejemplos de aplicaciones suministrados con Db2 for z/OS](#)

Db2 proporciona ejemplos de aplicaciones para ayudarle con las técnicas de programación e Db2 es y las prácticas de codificación dentro de cada uno de los cuatro entornos: batch, TSO, IMS, y CICS. Las aplicaciones de ejemplo contienen varias aplicaciones que pueden aplicarse a la gestión de una empresa.

## Tareas relacionadas

[Inserción de filas utilizando la sentencia INSERT](#)

Una forma de insertar datos en tablas es utilizar la sentencia SQL INSERT. Este método es útil para insertar pequeñas cantidades de datos o insertar datos de otra tabla o vista.

[Inserción de filas de una tabla en otra tabla](#)

Puede copiar una o más filas de una tabla a otra tabla.

[Inserción de datos y actualización de datos en una sola operación](#)

Puede actualizar datos existentes e insertar nuevos datos en una única operación. Esta operación es útil cuando desee actualizar una tabla con un conjunto de filas, algunas de las cuales representan cambios de filas existentes y otras son nuevas filas.

[Ejecución de una recuperación redirigida \(Db2 Utilities\)](#)

## Referencia relacionada

[LOAD \(Db2 Utilities\)](#)

[UNLOAD \(Db2 Utilities\)](#)

# Métodos para probar sentencias SQL

Puede probar sus instrucciones SQL utilizando el Procesamiento SQL mediante entrada de archivos (SPUFI) o el Entorno de procesamiento de SQL ( Db2 command line processor).

**Prueba con SPUFI:** Puede utilizar SPUFI (una interfaz entre ISPF y Db2) para probar instrucciones SQL en un entorno TSO/ ISPF. Con los paneles SPUFI, puede poner instrucciones SQL en un conjunto de datos que e Db2 , posteriormente, ejecuta. El panel principal de SPUFI tiene varias funciones que le permiten:

- Nombrar un conjunto de datos de entrada para contener las sentencias SQL que se pasan a Db2 para su ejecución
- Nombre un conjunto de datos de salida para contener los resultados de la ejecución de las sentencias SQL
- Especificar opciones de procesamiento SPUFI

**Prueba con el Db2 command line processor:** Puede utilizar el Db2 command line processor para probar instrucciones SQL desde los servicios del sistema UNIX en z/OS.

Las sentencias SQL que se ejecutan bajo SPUFI o el "Db2 command line processor" operan sobre tablas reales (en este caso, las tablas que usted creó para probar). Por consiguiente, antes de acceder a datos de Db2 :

- Asegúrese de que todas las tablas y vistas a las que hacen referencia sus sentencias SQL existen.
- Si las tablas o vistas no existen, créelas (o pídale a su administrador de base de datos que las cree). Puede utilizar SPUFI o el procesador de línea de comandos para emitir las instrucciones CREATE que se utilizan para crear las tablas y vistas que necesita para realizar pruebas.

### Conceptos relacionados

[Db2 command line processor , \( Db2 Commands\)](#)

### Tareas relacionadas

[Ejecución de SQL utilizando SPUFI](#)

Puede ejecutar sentencias SQL de forma dinámica en una sesión de TSO utilizando el recurso SPUFI (procesador de SQL que utiliza una entrada de archivo).

## Ejecución de SQL utilizando SPUFI

---

Puede ejecutar sentencias SQL de forma dinámica en una sesión de TSO utilizando el recurso SPUFI (procesador de SQL que utiliza una entrada de archivo).

### Antes de empezar

Antes de utilizar SPUFI, asigne un conjunto de datos de entrada para almacenar las sentencias SQL que desea ejecutar, si dicho conjunto de datos aún no existe.

Antes de comenzar esta tarea, puede especificar si se muestran los ID de mensajes TSO utilizando el comando TSO PROFILE. Para ver los ID de los mensajes, escriba TSO PROFILE MSGID en la línea de comandos de ISPF. Para suprimir los ID de mensajes, escriba TSO PROFILE NOMSGID.

Estas instrucciones asumen que usted dispone de ISPF.

### Acerca de esta tarea

**Importante:** Asegúrese de que el CCSID del terminal TSO coincida con el CCSID del terminal TSO (Db2). Si estos CCSID no coinciden, pueden producirse daños en los datos. Si SPUFI emite el mensaje de advertencia DSNE345I, finalice su sesión SPUFI y notifíquese al administrador del sistema.

SPUFI puede ejecutar instrucciones SQL que recuperan datos gráficos Unicode UTF-16. Sin embargo, SPUFI podría no ser capaz de mostrar algunos caracteres, si esos caracteres no tienen correspondencia en el CCSID EBCDIC SBCS de destino.

### Procedimiento

Para ejecutar SQL mediante SPUFI:

1. Abra SPUFI y especifique las opciones iniciales. Para abrir SPUFI y especificar las opciones iniciales:
  - a) Seleccione SPUFI en el menú de opciones principales de DB2I, como se muestra en [El menú de la opción primaria de DB2I \(Introducción a Db2 for z/OS\)](#).

Se muestra el panel SPUFI.

- b) Especifique el nombre del conjunto de datos de entrada y el nombre del conjunto de datos de salida.

En la siguiente figura se muestra un ejemplo de un panel SPUFI en el que se han especificado un conjunto de datos de entrada y un conjunto de datos de salida.

```
DSNESP01 SPUFI SSID: DSN
====>
Enter the input data set name: (Can be sequential or partitioned)
 1 DATA SET NAME..... ==> EXAMPLES(XMP1)
 2 VOLUME SERIAL.... ==> (Enter if not cataloged)
 3 DATA SET PASSWORD. ==> (Enter if password protected)

Enter the output data set name: (Must be a sequential data set)
 4 DATA SET NAME..... ==> RESULT

Specify processing options:
 5 CHANGE DEFAULTS... ==> Y (Y/N - Display SPUFI defaults panel?)
 6 EDIT INPUT..... ==> Y (Y/N - Enter SQL statements?)
 7 EXECUTE..... ==> Y (Y/N - Execute SQL statements?)
 8 AUTOCOMMIT..... ==> Y (Y/N - Commit after successful run?)
 9 BROWSE OUTPUT..... ==> Y (Y/N - Browse output data set?)

For remote SQL processing:
10 CONNECT LOCATION ==>

PRESS: ENTER to process END to exit HELP for more information
```

Figura 67. El panel SPUFI relleno

- c) Especifique nuevos valores en cualquiera de los otros campos del panel SPUFI.

Para obtener más información sobre estos campos, consulte “El panel SPUFI” en la página 1023.

2. Opcional: Cambie los valores predeterminados de SPUFI, como se describe en “Cambio de los valores predeterminados de SPUFI” en la página 1025.

3. Introduzca instrucciones SQL en SPUFI.

Si el conjunto de datos de entrada que especificó en el panel SPUFI ya contiene todas las instrucciones SQL que desea ejecutar, puede omitir este paso de edición especificando NO para el campo EDITAR ENTRADA en el panel SPUFI. Para introducir instrucciones SQL mediante SPUFI:

- a) Si el panel EDITAR no está ya abierto, en el panel SPUFI, especifique Y en el campo EDITAR ENTRADA y pulse INTRO. Si el conjunto de datos de entrada que ha especificado está vacío, se abre un panel de EDICIÓN vacío. De lo contrario, si el conjunto de datos de entrada contenía instrucciones SQL, dichas instrucciones SQL se muestran en un panel EDITAR.

- b) En el panel EDITAR, utilice el programa EDITAR (ISPF) para introducir o editar cualquier instrucción SQL que desee ejecutar. Mueva el cursor a la primera línea de entrada en blanco e introduzca la primera parte de una instrucción SQL.

Si el conjunto de datos de entrada que especificó en el panel SPUFI ya contiene todas las instrucciones SQL que desea ejecutar, puede omitir este paso de edición especificando NO para el campo EDITAR ENTRADA en el panel SPUFI.

Puede introducir el resto de la instrucción SQL en líneas posteriores, como se muestra en la siguiente figura:

```
EDIT -----userid.EXAMPLES(XMP1) ----- COLUMNS 001 072
COMMAND INPUT ==> SAVE SCROLL ==> PAGE
*****TOP OF DATA *****

000100 SELECT LASTNAME, FIRSTNAME, PHONENO
000200 FROM DSN8C10.EMP
000300 WHERE WORKDEPT= 'D11'
000400 ORDER BY LASTNAME;
*****BOTTOM OF DATA *****
```

Figura 68. El panel de edición: Despu s de introducir una instrucci n SQL

Tenga en cuenta las siguientes reglas y recomendaciones al editar este conjunto de datos de entrada:

- Sangra las líneas e introduce los estados en varias líneas para que sean más fáciles de leer. Introducir sus extractos en varias líneas no cambia la forma en que se procesan sus extractos.
- No ponga más de una instrucción SQL en una sola línea. Si lo hace, la primera sentencia se ejecuta, pero Db2 ignora las demás sentencias SQL de la misma línea. Puede poner más de una instrucción SQL en el conjunto de datos de entrada. Db2 ejecuta las sentencias en el orden en que las colocó en el conjunto de datos.
- Si la longitud de una instrucción SQL es superior a 71 bytes para un conjunto de datos de entrada con una longitud de registro de 79, o a 72 bytes para un conjunto de datos de entrada con una longitud de registro de 80, deberá continuar la instrucción SQL en líneas adicionales del conjunto de datos de entrada SPUFI. SPUFI concatena el texto en varias líneas sin añadir espacios adicionales al final de ninguna línea. Por lo tanto, si una instrucción SQL contiene dos valores con un espacio entre ellos, y el primer valor termina en la última posición de entrada permitida (71 o 72), es necesario añadir un espacio adicional en la siguiente línea antes del segundo valor.

Por ejemplo, supongamos que la longitud de registro de su conjunto de datos de entrada SPUFI es 80, por lo que la longitud máxima de una línea de entrada es 72. Supongamos también que la sentencia SQL que desea introducir tiene 81 bytes de longitud. Los bytes 69 a 81 contienen `FROM MYTABLE ;`. Si divide la instrucción SQL después de `FROM`, la primera línea de la instrucción termina en la columna 72, por lo que debe incluir un espacio en la columna 1 de la siguiente línea, antes de `MYTABLE ;`. De lo contrario, cuando SPUFI concatena las dos líneas, el resultado es `FROMMYTABLE ;`. Cuando SPUFI ejecuta la instrucción SQL, la instrucción SQL falla con SQLCODE -104.

- Termine cada instrucción SQL con el terminador de instrucción que especificó en el panel CURRENT SPUFI DEFAULTS.
- Guarde el conjunto de datos cada 10 minutos aproximadamente introduciendo el comando SAVE.

c) Pulse la tecla END PF.

El conjunto de datos se guarda y se muestra el panel SPUFI.

#### 4. Procesar sentencias SQL con SPUFI.

Puede utilizar SPUFI para enviar las instrucciones SQL en un conjunto de datos a Db2. Para procesar sentencias SQL mediante SPUFI:

- a) En el panel SPUFI, especifique SÍ en el campo EJECUTAR.
- b) Si no acaba de utilizar el panel EDITAR para editar el conjunto de datos de entrada como se describe en "Introducción de sentencias SQL en SPUFI", especifique NO en el campo EDITAR ENTRADA.
- c) Pulse Intro.

SPUFI pasa el conjunto de datos de entrada a Db2 para su procesamiento. Db2 ejecuta la instrucción SQL en el conjunto de datos de entrada y envía la salida al conjunto de datos de salida.

Se abre el conjunto de datos de salida.

Su declaración SQL puede tardar mucho tiempo en ejecutarse, dependiendo del tamaño de la tabla que Db2 deba buscar o del número de filas que Db2 deba procesar. En este caso, puede interrumpir el procesamiento pulsando la tecla " PA1 ". A continuación, responda al mensaje que le pregunta si realmente desea detener el procesamiento. Esta acción cancela la instrucción SQL en ejecución. Dependiendo de la cantidad de datos de entrada que Db2 haya podido procesar antes de que interrumpieras su procesamiento, es posible que Db2 aún no haya abierto el conjunto de datos de salida, o que el conjunto de datos de salida contenga todos o parte de los datos de resultados que se han producido hasta ahora.

## Resultados

### Instrucciones SQL que superan los umbrales de límite de recursos

El administrador del sistema puede utilizar el control de recursos de Db2 (regulador) para establecer límites de tiempo para el procesamiento de instrucciones SQL en SPUFI. Esos límites pueden ser límites de error o límites de advertencia.

Si ejecuta una instrucción SQL a través de SPUFI que se ejecuta más allá de este límite de tiempo de error, SPUFI finaliza el procesamiento de esa instrucción SQL y todas las instrucciones que siguen en el conjunto de datos de entrada de SPUFI. SPUFI muestra un panel que le permite confirmar o deshacer los cambios que haya realizado anteriormente y que no haya confirmado. Ese panel se muestra en la siguiente figura.

```
DSNESP04 SQL STATEMENT RESOURCE LIMIT EXCEEDED SSID: DSN
====>

The following SQL statement has encountered an SQLCODE of -905 or -495:

Statement text

Your SQL statement has exceeded the resource utilization threshold set
by your site administrator.

You must ROLLBACK or COMMIT all the changes made since the last COMMIT.
SPUFI processing for the current input file will terminate immediately
after the COMMIT or ROLLBACK is executed.

1 NEXT ACTION ===> (Enter COMMIT or ROLLBACK)
PRESS: ENTER to process HELP for more information
```

Figura 69. El panel de error de la función de límite de recursos

Si ejecuta una instrucción SQL a través de SPUFI que se ejecuta más allá del límite de tiempo de advertencia para el gobierno predictivo, SPUFI muestra el panel LÍMITE DE RECURSOS DE INSTRUCCIÓN SQL EXCEDIDO. En este panel, puede indicarle a Db2 que continúe ejecutando esa instrucción o que detenga el procesamiento de esa instrucción y continúe con la siguiente instrucción en el conjunto de datos de entrada SPUFI. Ese panel se muestra en la siguiente figura.

```
DSNESP05 SQL STATEMENT RESOURCE LIMIT EXCEEDED SSID: DSN
====>

The following SQL statement has encountered an SQLCODE of 495:

Statement text

You can now either CONTINUE executing this statement or BYPASS the execution
of this statement. SPUFI processing for the current input file will continue
after the CONTINUE or BYPASS processing is completed.

1 NEXT ACTION ===> (Enter CONTINUE or BYPASS)
PRESS: ENTER to process HELP for more information
```

Figura 70. El panel de advertencia de límite de recursos

### **Tareas relacionadas**

[Establecimiento de límites para el uso de recursos de sistema utilizando el recurso de límite de recursos \(Db2 Performance\)](#)

### **Referencia relacionada**

[Utilización de ISPF/PDF para asignar conjuntos de datos \(z/OS ISPF User's Guide Vol II\)](#)

### **Información relacionada**

[Lección 1.1: Consulta interactiva de datos \(Introducción a Db2 para z/OS\)](#)

## **Contenido de un conjunto de datos de entrada SPUFI**

Un conjunto de datos de entrada SPUFI puede contener sentencias SQL, comentarios y sentencias de control SPUFI.

Puede poner comentarios sobre las sentencias SQL en líneas separadas o en la misma línea. En cualquier caso, utilice dos guiones (--) para comenzar un comentario. Especifique cualquier texto que no sea #SET TERMINATOR o #SET TOLWARN después del marcador de comentario. Db2 ignora todo lo demás a la derecha de los dos guiones.

## **El panel SPUFI**

El panel SPUFI es el primer panel que debe completar para ejecutar la aplicación SPUFI.

Después de completar cualquier campo en el panel SPUFI y pulsar Intro, se guardan esos ajustes.

Cuando el panel SPUFI se muestre de nuevo, los campos de entrada de datos del panel contendrán los valores que introdujo anteriormente. Puede especificar nombres de conjuntos de datos y opciones de procesamiento cada vez que se muestre el panel SPUFI, según sea necesario. Los valores que no cambie seguirán vigentes.

Las siguientes descripciones explican los campos que están disponibles en el panel SPUFI.

### **1,2,3 INTRODUCIR DATOS NOMBRE DEL CONJUNTO**

Identifique el conjunto de datos de entrada en los campos 1 a 3. Este conjunto de datos contiene una o más instrucciones SQL que desea ejecutar. Asigne este conjunto de datos antes de utilizar SPUFI, si aún no existe uno. Tenga en cuenta las siguientes reglas:

- El nombre del conjunto de datos debe ajustarse a las convenciones de nomenclatura estándar de TSO.
- El conjunto de datos puede estar vacío antes de que comience la sesión. A continuación, puede añadir las instrucciones SQL editando el conjunto de datos de SPUFI.
- El conjunto de datos puede ser secuencial o particionado, pero debe tener las siguientes características DCB:
  - Un formato de registro (RECFM) de F o FB.
  - Una longitud de registro lógico (LRECL) de 79 u 80. Utilice 80 para cualquier conjunto de datos que el comando EXPORT de QMF no haya creado.
- Los datos del conjunto de datos pueden comenzar en la columna 1. Puede extenderse a la columna 71 si la longitud lógica del registro es 79, y a la columna 72 si la longitud lógica del registro es 80. SPUFI asume que los últimos 8 bytes de cada registro son para números de secuencia.

Si utiliza este panel por segunda vez, el nombre del conjunto de datos que utilizó anteriormente se muestra en el campo DATA SET NAME. Para crear un nuevo miembro de un conjunto de datos particionado existente, cambie solo el nombre del miembro.

### **4 NOMBRE DEL CONJUNTO DE DATOS DE SALIDA**

Introduzca el nombre de un conjunto de datos para recibir el resultado de la instrucción SQL. No es necesario que asigne el conjunto de datos antes de hacer esto.

Si el conjunto de datos existe, el nuevo resultado reemplaza su contenido. Si el conjunto de datos no existe, Db2 asigna un conjunto de datos en el tipo de dispositivo especificado en el panel CURRENT SPUFI DEFAULTS y, a continuación, cataloga el nuevo conjunto de datos. El dispositivo debe ser un

dispositivo de almacenamiento de acceso directo, y usted debe estar autorizado para asignar espacio en ese dispositivo.

Los atributos requeridos para el conjunto de datos de salida son:

- Organización: secuencial
- Formato de grabación: F, FB, FBA, V, VB o VBA
- Longitud de registro: de 80 a 32 768 bytes, no menos que el conjunto de datos de entrada

“Ejecución de SQL utilizando SPUFI” en la página 1019 muestra la opción más sencilla, introduciendo **RESULTADO**. SPUFI asigna un conjunto de datos llamado *userid.RESULT* y envía todos los resultados a ese conjunto de datos. Si ya existe un conjunto de datos llamado *userid.RESULT*, SPUFI le envía una salida de tipo “Db2”, reemplazando todos los datos existentes.

## 5 CHANGE DEFAULTS

Le permite cambiar los valores de control y las características del conjunto de datos de salida y el formato de su sesión SPUFI. Si especifica **SÍ**, puede consultar el panel de valores predeterminados de SPUFI. Visite “Cambio de los valores predeterminados de SPUFI” en la página 1025 para obtener más información sobre los valores que puede especificar y cómo afectan al procesamiento de SPUFI y a las características de salida. No es necesario cambiar los valores predeterminados de SPUFI para este ejemplo.

## 6 EDITAR ENTRADA

Para editar el conjunto de datos de entrada, deje **Y (SÍ)** en la línea 6. Puede utilizar el editor ISPF para crear un nuevo miembro del conjunto de datos de entrada e introducir instrucciones SQL en él. (Para procesar un conjunto de datos que ya contiene un conjunto de sentencias SQL que desea ejecutar inmediatamente, introduzca **N (NO)**). Especificar N omite el paso 3 descrito en “Ejecución de SQL utilizando SPUFI” en la página 1019.)

## 7 EJECUTAR

Para ejecutar las sentencias SQL contenidas en el conjunto de datos de entrada, deje **Y(SÍ)** en la línea 7.

SPUFI gestiona las sentencias SQL que pueden prepararse dinámicamente.

## 8 AUTOCOMMIT

Para que los cambios en los datos de la Db2 sean permanentes, deje **Y (SÍ)** en la línea 8. Especificar **Y** hace que SPUFI emita COMMIT si todas las sentencias se ejecutan correctamente. Si no se ejecutan correctamente todas las instrucciones, SPUFI emite una instrucción ROLLBACK, que elimina los cambios ya realizados en el archivo (volviendo al último punto de confirmación).

Si especifica **N**, Db2 muestra el panel SPUFI COMMIT OR ROLLBACK después de ejecutar el SQL en su conjunto de datos de entrada. Ese panel le pide que CONFIRME, REVERTA o APIQUE cualquier actualización realizada por el SQL. Si introduce DEFER, no se compromete ni se deshace de los cambios.

## 9 NAVEGAR SALIDA

Para ver los resultados de su consulta, deje **Y(SÍ)** en la línea 9. SPUFI guarda los resultados en el conjunto de datos de salida. Puede consultarlos en cualquier momento, hasta que elimine o sobrescriba el conjunto de datos.

## 10 CONECTAR UBICACIÓN

Especifique el nombre del servidor de base de datos, si procede, al que desea enviar instrucciones SQL. A continuación, SPUFI emite una declaración CONNECT de tipo 2 a este servidor.

SPUFI es un paquete vinculado localmente. Las sentencias SQL en el conjunto de datos de entrada solo pueden procesarse si la sentencia CONNECT se ejecuta correctamente. Si la solicitud de conexión falla, el conjunto de datos de salida contiene los códigos de retorno SQL resultantes y los mensajes de error.

### Referencia relacionada

[Acciones permitidas en instrucciones SQL \(Db2 SQL\)](#)

[COMMIT declaración \(Db2 SQL\)](#)

## Cambio de los valores predeterminados de SPUFI

Antes de ejecutar sentencias SQL en SPUFI, puede cambiar el comportamiento de ejecución predeterminado, como el terminador SQL y el nivel de aislamiento.

### Acerca de esta tarea

SPUFI proporciona valores predeterminados la primera vez que se utiliza SPUFI para todas las opciones, excepto el nombre del subsistema de la interfaz de programación de aplicaciones ( Db2 ). Cualquier cambio que realice en estos valores permanecerá vigente hasta que vuelva a cambiarlos.

### Procedimiento

Para cambiar los valores predeterminados de SPUFI:

1. En el panel SPUFI, especifique SÍ en el campo CAMBIAR VALORES PREDETERMINADOS.
2. Pulse Intro.

Se abre el panel CONFIGURACIÓN ACTUAL DE SPUFI. La siguiente figura muestra los valores predeterminados iniciales.

```
DSNESP02 CURRENT SPUFI DEFAULTS SSID: DSN
====>
Enter the following to control your SPUFI session:
 1 SQL TERMINATOR .. ===> ; (SQL Statement Terminator)
 2 ISOLATION LEVEL ===> RR (RR=Repeatable Read, CS=Cursor Stability)
 (UR=Uncommitted Read)
 3 MAX SELECT LINES ===> 250 (Maximum lines to be returned from a SELECT)
 4 ALLOW SQL WARNINGS==> NO (Continue fetching after SQL warning)
 5 CHANGE PLAN NAMES ===> NO (Change the plan names used by SPUFI)
 6 SQL FORMAT ===> SQL (SQL, SQLCOMNT, or SQLPL)
Output data set characteristics:
 7 SPACE UNIT ===> TRK (TRK or CYL)
 8 PRIMARY SPACE ... ===> 5 (Primary space allocation 1-999)
 9 SECONDARY SPACE . ===> 6 (Secondary space allocation 0-999)
10 RECORD LENGTH ... ===> 4092 (LRECL= logical record length)
11 BLOCKSIZE ===> 4096 (Size of one block)
12 RECORD FORMAT.... ===> VB (RECFM= F, FB, FBA, V, VB, or VB)
13 DEVICE TYPE..... ===> SYSDA (Must be a DASD unit name)
Output format characteristics:
14 MAX NUMERIC FIELD ==> 33 (Maximum width for numeric field)
15 MAX CHAR FIELD .. ===> 80 (Maximum width for character field)
16 COLUMN HEADING .. ===> NAMES (NAMES, LABELS, ANY, or BOTH)
```

PRESS: ENTER to process      END to exit      HELP for more information

*Figura 71. El panel predeterminado de SPUFI*

3. Especifique cualquier valor nuevo en los campos de este panel. Todos los campos deben contener un valor.
4. Pulse Intro.

SPUFI guarda los cambios y abre uno de los siguientes paneles o conjuntos de datos:

- El panel CURRENT SPUFI DEFAULTS - PANEL 2. Este panel se abre si especificó SÍ en el campo CAMBIAR NOMBRES DE PLAN.
- Panel EDITAR. Este panel se abre si especificó SÍ en el campo EDITAR ENTRADA en el panel SPUFI.
- Conjunto de datos de salida. Este conjunto de datos se abre si especificó NO en el campo EDITAR ENTRADA del panel SPUFI.
- Panel SPUFI. Este panel se abre si especificó NO para todas las opciones de procesamiento en el panel SPUFI.

Si pulsa la tecla FIN en el panel VALORES PREDETERMINADOS DE SPUFI ACTUALES, se mostrará el panel SPUFI y perderá todos los cambios que haya realizado en el panel VALORES PREDETERMINADOS DE SPUFI ACTUALES.

5. Si se abre el panel VALORES PREDETERMINADOS ACTUALES DE SPUFI - PANEL 2, especifique los valores de los campos de ese panel y pulse Intro. Todos los campos deben contener un valor.

**Importante:** Si especifica un nombre de plan no válido o incorrecto, SPUFI podría experimentar errores operativos o sus datos podrían contaminarse.

SPUFI guarda los cambios y abre uno de los siguientes paneles o conjuntos de datos:

- Panel EDITAR. Este panel se abre si especificó SÍ en el campo EDITAR ENTRADA en el panel SPUFI.
- Conjunto de datos de salida. Este conjunto de datos se abre si especificó NO en el campo EDITAR ENTRADA del panel SPUFI.
- Panel SPUFI. Este panel se abre si especificó NO para todas las opciones de procesamiento en el panel SPUFI.

## Resultados

A continuación, continúe con una de las siguientes tareas:

- Si desea agregar instrucciones SQL al conjunto de datos de entrada o editar las instrucciones SQL en el conjunto de datos de entrada, introduzca las instrucciones SQL en SPUFI.
- De lo contrario, si el conjunto de datos de entrada ya contiene las instrucciones SQL que desea ejecutar, procese las instrucciones SQL con SPUFI.

### Referencia relacionada

#### [Panel CURRENT SPUFI DEFAULTS](#)

Utilice el panel CURRENT SPUFI DEFAULTS para especificar los valores predeterminados SPUFI.

#### [Panel CURRENT SPUFI DEFAULTS - PANEL 2](#)

Utilice el panel CURRENT SPUFI DEFAULTS - PANEL 2 para especificar la información de nombre del plan predeterminado.

## Panel CURRENT SPUFI DEFAULTS

Utilice el panel CURRENT SPUFI DEFAULTS para especificar los valores predeterminados SPUFI.

Las siguientes descripciones explican la información del panel VALORES PREDETERMINADOS DE SPUFI ACTUALES.

### 1 TERMINADOR DE SQL

Especifique el carácter que utiliza para finalizar cada instrucción SQL. Puede especificar cualquier carácter, *excepto* los que figuran en la siguiente tabla. Un punto y coma (;) es el terminador predeterminado de SQL.

*Tabla 154. Caracteres especiales no válidos para el terminador SQL*

Nombre	Carácter	Representación hexadecimal
en blanco		X'40'
coma	,	X'5E'
comillas dobles	"	X'7F'
abrir paréntesis	(	X'4D'
cerrar paréntesis	)	X'5D'
comillas simples	'	X'7D'
subrayado	-	X'6D'

Utilice un carácter que no sea punto y coma si tiene previsto ejecutar una instrucción que contenga puntos y comas incrustados. Por ejemplo, supongamos que elige el carácter # como terminador de la declaración. Entonces, una instrucción CREATE TRIGGER con punto y coma incrustado se parece a la siguiente instrucción:

```
CREATE TRIGGER NEW_HIRe
 AFTER INSERT ON EMP
 FOR EACH ROW MODE DB2SQL
 BEGIN ATOMIC
 UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
 END#
```

Una instrucción CREATE PROCEDURE con punto y coma incrustado tiene el siguiente aspecto:

```
CREATE PROCEDURE PROC1 (IN PARM1 INT, OUT SCODE INT)
 LANGUAGE SQL
 BEGIN
 DECLARE SQLCODE INT;
 DECLARE EXIT HANDLER FOR SQLEXCEPTION
 SET SCODE = SQLCODE;
 UPDATE TBL1 SET COL1 = PARM1;
 END #
```

Tenga cuidado de elegir un carácter para el terminador SQL que no se utilice dentro de la instrucción.

También puede establecer o cambiar el terminador SQL dentro de un conjunto de datos de entrada SPUFI utilizando la instrucción " --#SET TERMINATOR ".

## 2 NIVEL DE AISLAMIENTO

Especifique el nivel de aislamiento para sus sentencias SQL.

## 3 LÍNEAS MÁXIMAS DE SELECCIÓN

El número máximo de filas que puede devolver una instrucción SELECT. Para limitar el número de filas recuperadas, introduzca un número entero mayor que 0.

## 4 PERMITIR ADVERTENCIAS SQL

Introduzca SÍ o NO para indicar si SPUFI continuará procesando una instrucción SQL después de recibir advertencias SQL:

### YES

Si se produce una advertencia cuando SPUFI ejecuta un OPEN o FETCH para una sentencia SELECT, SPUFI continúa procesando la sentencia SELECT.

### NEE

Si se produce una advertencia cuando SPUFI ejecuta un OPEN o FETCH para una sentencia SELECT, SPUFI deja de procesar la sentencia SELECT. Si se produce SQLCODE +802 cuando SPUFI ejecuta un FETCH para una sentencia SELECT, SPUFI continúa procesando la sentencia SELECT.

También puede especificar cómo SPUFI preprocesa la entrada SQL utilizando la instrucción " --#SET TOLWARN ".

## 5 CAMBIAR NOMBRES DE PLAN

Si introduce SÍ en este campo, podrá cambiar los nombres de los planes en un panel de valores predeterminados de SPUFI posterior, DSNESP07. Introduzca SÍ en este campo solo si está seguro de que desea cambiar los nombres de los planes que utiliza SPUFI. Consulte con el administrador del sistema de Db2 si no está seguro de si desea cambiar los nombres de los planes. El uso de un nombre de plan no válido o incorrecto podría provocar errores operativos en SPUFI o podría causar la contaminación de datos.

## 6 FORMATO SQL

Especifique cómo SPUFI preprocesa la entrada SQL antes de pasársela a Db2. Seleccione una de las siguientes opciones:

### SQL

Este es el modo preferido para las sentencias SQL que no sean de lenguaje de procedimiento SQL. Cuando se utiliza esta opción, que es la predeterminada, SPUFI contrae cada línea de una

instrucción SQL en una sola línea antes de pasar la instrucción a Db2. SPUFI también descarta todos los comentarios SQL.

#### **SQLCOMNT**

Este modo es adecuado para todos los SQL, pero está pensado principalmente para el procesamiento de lenguaje procedimental SQL. Cuando esta opción está activa, el comportamiento es similar al modo SQL, excepto que SPUFI no descarta los comentarios SQL. En su lugar, termina automáticamente cada comentario SQL con un carácter de avance de línea (hex 25), a menos que el comentario ya esté terminado por uno o más caracteres de formato de línea. Utilice esta opción para procesar el lenguaje de procedimientos SQL con una modificación mínima por parte de SPUFI.

#### **SQLPL**

Este modo es adecuado para todos los SQL, pero está pensado principalmente para el procesamiento de lenguaje procedimental SQL. Cuando esta opción está activa, SPUFI conserva los comentarios SQL y termina cada línea de una instrucción SQL con un carácter de avance de línea (hex 25) antes de pasar la instrucción a Db2. Las líneas que terminan con un token de división no terminan con un carácter de avance de línea. Utilice este modo para obtener mejores diagnósticos y depuración del lenguaje de procedimientos SQL.

También puede especificar cómo SPUFI preprocesa la entrada SQL utilizando la instrucción " --#SET SQLFORMAT ".

### **7 UNIDAD ESPACIAL**

Especifique cómo se debe asignar el espacio para el conjunto de datos de salida SPUFI.

#### **TRK**

Seguimiento

#### **CYL**

Cilindro

### **8 ESPACIO PRINCIPAL**

Especifique cuántas pistas o cilindros de espacio primario se van a asignar.

### **9 ESPACIO SECUNDARIO**

Especifique cuántas pistas o cilindros de espacio secundario se van a asignar.

### **10 DURACIÓN DE LA GRABACIÓN**

La longitud del registro debe ser de al menos 80 bytes. La longitud máxima de registro depende del tipo de dispositivo que utilice. El valor predeterminado permite un registro de 32756 bytes.

Cada registro puede contener una sola línea de salida. Si una línea es más larga que un registro, la salida se trunca y SPUFI descarta los campos que se extienden más allá de la longitud del registro.

### **11 TAMAÑO DE BLOQUE**

Siga las reglas normales para seleccionar el tamaño del bloque. Para el formato de registro F, el tamaño del bloque es igual a la longitud del registro. Para FB y FBA, elija un tamaño de bloque que sea un múltiplo par de LRECL. Solo para VB y VBA, el tamaño del bloque debe ser 4 bytes mayor que el tamaño del bloque para FB o FBA.

### **12 FORMATO DE REGISTRO**

Especifique F, FB, FBA, V, VB o VBA. Los formatos FBA y VBA insertan un carácter de control de impresora después del número de líneas especificado en el campo LINES/PAGE OF LISTING (LÍNEAS/PÁGINA DE LISTADO) en el panel de valores predeterminados de DB2I. El formato de registro predeterminado es VB (bloque de longitud variable).

### **13 TIPO DE DISPOSITIVO**

Especifique un z/OS nombre para los tipos de dispositivos de almacenamiento de acceso directo. El valor predeterminado es SYSDA. SYSDA especifica que z/OS es seleccionar un dispositivo de almacenamiento de acceso directo adecuado.

### **14 NÚMERO MÁXIMO DE CAMPOS**

El ancho máximo de una columna de valores numéricos en su salida. Elija un valor superior a 0. El valor predeterminado es 33.

## **15 MÁX. CAMPO DE CARGO**

El ancho máximo de una columna de valores de caracteres en su salida. Las cadenas de datos DATETIME y GRAPHIC se representan externamente como caracteres, y SPUFI incluye sus valores predeterminados con los valores predeterminados para los campos de caracteres. Elija un valor superior a 0. El IBM -suministrado por defecto es 250.

## **16 ENCABEZADO DE COLUMNA**

Puede especificar NAMES, LABELS, ANY o BOTH para los encabezados de columna.

- NAMES utiliza únicamente nombres de columna.
- ETIQUETAS (predeterminado) utiliza etiquetas de columna. Deje el título en blanco si no existe ninguna etiqueta.
- CUALQUIERA utiliza etiquetas o nombres de columna existentes.
- BOTH crea dos líneas de título, una con nombres y otra con etiquetas.

Los nombres de columna son los identificadores de columna que puede utilizar en las sentencias SQL. Si una instrucción SQL tiene una cláusula AS para una columna, SPUFI muestra el contenido de la cláusula AS en el encabezado, en lugar del nombre de la columna. Las etiquetas de columna se definen con sentencias LABEL.

### **Conceptos relacionados**

#### Salida de SPUFI

SPUFI da formato y muestra el conjunto de datos de salida utilizando el programa ISPF Browse.

#### **Tareas relacionadas**

##### Cambio de los valores predeterminados de SPUFI

Antes de ejecutar sentencias SQL en SPUFI, puede cambiar el comportamiento de ejecución predeterminado, como el terminador SQL y el nivel de aislamiento.

##### Ejecución de SQL utilizando SPUFI

Puede ejecutar sentencias SQL de forma dinámica en una sesión de TSO utilizando el recurso SPUFI (procesador de SQL que utiliza una entrada de archivo).

## **Panel CURRENT SPUFI DEFAULTS - PANEL 2**

Utilice el panel CURRENT SPUFI DEFAULTS - PANEL 2 para especificar la información de nombre del plan predeterminado.

Este panel se abre si especifica SÍ en el campo CAMBIAR NOMBRES DE PLAN del panel VALORES PREDETERMINADOS ACTUALES DE SPUFI.

Figura 72 en la página 1030 muestra los valores iniciales predeterminados.

```

DSNEP07 CURRENT SPUFI DEFAULTS - PANEL 2 SSID: DSN
==>
Enter the following to control your SPUFI session:
 1 CS ISOLATION PLAN ==> DSNESPCS (Name of plan for CS isolation level)
 2 RR ISOLATION PLAN ==> DSNESPRR (Name of plan for RR isolation level)
 3 UR ISOLATION PLAN ==> DSNESPUR (Name of plan for UR isolation level)

Indicate warning message status:
 4 BLANK CCSID WARNING ==> YES (Show warning if terminal CCSID is blank)

PRESS: ENTER to process END to exit HELP for more information

```

*Figura 72. VALORES PREDETERMINADOS ACTUALES DE SPUFI - PANEL 2*

Las siguientes descripciones explican la información del panel VALORES PREDETERMINADOS ACTUALES DE SPUFI - PANEL 2.

#### **1 PLAN DE AISLAMIENTO DE CS**

Especifique el nombre del plan que utiliza SPUFI cuando especifique un nivel de aislamiento de estabilidad del cursor (CS). Por defecto, este nombre es DSNESPCS.

#### **2 PLAN DE AISLAMIENTO RR**

Especifique el nombre del plan que utiliza SPUFI cuando especifique un nivel de aislamiento de lectura repetible (RR). De forma predeterminada, este nombre es DSNESPRR.

#### **3. SU PLAN DE AISLAMIENTO**

Especifique el nombre del plan que utiliza SPUFI cuando especifique un nivel de aislamiento de lectura no confirmada (UR). De forma predeterminada, este nombre es DSNESPUR.

#### **4 BLANK CCSID ALERT**

Indique si desea recibir el mensaje DSNE345I cuando la configuración del CCSID del terminal esté en blanco. Se produce una configuración de CCSID de terminal en blanco cuando no se puede consultar la página de códigos y el juego de caracteres del terminal o si no son compatibles con ISPF.

**Recomendación :** Para evitar una posible contaminación de datos, utilice la configuración predeterminada de SÍ, a menos que el administrador del sistema de su Db2 le indique específicamente que utilice NO.

## **Establecimiento del carácter terminador de SQL en un conjunto de datos de entrada SPUFI**

En un conjunto de datos de entrada SPUFI, puede sustituir el carácter terminador especificado en el panel CURRENT SPUFI DEFAULTS. El carácter terminador SQL predeterminado es un punto y coma (;).

### **Acerca de esta tarea**

Anular el carácter de terminación SQL predeterminado es útil si necesita utilizar un carácter de terminación SQL diferente para una instrucción SQL concreta.

Para establecer el carácter terminador SQL en un conjunto de datos de entrada SPUFI, especifique el texto --#SET TERMINATOR *character* antes de la instrucción SQL a la que desea aplicar este carácter. Este texto especifica que SPUFI debe interpretar *el carácter* como un terminador de instrucción. Puede especificar cualquier carácter de un solo byte, **excepto** los caracteres que se enumeran en [Tabla 155](#) en la [página 1031](#). Elija un carácter para el terminador SQL que no se utilice dentro de la instrucción. El

terminador que especifique anula un terminador que haya especificado en la opción 1 del panel CURRENT SPUFI DEFAULTS o en una instrucción --#SET TERMINATOR anterior.

Tabla 155. Carácteres especiales no válidos para el terminador SQL

Nombre	Carácter	Representación hexadecimal
en blanco		X'40'
coma	,	X'5E'
comillas dobles	"	X'7F'
abrir paréntesis	(	X'4D'
cerrar paréntesis	)	X'5D'
comillas simples	'	X'7D'
subrayado	_	X'6D'

Utilice un carácter que no sea punto y coma si tiene previsto ejecutar una instrucción que contenga puntos y comas incrustados. Por ejemplo, supongamos que elige el carácter # como terminador de la declaración. En este caso, una instrucción CREATE TRIGGER con punto y coma incrustado tiene este aspecto:

```
CREATE TRIGGER NEW_HIRe
AFTER INSERT ON EMP
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
 UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
END#
```

## Control de la tolerancia de avisos en SPUFI

Cuando utilice SPUFI, puede especificar la acción que SPUFI debe realizar cuando se produce un aviso.

### Acerca de esta tarea

Para controlar la tolerancia de las advertencias, especifique una de las siguientes sentencias de control TOLWARN:

#### --#SET TOLWARN NO

Si se produce una advertencia cuando SPUFI ejecuta una instrucción OPEN o FETCH para SELECT, SPUFI detiene el procesamiento de la instrucción SELECT. Si se produce SQLCODE +802 cuando SPUFI ejecuta un FETCH para una sentencia SELECT, SPUFI continúa procesando la sentencia SELECT.

#### --#SET TOLWARN YES

Si se produce una advertencia cuando SPUFI ejecuta una instrucción OPEN o FETCH para SELECT, SPUFI continúa procesando la instrucción SELECT.

#### --#SET TOLWARN QUIET

Igual que YES, excepto que SPUFI suprime todos los mensajes de advertencia SQL de OPEN o FETCH si el SQLCODE es 0 o mayor.

### Ejemplo

El siguiente ejemplo activa y luego desactiva la tolerancia de las advertencias SQL:

```
SELECT * FROM MY.T1;
--#SET TOLWARN YES
SELECT * FROM YOUR.T1;
--#SET TOLWARN NO
```

## Salida de SPUFI

SPUFI da formato y muestra el conjunto de datos de salida utilizando el programa ISPF Browse.

Un conjunto de datos de salida contiene los siguientes elementos para cada instrucción SQL que ejecuta Db2 :

- La instrucción SQL ejecutada, copiada del conjunto de datos de entrada
- Los resultados de ejecutar la instrucción SQL.
- El SQLCODE y, si la ejecución no tiene éxito, el SQLCA formateado.

Al final del conjunto de datos hay estadísticas resumidas que describen el procesamiento del conjunto de datos de entrada en su conjunto.

Para las sentencias SELECT que se ejecutan con SPUFI, el mensaje "SQLCODE IS 100" indica un resultado sin errores. Si el mensaje SQLCODE IS 100 es el único resultado, Db2 no puede encontrar ninguna fila que satisfaga la condición especificada en la instrucción.

Para todos los demás tipos de instrucciones SQL que se ejecutan con SPUFI, el mensaje "SQLCODE IS 0" indica un resultado sin errores.

### Ejemplo de salida SPUFI

Por ejemplo, el programa de muestra devuelve el siguiente resultado.

```
BROWSE-- userid.RESULT COLUMNS 001 072
COMMAND INPUT ==> SCROLL ==> PAGE
-----+-----+-----+-----+-----+-----+-----+
SELECT LASTNAME, FIRSTNME, PHONENO 00010000
 FROM DSN8C10.EMP 00020000
 WHERE WORKDEPT = 'D11' 00030000
 ORDER BY LASTNAME; 00040000
-----+-----+-----+-----+-----+-----+-----+
LASTNAME FIRSTNME PHONENO
ADAMSON BRUCE 4510
BROWN DAVID 4501
JOHN REBA 0672
JONES WILLIAM 0942
LUTZ JENNIFER 0672
PIANKA ELIZABETH 3782
SCOUTTEN MARILYN 1682
STERN IRVING 6423
WALKER JAMES 2986
YAMAMOTO KIYOSHI 2890
YOSHIMURA MASATOSHI 2890
DSNE610I NUMBER OF ROWS DISPLAYED IS 11
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+
DSNE617I COMMIT PERFORMED, SQLCODE IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+-----+
DSNE601I SQL STATEMENTS ASSUMED TO BE BETWEEN COLUMNS 1 AND 72
DSNE620I NUMBER OF SQL STATEMENTS PROCESSED IS 1
DSNE621I NUMBER OF INPUT RECORDS READ IS 4
DSNE622I NUMBER OF OUTPUT RECORDS WRITTEN IS 30
```

Figura 73. Conjunto de datos resultantes del problema de muestra

### Reglas de formato para resultados de la instrucción SELECT en SPUFI

Los resultados de las sentencias SELECT siguen estas reglas:

- Si los datos numéricos o de caracteres de una columna no se pueden mostrar por completo:
  - Los valores de caracteres y los valores binarios demasiado amplios se truncan a la derecha.
  - Los valores numéricos demasiado amplios se muestran como asteriscos (\*).
  - Para las columnas que no sean LOB y XML, si se produce un truncamiento, el conjunto de datos de salida contiene un mensaje de advertencia. Debido a que las columnas LOB y XML son generalmente

más largas que el valor que elige para el campo MAX CHAR FIELD en el panel CURRENT SPUFI DEFAULTS, SPUFI no muestra ningún mensaje de advertencia cuando trunca la salida de la columna LOB o XML.

Puede cambiar la cantidad de datos que se muestran para las columnas numéricas y de caracteres cambiando los valores en el panel CURRENT SPUFI DEFAULTS, como se describe en “[Cambio de los valores predeterminados de SPUFI](#)” en la página 1025.

- Un valor nulo se muestra como una serie de guiones (-).
- Un valor de columna ROWID, BLOB, BINARY o VARBINARY se muestra en hexadecimal.
- Un valor de columna CLOB se muestra de la misma manera que un valor de columna VARCHAR.
- Un valor de columna DBCLOB se muestra de la misma manera que un valor de columna VARGRAPHIC.
- Una columna XML se muestra de la misma manera que una columna LOB.
- Un encabezado identifica cada columna seleccionada y se repite en la parte superior de cada página de salida. El contenido del encabezado depende del valor que especificó en el campo ENCABEZADO DE COLUMNA del panel VALORES PREDETERMINADOS ACTUALES DE SPUFI.

## Contenido de los mensajes de SPUFI

Los mensajes en la salida SPUFI contienen la siguiente información:

- El SQLCODE y, si la ejecución no tiene éxito, el SQLCA formateado. Si se producen varias condiciones SQL, SPUFI devuelve primero el error SQLCODE, seguido de cualquier condición SQLCODE anterior que se haya producido al ejecutar la sentencia SQL de entrada. Si el SQLCODE final es un error, la mayoría de las advertencias SQLCODE anteriores pueden ignorarse.
- Qué posiciones de caracteres del conjunto de datos de entrada escaneó SPUFI para encontrar sentencias SQL. Esta información le ayuda a comprobar las suposiciones que SPUFI hizo sobre la ubicación de los números de línea (si los hay) en su conjunto de datos de entrada.
- Algunas estadísticas generales:
  - Número de sentencias SQL que se procesan
  - Número de registros de entrada que se leen (del conjunto de datos de entrada)
  - Número de registros de salida que se escriben (en el conjunto de datos de salida).

Otros mensajes que podría recibir del procesamiento de instrucciones SQL incluyen:

- El número de filas que procesó Db2 , que:
  - Su operación de selección recuperada
  - Su operación de actualización modificada
  - Su operación de inserción añadida a una tabla
  - Su operación de eliminación eliminada de una tabla
- En qué columnas se muestran datos truncados porque los datos eran demasiado anchos

## Probar una función externa definida por el usuario

Algunas herramientas de depuración de uso común, como TSO TEST, no están disponibles en el entorno donde se ejecutan las funciones definidas por el usuario. Debe utilizar estrategias de prueba alternativas.

## Probar una función definida por el usuario mediante z/OS Debugger

Puede utilizar z/OS Debugger para probar rutinas de programación ( Db2 for z/OS ), incluidas funciones definidas por el usuario, que estén escritas en cualquiera de los lenguajes compatibles. z/OS Debugger funciona con el Entorno de idiomas.

## Acerca de esta tarea

Puede utilizar z/OS Debugger de forma interactiva o en modo por lotes. Para probar su función definida por el usuario utilizando z/OS Debugger, debe tener instalado z/OS Debugger en el z/OS sistema donde se ejecuta la función definida por el usuario.

## Procedimiento

Para depurar su rutina con z/OS Debugger, utilice uno de los siguientes métodos:

- Utilice z/OS Debugger de forma interactiva siguiendo los pasos que se indican a continuación.
  - a) Compile la rutina con la opción TEST. La opción TEST coloca información en el programa que utiliza z/OS Debugger durante una sesión de depuración.
  - b) Invocar a z/OS Debugger.

Una forma de hacerlo es especificar la opción TEST del entorno de ejecución de Language Environment. La opción TEST controla cuándo y cómo se invoca z/OS Debugger. El lugar más conveniente para especificar las opciones de tiempo de ejecución es en la cláusula RUN OPTIONS de la declaración CREATE FUNCTION o ALTER FUNCTION para la función definida por el usuario.

Por ejemplo, puede codificar la opción TEST utilizando los siguientes parámetros:

```
TEST(ALL,*,PROMPT,TCPIP&ABC.EXAMPLE.COM%8001:*)
```

Para obtener más información, consulte [Sintaxis de la opción de tiempo de ejecución TEST](#) y [Ejemplo: Opciones de tiempo de ejecución de TEST](#).

Para obtener más información sobre la depuración interactiva desde varias interfaces, consulte los siguientes temas:

- [Depuración remota con un Eclipse IDE](#)
  - [Depuración de programas en modo de pantalla completa](#)
  - Utilícelo z/OS Debugger en modo por lotes completando los siguientes pasos. z/OS Debugger en el sistema de z/OS, donde se ejecuta la función definida por el usuario.
    - a) Compile la función definida por el usuario con la opción TEST si planea utilizar la opción de tiempo de ejecución TEST del Entorno de lenguaje para invocar z/OS Debugger. La opción TEST coloca información en el programa que utiliza z/OS Debugger durante una sesión de depuración.
    - b) Asigne un conjunto de datos de registro para recibir la salida de z/OS Debugger. Ponga una declaración DD para el conjunto de datos de registro en el procedimiento de inicio para el espacio de direcciones de procedimientos almacenados.
    - c) Introduzca los comandos de un conjunto de datos que desee ejecutar z/OS Debugger. Ponga una declaración DD para ese conjunto de datos en el procedimiento de inicio para el espacio de direcciones de procedimientos almacenados. Para definir el conjunto de datos de comandos z/OS Debugger, especifique el nombre del conjunto de datos de comandos o el nombre DD en la opción de tiempo de ejecución TEST.
- Por ejemplo, para especificar que z/OS Debugger utiliza los comandos que están en el conjunto de datos asociado con el nombre de DD TESTDD, incluya el siguiente parámetro en la opción TEST:

```
TEST(ALL,TESTDD,PROMPT,*)
```

El primer comando en el conjunto de datos de comandos debe ser:

```
SET LOG ON FILE ddname;
```

Este comando dirige la salida de su sesión de depuración al conjunto de datos de registro que definió en el paso anterior. Por ejemplo, si ha definido un conjunto de datos de registro con el nombre DD INSPLOG en el procedimiento de inicio del espacio de direcciones de los procedimientos almacenados, el primer comando debe ser el siguiente:

```
SET LOG ON FILE INSPLOG;
```

d) Invoque z/OS Debugger utilizando uno de los siguientes métodos:

- Especifique la opción de tiempo de ejecución TEST. El lugar más conveniente para hacerlo es en el parámetro RUN OPTIONS de la declaración CREATE o ALTER para el procedimiento almacenado.
- Poner llamadas CEETEST en el código fuente del procedimiento almacenado. Si utiliza este enfoque para un procedimiento almacenado existente, debe volver a compilarlo, volver a vincularlo y enlazarlo, y emitir los comandos STOP PROCEDURE y START PROCEDURE para volver a cargar el procedimiento almacenado.

Puede combinar la opción de tiempo de ejecución TEST con llamadas CEETEST. Por ejemplo, es posible que desee utilizar TEST para nombrar el conjunto de datos de comandos, pero utilizar llamadas CEETEST para controlar cuando z/OS Debugger toma el control.

Para obtener más información, consulte [Iniciar depurador de z/OS s en modo por lotes](#).

#### Referencia relacionada

[Instrucción CREATE FUNCTION \(resumen\) \( Db2 SQL\)](#)

[IBM Debug for z/OS](#)

## Probar una función definida por el usuario enviando los mensajes de depuración a SYSPRINT

Puede incluir extractos impresos simples en su código de función definido por el usuario que enruta a SYSPRINT. A continuación, utilice System Display and Search Facility (SDSF) para examinar el contenido de SYSPRINT mientras se ejecuta el espacio de direcciones de procedimiento almacenado establecido por WLM.

#### Acerca de esta tarea

Puede serializar E/S ejecutando el espacio de direcciones de procedimiento almacenado establecido por WLM con NUMTCB=1.

## Probar una función definida por el usuario mediante aplicaciones de controlador

Puede escribir una pequeña aplicación de controlador que llame a una función definida por el usuario como un subprograma y pase la lista de parámetros para la función definida por el usuario. A continuación, puede probar y depurar la función definida por el usuario como una aplicación normal de TSO ( Db2 ).

#### Acerca de esta tarea

A continuación, puede utilizar TSO TEST y otras herramientas de depuración de uso común.

## Probar una función definida por el usuario mediante sentencias SQL INSERT

Puede utilizar SQL para insertar información de depuración en una tabla de e Db2 . Esto permite que otras máquinas de la red (como estaciones de trabajo) accedan fácilmente a los datos de la tabla mediante el acceso DRDA.

#### Acerca de esta tarea

Db2 descarta la información de depuración si la aplicación ejecuta la sentencia ROLLBACK. Para evitar la pérdida de los datos de depuración, codifique la aplicación de llamada para que recupere los datos de diagnóstico antes de ejecutar la instrucción ROLLBACK.

# Depuración de procedimientos almacenados

---

A la hora de depurar procedimientos almacenados, es posible que deba utilizar distintas técnicas de las que usaría para programas de aplicación normales. Por ejemplo, algunas herramientas de depuración que se utilizan frecuentemente, como TSO TEST, no están disponibles en el entorno en el que se ejecutan los procedimientos almacenados.

## Procedimiento

Para depurar un procedimiento almacenado, realice una o más de las siguientes acciones:

- Tome una o más de las siguientes medidas generales, que son apropiadas en muchas situaciones con procedimientos almacenados:
  - Asegúrese de que todos los procedimientos almacenados estén escritos para manejar cualquier error SQL.
  - Depurar procedimientos almacenados como programas independientes en una estación de trabajo. Si dispone de herramientas de depuración en una estación de trabajo, considere la posibilidad de realizar la mayor parte de su desarrollo y pruebas en una estación de trabajo antes de instalar un procedimiento almacenado en z/OS. Esta técnica da lugar a muy poca actividad de depuración en z/OS.
  - Grabar mensajes de depuración de procedimientos almacenados en un archivo de disco o en un archivo de cola JES.
  - Almacenar información de depuración en una tabla. Esta técnica es especialmente útil para procedimientos almacenados remotos.
  - Utilice el comando DISPLAY para ver información sobre procedimientos almacenados concretos, incluidas estadísticas e información de hilos.
  - En el procedimiento almacenado que está depurando, emita comandos DISPLAY. Puede ver los resultados de DISPLAY en la salida de SDSF. Los resultados de DISPLAY pueden ayudarle a encontrar información sobre la tarea iniciada que está asociada con el espacio de direcciones para el entorno de aplicación WLM.
  - Si es necesario, utilice el comando DETENER PROCEDIMIENTO para detener las llamadas a uno o más procedimientos almacenados problemáticos. Puede reiniciarlos más tarde.
  - Si el espacio de direcciones de los procedimientos almacenados tiene asignado el conjunto de datos CEEDUMP, consulte la información de diagnóstico en la salida de CEEDUMP.
- Para procedimientos almacenados en COBOL, C y C++, utilice la herramienta de depuración para z/OS.
- Para los procedimientos almacenados COBOL, compile el procedimiento almacenado con la opción TEST(SYM) si desea que se incluya un volcado de variables locales formateado en la salida CEEDUMP.
- Para procedimientos SQL nativos, procedimientos SQL externos y procedimientos almacenados de Java, utilice el Unified Debugger.
- Para los procedimientos almacenados externos, considere tomar una o ambas de las siguientes acciones:
  - Utiliza una aplicación de controlador.
  - Cree o modifique la definición del procedimiento almacenado para incluir la opción PARAMETER STYLE SQL. Esta opción permite que el procedimiento almacenado comparta cualquier información de error con la aplicación que realiza la llamada. Asegúrese de que su procedimiento sigue las convenciones de vinculación para procedimientos almacenados.
- Si ha cambiado un procedimiento almacenado o un procedimiento JCL de inicio para un entorno de aplicación WLM, determine si necesita actualizar el entorno WLM. Debe actualizar el entorno WLM antes de que surtan efecto ciertos cambios en los procedimientos almacenados.

## Tareas relacionadas

[Gestión de condiciones SQL en un procedimiento SQL](#)

En un procedimiento SQL, puede especificar cómo gestiona el programa algunos errores y avisos de SQL.

[Visualización de información sobre procedimientos almacenados con mandatos de Db2 \(Db2 Administration Guide\)](#)

[Renovación de un entorno de aplicación de WLM para procedimientos almacenados \(Db2 Administration Guide\)](#)

[Implementación de procedimientos almacenados de Db2 \(Db2 Administration Guide\)](#)

#### **Referencia relacionada**

[Convenciones de enlace para procedimientos de almacenamiento externo](#)

La convención de enlace para un procedimiento almacenado puede ser GENERAL, GENERAL WITH NULLS o SQL. Estas convenciones de enlace se aplican únicamente a procedimientos de almacenamiento externos.

[-START PROCEDURE comando \(Db2\) \( Db2 Comandos\)](#)

[-STOP PROCEDURE comando \(Db2\) \( Db2 Comandos\)](#)

#### **Información relacionada**

[Db2 para procedimientos almacenados de SQL Server \( z/OS Stored Procedures: Through the CALL and Beyond \( IBM Redbooks \)](#)

## **Depuración de procedimientos almacenados mediante el uso del Unified Debugger**

Puede utilizar el depurador de procedimientos ( Unified Debugger ) para depurar de forma remota procedimientos SQL nativos, procedimientos SQL externos y procedimientos almacenados en Java que se ejecutan en servidores de Db2 for z/OS . Unified Debugger también admite la depuración de llamadas a procedimientos almacenados anidados.

#### **Acerca de esta tarea**

Con el depurador ( Unified Debugger), puede observar la ejecución del código de procedimiento, establecer puntos de interrupción para las líneas y ver o modificar los valores de las variables.

#### **Procedimiento**

Para depurar procedimientos almacenados utilizando el Unified Debugger:

1. Configure el Unified Debugger siguiendo estos pasos:

- a) Asegúrese de que el trabajo DSNTIJRT ha creado correctamente los procedimientos almacenados que proporcionan soporte de servidor para el Unified Debugger. Este trabajo se ejecuta durante el proceso de instalación y migración. Los procedimientos almacenados que este trabajo crea deben ejecutarse en entornos WLM.

**Recomendación:** Inicialmente, defina y utilice el entorno WLM principal de Db2 DSNWLM\_GENERAL para ejecutar el procedimiento almacenado SYSPROC.DBG\_RUNSESSIONMANAGER y el entorno WLM principal DSNWLM\_DEBUGGER para ejecutar los demás procedimientos almacenados para Unified Debugger.

- b) Defina las características del modo de depuración para el procedimiento almacenado que desea depurar completando una de las siguientes acciones:

- Para los procedimientos SQL nativos, defina los procedimientos con la opción ALLOW DEBUG MODE y la opción WLM ENVIRONMENT FOR DEBUG MODE. Si el procedimiento ya existe, puede utilizar la instrucción ALTER PROCEDURE para especificar estas opciones.
- Para un procedimiento SQL externo, utilice DSNTPSMP para crear el procedimiento SQL con la opción BUILD\_DEBUG.
- Si implementa el procedimiento con Db2 Developer Extension, especifique Habilitar depuración en las opciones de implementación.
- Para los procedimientos almacenados de Java, defina los procedimientos con la opción ALLOW DEBUG MODE, seleccione un entorno WLM adecuado para la depuración de Java y compile el código Java con la opción -G.

- c) Otorgue el privilegio DEBUGSESSION al usuario que ejecuta el cliente de depuración.
2. Incluya puntos de interrupción en sus rutinas o archivos ejecutables.
3. Siga las instrucciones para depurar procedimientos almacenados en la documentación de Db2 Developer Extension. Para obtener más información, consulte [Depuración de procedimientos almacenados SQL nativos y externos](#).

### Conceptos relacionados

Procedimientos almacenados y funciones definidas por el usuario de Java (Db2 Application Programming for Java)

### Tareas relacionadas

[Creación de un procedimiento SQL externo utilizando DSNTPSMP](#)

El procesador de procedimiento SQL, DSNTPSMP, es uno de los distintos métodos que puede utilizar para crear y preparar un procedimiento SQL externo. DSNTPSMP es un procedimiento almacenado REXX que puede invocar desde el programa de aplicación.

[Desarrollo de rutinas de base de datos \(IBM Data Studio, IBM Optim Database Administrator, IBM infoSphere Data Architect, IBM Optim Development Studio\)](#)

[Instalación del Unified Debugger gestor de sesiones en un z/OS sistema \( Db2 Instalación y migración\)](#)

### Referencia relacionada

Programas de ejemplo para ayudar a preparar y ejecutar procedimientos SQL externos

Db2 proporciona trabajos de ejemplo para ayudarle a preparar y ejecutar procedimientos SQL externos.

Todos los ejemplos están en el conjunto de datos de DSN1210.SDSNSAMP. Antes de poder ejecutar los ejemplos, debe personalizarlos para la instalación.

[Declaración ALTER PROCEDURE \(SQL - procedimiento nativo\) \( Db2 SQL\)](#)

[Instrucción CREATE PROCEDURE \(SQL - procedimiento nativo\) \( Db2 SQL\)](#)

### Información relacionada

[Db2 para procedimientos almacenados de SQL Server \( z/OS Stored Procedures: Through the CALL and Beyond \( IBM Redbooks \)](#)

## Depuración de procedimientos almacenados con z/OS Debugger

Puede utilizar z/OS Debugger para probar rutinas de z/OS , incluidos procedimientos almacenados, que estén escritos en cualquiera de los lenguajes compilados que admite. Puede probar estos procedimientos almacenados de forma interactiva o en modo por lotes.

### Procedimiento

Para depurar su rutina con z/OS Debugger, utilice uno de los siguientes métodos:

- Utilice z/OS Debugger de forma interactiva siguiendo los pasos que se indican a continuación.
  - a) Compile la rutina con la opción TEST. La opción TEST coloca información en el programa que utiliza z/OS Debugger durante una sesión de depuración.
  - b) Invocar a z/OS Debugger.

Una forma de hacerlo es especificar la opción TEST del entorno de ejecución de Language Environment. La opción TEST controla cuándo y cómo se invoca z/OS Debugger . El lugar más conveniente para especificar las opciones de tiempo de ejecución es en el parámetro RUN OPTIONS de la instrucción CREATE PROCUDURE o ALTER PROCEDURE para el procedimiento almacenado. Para obtener más información, consulte [Preparación de un programa de procedimientos almacenados \( Db2 \)](#).

Por ejemplo, puede codificar la opción TEST utilizando los siguientes parámetros:

```
TEST(ALL, *, PROMPT, TCPIP&ABC.EXAMPLE.COM%8001:::)
```

Para obtener más información, consulte [Sintaxis de la opción de tiempo de ejecución TEST](#) y [Ejemplo: Opciones de tiempo de ejecución de TEST](#).

Para obtener más información sobre la depuración interactiva desde varias interfaces, consulte los siguientes temas:

- [Depuración remota con un Eclipse IDE](#)
- [Depuración de programas en modo de pantalla completa](#)
- Utilícelo z/OS Debugger en modo por lotes completando los siguientes pasos. z/OS Debugger en el sistema de z/OS , donde se ejecuta el procedimiento almacenado.
  - a) Compile el procedimiento almacenado con la opción TEST si planea usar la opción de tiempo de ejecución TEST del Entorno de lenguaje para invocar z/OS Debugger. La opción TEST coloca información en el programa que utiliza z/OS Debugger durante una sesión de depuración.
  - b) Asigne un conjunto de datos de registro para recibir la salida de z/OS Debugger. Ponga una declaración DD para el conjunto de datos de registro en el procedimiento de inicio para el espacio de direcciones de procedimientos almacenados.
  - c) Introduzca los comandos de un conjunto de datos que desee ejecutar z/OS Debugger. Ponga una declaración DD para ese conjunto de datos en el procedimiento de inicio para el espacio de direcciones de procedimientos almacenados. Para definir el conjunto de datos de comandos z/OS Debugger, especifique el nombre del conjunto de datos de comandos o el nombre DD en la opción de tiempo de ejecución TEST.  
Por ejemplo, para especificar que z/OS Debugger utiliza los comandos que están en el conjunto de datos asociado con el nombre de DD TESTDD, incluya el siguiente parámetro en la opción TEST:

```
TEST(ALL,TESTDD,PROMPT,*)
```

El primer comando en el conjunto de datos de comandos debe ser:

```
SET LOG ON FILE ddname;
```

Este comando dirige la salida de su sesión de depuración al conjunto de datos de registro que definió en el paso anterior. Por ejemplo, si ha definido un conjunto de datos de registro con el nombre DD INSPLOG en el procedimiento de inicio del espacio de direcciones de los procedimientos almacenados, el primer comando debe ser el siguiente:

```
SET LOG ON FILE INSPLOG;
```

d) Invoque z/OS Debugger utilizando uno de los siguientes métodos:

- Especifique la opción de tiempo de ejecución TEST. El lugar más conveniente para hacerlo es en el parámetro RUN OPTIONS de la declaración CREATE o ALTER para el procedimiento almacenado.
- Poner llamadas CEETEST en el código fuente del procedimiento almacenado. Si utiliza este enfoque para un procedimiento almacenado existente, debe volver a compilarlo, volver a vincularlo y enlazarlo, y emitir los comandos STOP PROCEDURE y START PROCEDURE para volver a cargar el procedimiento almacenado.

Puede combinar la opción de tiempo de ejecución TEST con llamadas CEETEST. Por ejemplo, es posible que desee utilizar TEST para nombrar el conjunto de datos de comandos, pero utilizar llamadas CEETEST para controlar cuando z/OS Debugger toma el control.

Para obtener más información, consulte [Iniciar depurador de z/OS s en modo por lotes](#).

#### Referencia relacionada

[IBM Debug for z/OS](#)

## Grabar mensajes de depuración de procedimientos almacenados en un archivo

Puede depurar procedimientos almacenados externos y procedimientos SQL externos registrando mensajes de depuración en un archivo de disco o en un archivo de cola JES. No puede utilizar esta técnica de depuración para procedimientos SQL nativos o procedimientos almacenados en Java.

## Procedimiento

Para registrar mensajes de depuración de procedimientos almacenados en un archivo:

1. Especifique la opción de tiempo de ejecución MSGFILE del entorno de lenguaje (LE) para el procedimiento almacenado. Esta opción identifica dónde debe escribir LE los mensajes de depuración. Para especificar esta opción, incluya la cláusula RUN OPTIONS en la instrucción CREATE PROCEDURE o en una instrucción ALTER PROCEDURE.

Especifique los siguientes parámetros MSGFILE:

- Utilice el primer parámetro MSGFILE para especificar la instrucción JCL DD que identifica el conjunto de datos para los mensajes de depuración. Puede dirigir los mensajes de depuración a un archivo de disco o a un archivo de cola JES. Para evitar que varios procedimientos compartan un conjunto de datos, asegúrese de especificar una instrucción DD única.
  - Utilice la opción ENQ para serializar E/S en el archivo de mensajes. Esta acción es necesaria, porque pueden haber varios TCB activos en el espacio de direcciones del procedimiento almacenado. Alternativamente, si depura sus aplicaciones con poca frecuencia o en un sistema de prueba e Db2 , puede serializar E/S ejecutando temporalmente el espacio de direcciones de procedimientos almacenados con NUMTCB=1 en el procedimiento de inicio del espacio de direcciones de procedimientos almacenados.
2. Para cada instancia de MSGFILE que especifique, añada una instrucción DD al procedimiento JCL que se utiliza para iniciar el espacio de direcciones de los procedimientos almacenados.

### Referencia relacionada

[Declaración ALTER PROCEDURE \(procedimiento externo\) \( Db2 SQL\)](#)

[Declaración ALTER PROCEDURE \(SQL - procedimiento externo\) \(obsoleta\) \( Db2 SQL\)](#)

[Instrucción CREATE PROCEDURE \(procedimiento externo\) \( Db2 SQL\)](#)

[Instrucción CREATE PROCEDURE \(SQL - procedimiento externo\) \(obsoleta\) \( Db2 SQL\)](#)

[Declaración GRANT \(privilegios del sistema\) \( Db2 SQL\)](#)

[Utilización de MSGFILE de Language Environment \(z/OS Language Environment Programming Guide\)](#)

## Aplicaciones de controlador para procedimientos de depuración

Puede escribir una pequeña aplicación de controlador que llame al procedimiento almacenado como un subprograma y pase la lista de parámetros que admite el procedimiento almacenado. A continuación, puede probar y depurar el procedimiento almacenado como una aplicación normal de TSO ( Db2 ).

Con este método, puede utilizar TSO TEST y otras herramientas de depuración de uso común.

**Restricción :** No puede utilizar esta técnica para procedimientos SQL

## Db2 tablas que contienen información de depuración

Puede utilizar instrucciones SQL para insertar información de depuración en una tabla de Db2 . La inserción de esta información en una tabla permite a otras máquinas de la red (como una estación de trabajo) acceder fácilmente a los datos de la tabla mediante el acceso DRDA.

Db2 descarta la información de depuración si la aplicación ejecuta la sentencia ROLLBACK. Para evitar la pérdida de los datos de depuración, codifique la aplicación de llamada para que recupere los datos de diagnóstico antes de ejecutar la instrucción ROLLBACK.

## Depuración de un programa de aplicación

Muchos sitios tienen directrices sobre qué debe hacer si un programa termina de forma anómala.

## Acerca de esta tarea

Para obtener información sobre las instalaciones de prueba del compilador o ensamblador, consulte las publicaciones del compilador o CODE/370. Las publicaciones del compilador incluyen información sobre el depurador adecuado para el lenguaje que está utilizando.

También puede utilizar la prueba de diálogo de ISPF para depurar su programa. Puede ejecutar toda o parte de su solicitud, examinar los resultados, realizar cambios y volver a ejecutarla.

### Referencia relacionada

[Prueba de diálogo \(opción 7\) \(z/OS ISPF User's Guide Vol II\)](#)

## Localización del problema en una aplicación

Si su programa no se ejecuta correctamente, debe aislar el problema. Debería comprobar varios elementos.

### Acerca de esta tarea

Esos artículos son:

- Resultado del precompilador, que consiste en errores y advertencias. Asegúrese de haber resuelto todos los errores y advertencias.
- Salida del compilador o ensamblador. Asegúrese de haber resuelto todos los mensajes de error.
- Salida del editor de enlaces.
  - ¿Ha resuelto todas las referencias externas?
  - ¿Ha incluido todos los módulos necesarios en el orden correcto?
  - ¿Incluiste el módulo de interfaz de idioma correcto? El módulo de interfaz de idioma correcto es:
    - DSNELI o DSNULI para TSO
    - DFSLI000 para IMS
    - DSNCLI o DSNULI para CICS
    - DSNALI o DSNULI para la función de adjuntar llamadas
    - DSNRLI o DSNULI para el servicio de depósito de los Servicios de Recuperación de Recursos
  - ¿Ha especificado el punto de entrada correcto a su programa?
- Resultado del proceso de enlace.
  - ¿Ha resuelto todos los mensajes de error?
  - ¿Ha especificado un nombre de plan? De lo contrario, el proceso de enlace asume que desea procesar el DBRM con fines de diagnóstico, pero que no desea producir un plan de aplicación.
  - ¿Ha especificado todos los paquetes asociados a los programas que componen la aplicación y sus nombres de conjuntos de datos particionados (PDS) en un único plan de aplicación?
- Su JCL.

### IMS

- Si está utilizando IMS, ¿ha incluido la declaración de opción DL/I en el formato correcto?
- ¿Ha incluido el parámetro de tamaño de región en la instrucción EXEC? ¿Especifica un tamaño de región lo suficientemente grande para el almacenamiento requerido para la interfaz de Db2, el TSO, IMS, o CICS sistema y su programa?
- ¿Ha incluido los nombres de todos los conjuntos de datos (Db2 y non-Db2) que requiere el programa?
- Tu programa.

También puede utilizar volcados para ayudar a localizar problemas en su programa. Por ejemplo, una de las situaciones de error más comunes se produce cuando el programa se está ejecutando y aparece

un mensaje indicando que se ha interrumpido. En esta situación, su procedimiento de prueba podría ser capturar un volcado TSO. Para ello, debe asignar un conjunto de datos de volcado SYSUDUMP o SYSABEND antes de llamar a Db2. Cuando pulsas la tecla INTRO (después del mensaje de error y del mensaje LISTO), el sistema solicita un volcado. A continuación, debe utilizar el comando FREE para desasignar el conjunto de datos de volcado.

## Mensajes de error y advertencia del precompilador

En algunas circunstancias, las declaraciones que genera el precompilador de Db2 podrían producir mensajes de error de compilación o ensamblado. Necesita saber por qué se producen los mensajes cuando compila sentencias fuente producidas por Db2.

## Salida SYSTERM del precompilador

La salida SYSTERM proporciona un breve resumen de los resultados del precompilador, todos los mensajes de error que generó el precompilador y la instrucción que contiene el error, cuando es posible.

El precompilador Db2 proporciona una salida SYSTERM cuando se asigna el nombre de DD SYSTERM. Si utiliza los paneles de preparación de programas para preparar y ejecutar su programa, DB2I asigna SYSTERM según la opción TERM que especifique.

Puede utilizar el número de línea que se proporciona en cada mensaje de error en la salida de SYSTERM para localizar la instrucción fuente defectuosa.

Figura 74 en la página 1042 muestra el formato de salida de SYSTERM.

```
DB2 SQL PRECOMPILER MESSAGES
DSNH104I E DSNHPARS LINE 32 COL 26 ILLEGAL SYMBOL "X" VALID SYMBOLS ARE: , FROM1
SELECT VALUE INTO HIPPO X;2

DB2 SQL PRECOMPILER STATISTICS
SOURCE STATISTICS3
 SOURCE LINES READ: 36
 NUMBER OF SYMBOLS: 15
 SYMBOL TABLE BYTES EXCLUDING ATTRIBUTES: 1848
 THERE WERE 1 MESSAGES FOR THIS PROGRAM.4
 THERE WERE 0 MESSAGES SUPPRESSED BY THE FLAG OPTION.5
 111664 BYTES OF STORAGE WERE USED BY THE PRECOMPILER.6
 RETURN CODE IS 87
```

*Figura 74. Db2 salida del precompilador SYSTERM*

### Notas:

1. Mensaje de error.
2. Fuente de la instrucción SQL.
3. Resúmenes de las estadísticas de origen.
4. Declaración resumida del número de errores detectados.
5. Declaración resumida que indica el número de errores que se detectaron pero no se imprimieron. Esta situación puede ocurrir si especifica una opción FLAG distinta de I.
6. Declaración de requisitos de almacenamiento que indica cuántos bytes de almacenamiento de trabajo utilizó realmente el precompilador de Db2 para procesar sus declaraciones de origen. Ese valor le ayuda a determinar los requisitos de asignación de almacenamiento para su programa.
7. Código de retorno: 0 = éxito, 4 = advertencia, 8 = error, 12 = error grave y 16 = error irrecuperable.

## Salida SYSPRINT del precompilador

La salida SYSPRINT del precompilador de Db2 muestra los resultados de la operación de precompilación. Este resultado también puede incluir una lista de las opciones que se utilizaron, un listado del código fuente y un listado de referencias cruzadas de variables de host.

Cuando utiliza los paneles de preparación de programas para preparar y ejecutar su programa, Db2 asigna SYSPRINT de acuerdo con la opción TERM que especifique (en la línea 12 del panel PROGRAM PREPARATION: COMPILE, PRELINK, LINK, AND RUN). Como alternativa, cuando utilice el procedimiento de comando DSNH (CLIST), puede especificar PRINT(TERM) para obtener la salida SYSPRINT en su terminal, o puede especificar PRINT(*calificador*) para colocar la salida SYSPRINT en un conjunto de datos llamado *authorizationID.calificador.PCLIST*. Db2 , en el supuesto de que no especifique IMPRIMIR como DEJAR, NINGUNO o TÉRMINO, emite un mensaje cuando finaliza el precompilador, indicándole dónde encontrar sus listados de precompilador. Esto le ayuda a localizar sus diagnósticos de forma rápida y sencilla.

La salida SYSPRINT puede proporcionar información sobre su módulo fuente precompilado si especifica las opciones SOURCE y XREF al iniciar el precompilador de Db2 .

El formato de salida de SYSPRINT es el siguiente:

- Una lista de las opciones del precompilador de Db2 que están en vigor durante la precompilación (si no especificó NOOPTIONS).
- Una lista de sus declaraciones de origen (solo si especificó la opción ORIGEN). Se muestra un ejemplo en [Figura 75 en la página 1044](#).
- Una lista de los nombres simbólicos utilizados en las sentencias SQL (esta lista aparece solo si especifica la opción XREF). En [Figura 76 en la página 1044](#) se muestra un ejemplo.
- Un resumen de los errores detectados por el precompilador de Db2 y una lista de los mensajes de error generados por el precompilador. Se muestra un ejemplo en

El siguiente código muestra una lista de ejemplo de opciones del precompilador de Db2 , tal como se muestra en la salida de SYSPRINT.

```
DB2 SQL PRECOMPILER VERSION 11 REL. 1.0

OPTIONS SPECIFIED: HOST(PLI),SOURCE,XREF,STDSQL(NO),TWOPASS
DSNHDECP LOADED FROM - (USER99.RELM.TESTLIB(DSNHDECP))
OPTIONS USED - SPECIFIED OR DEFAULTED
APOST
APOSTSQL
ATTACH(TSO)
CCSID(37)
CONNECT(2)
DEC(15)
FLAG(I)
FLOAT(S390)
HOST(PLI)
LINECOUNT(60)
MARGINS(2,72)
NEWFUN(V11)
OPTIONS
PERIOD
SOURCE
SQL(DB2)
STDSQL(NO)
TWOPASS
XREF
```

#### Notas:

1. Esta sección enumera las opciones que se especifican en el momento de la precompilación. Esta lista no aparece si una de las opciones del precompilador es NOOPTIONS.
2. Esta sección enumera las opciones que están en vigor, incluyendo los valores predeterminados, los valores forzados y las opciones que usted especificó. El precompilador de Db2 anula o ignora cualquier opción que especifique que sea inapropiada para el lenguaje del host.

La siguiente figura muestra un ejemplo de lista de sentencias fuente tal y como se muestra en la salida de SYSPRINT.

```

DB2 SQL PRECOMPILER TMN5P40:PROCEDURE OPTIONS (MAIN): PAGE 2
1 TMN5P40:PROCEDURE OPTIONS(MAIN) ; 00000100
2 /******00000200
3 * program description and prologue 00000300
:
1324 /*****00132400
1325 /* GET INFORMATION ABOUT THE PROJECT FROM THE */ 00132500
1326 /* PROJECT TABLE. */ 00132600
1327 /*****00132700
1328 EXEC SQL SELECT ACTNO, PREQPROJ, PREQACT 00132800
1329 INTO PROJ_DATA 00132900
1330 FROM TPREREQ 00133000
1331 WHERE PROJNO = :PROJ_NO; 00133100
1332 00133200
1333 /*****00133300
1334 /* PROJECT IS FINISHED. DELETE IT. */ 00133400
1335 /*****00133500
1336 00133600
1337 EXEC SQL DELETE FROM PROJ 00133700
1338 WHERE PROJNO = :PROJ_NO; 00133800
:
1523 END; 00152300

```

Figura 75. Db2 salida del precompilador SYSPRINT: sección de sentencias fuente

#### Notas:

- La columna izquierda de números de secuencia, que genera el precompilador de Db2 , se utiliza con la lista de referencias cruzadas de símbolos, los mensajes de error del precompilador y los mensajes de error de BIND.
- La columna de la derecha muestra los números de secuencia que proceden de los números de secuencia que se suministran con sus extractos bancarios.

La siguiente figura muestra un ejemplo de lista de nombres simbólicos tal y como se muestra en la salida de SYSPRINT.

DB2 SQL PRECOMPILER	SYMBOL CROSS-REFERENCE LISTING	PAGE 29
DATA NAMES	DEFN	REFERENCE
"ACTNO"	****	FIELD 1328
"PREQACT"	****	FIELD 1328
"PREQPROJ"	****	FIELD 1328
"PROJNO"	****	FIELD 1331 1338
...		
PROJ_DATA	495	CHARACTER(35) 1329
PROJ_NO	496	CHARACTER(3) 1331 1338
"TPREREQ"	****	TABLE 1330 1337

Figura 76. Db2 salida del precompilador SYSPRINT: Sección de referencias cruzadas de símbolos

#### Notas:

##### NOMBRES DE LOS DATOS

Identifica los nombres simbólicos que se utilizan en las declaraciones de origen. Los nombres entre comillas dobles ("") o apóstrofes ('') son nombres de entidades SQL como tablas, columnas e ID de autorización. Otros nombres son variables de host.

##### DEFN

Es el número de la línea que genera el precompilador para definir el nombre. \*\*\*\* significa que el objeto no estaba definido o que el precompilador no reconoció las declaraciones.

## REFERENCIA.

Contiene dos tipos de información: el nombre simbólico, que define el programa fuente, y las líneas que hacen referencia al nombre simbólico. Si el nombre simbólico se refiere a una variable de host válida, la lista también identifica el tipo de datos o la palabra ESTRUCTURA.

El siguiente código muestra un ejemplo de informe resumido de errores tal y como se muestra en la salida de SYSPRINT.

```
DB2 SQL PRECOMPILER STATISTICS

SOURCE STATISTICS
 SOURCE LINES READ: 15231
 NUMBER OF SYMBOLS: 1282
 SYMBOL TABLE BYTES EXCLUDING ATTRIBUTES: 64323

THERE WERE 1 MESSAGES FOR THIS PROGRAM.4
THERE WERE 0 MESSAGES SUPPRESSED.5
65536 BYTES OF STORAGE WERE USED BY THE PRECOMPILER.6
RETURN CODE IS 8.7
DSNH104I E LINE 590 COL 64 ILLEGAL SYMBOL: 'X'; VALID SYMBOLS ARE: ,FROM8
```

### Notas:

1. Declaración resumida que indica el número de líneas de origen.
2. Declaración resumida que indica el número de nombres simbólicos en la tabla de símbolos (nombres SQL y nombres de host).
3. Declaración de requisitos de almacenamiento que indica el número de bytes para la tabla de símbolos.
4. Declaración resumida que indica el número de mensajes que se imprimen.
5. Declaración resumida que indica el número de errores detectados pero no impresos. Puede que reciba este mensaje si especifica la opción FLAG.
6. Declaración de requisitos de almacenamiento que indica el número de bytes de almacenamiento de trabajo que utiliza realmente el precompilador de Db2 para procesar sus instrucciones de origen.
7. Código de retorno 0 = éxito, 4 = advertencia, 8 = error, 12 = error grave y 16 = error irrecuperable.
8. Mensajes de error (este ejemplo detecta solo un error).

## Técnicas para depurar programas en TSO

Documentar los errores que se identifican durante la prueba de una aplicación TSO le ayuda a investigar y corregir problemas en el programa.

La siguiente información puede ser útil:

- El nombre del plan de aplicación del programa
- Los datos de entrada que se están procesando
- La sentencia SQL fallida y su función
- El contenido del SQLCA (área de comunicación SQL) y, si su programa acepta sentencias SQL dinámicas, el SQLDA (área de descriptor SQL)
- La fecha y la hora del día
- El código de error y cualquier mensaje de error

Cuando su programa encuentra un error que no resulta en un abend, puede pasar toda la información de error requerida a una rutina de error estándar. Los programas en línea también pueden enviar un mensaje de error al terminal.

## El comando TSO TEST

El comando TSO TEST es especialmente útil para depurar programas ensambladores.

El siguiente ejemplo es un procedimiento de comando (CLIST) que ejecuta una aplicación de programa de terminal ( Db2 ) llamada MYPROG bajo TSO TEST, y establece una dirección de parada en la entrada al programa. El nombre del subsistema de Db2 en este ejemplo es DB4.

```
PROC 0
TEST 'prefix.SDSNLOAD(DSN)' CP
DSN SYSTEM(DB4)
AT MYPROG.MYPROG.+0 DEFER
GO
RUN PROGRAM(MYPROG) LIBRARY('L186331.RUNLIB.LOAD(MYPROG)')
```

### Referencia relacionada

[Mandato TEST \(TSO/E Command Reference\)](#)

## Técnicas para depurar programas en IMS

Documentar los errores que se identifican durante la prueba de una IMS aplicación le ayuda a investigar y corregir problemas en el programa.

La siguiente información puede ser útil:

- El nombre del plan de aplicación para el programa
- El mensaje de entrada que se está procesando
- El nombre del terminal lógico de origen
- La declaración errónea y su función
- El contenido del SQLCA (área de comunicación SQL) y, si su programa acepta sentencias SQL dinámicas, el SQLDA (área de descriptor SQL)
- La fecha y la hora del día
- El nombre del programa en PSB
- El código de transacción que el programa estaba procesando
- La función de llamada (es decir, el nombre de una función DL/I)
- El contenido del PCB al que se refiere la llamada del programa
- Si se estaba ejecutando una llamada a la base de datos DL/I, los SSA, si los hubiera, que utilizó la llamada
- El código de finalización del error, el código de motivo del error y cualquier mensaje de error de volcado

Cuando su programa encuentra un error, puede pasar toda la información de error requerida a una rutina de error estándar. Los programas en línea también pueden enviar un mensaje de error al terminal lógico de origen.

Un programa interactivo también puede enviar un mensaje al operador de operador de terminal maestro (MTO) con información sobre la finalización del programa. Para ello, el programa coloca el nombre lógico del terminal maestro en un PCB expreso y emite una o más llamadas ISRT.

Algunas organizaciones ejecutan un BMP al final del día para enumerar todos los errores que se han producido durante el día. Si su organización hace esto, puede enviar un mensaje utilizando una PCB expresa que tenga su destino establecido para ese BMP.

**Simulador de terminal por lotes:** el Simulador de terminal por lotes (BTS) le permite probar programas de aplicación IMS. BTS rastrea las llamadas DL/I y las sentencias SQL del programa de aplicación, y simula las funciones de comunicación de datos. Puede hacer que un terminal TSO aparezca como un IMS terminal para el operador de la terminal, lo que permite al usuario interactuar con la aplicación como si fuera una aplicación en línea. El usuario puede utilizar cualquier programa de aplicación que esté bajo su control para acceder a cualquier base de datos (ya sea DL/I o Db2) que esté bajo su control. El acceso a las bases de datos de Db2 requiere que BTS funcione en modo BMP por lotes o BMP TSO.

## Técnicas para depurar programas en CICS

Documentar los errores que se identifican durante la prueba de una CICS aplicación le ayuda a investigar y corregir problemas en el programa.

La siguiente información puede ser útil:

- El nombre del plan de aplicación del programa
- Los datos de entrada que se están procesando
- El ID del terminal lógico de origen
- La sentencia SQL fallida y su función
- El contenido del SQLCA (área de comunicación SQL) y, si su programa acepta sentencias SQL dinámicas, el SQLDA (área de descriptor SQL)
- La fecha y la hora del día
- Datos que son peculiares de CICS que debe registrar
- Código Abend y mensajes de error de volcado
- Volcado de transacciones, si se produce

Al utilizar CICS las instalaciones, puede obtener un registro de errores impreso; también puede imprimir los contenidos de SQLCA y SQLDA.

## Ayudas de depuración para CICS

CICS proporciona las siguientes ayudas para la comprobación, supervisión y depuración de programas de aplicación:

- **Instalación de diagnóstico de ejecución (nivel de comando) (EDF).** EDF muestra CICS comandos para todas las versiones de CICS.
- **Recuperación de la tarde.** Puede utilizar el comando HANDLE ABEND para tratar las condiciones de abend. Puede utilizar el comando ABEND para provocar un error en una tarea.
- **Instalación de seguimiento.** Una tabla de seguimiento puede contener entradas que muestren la ejecución de varios CICS comandos, sentencias SQL y entradas generadas por programas de aplicación; estas entradas pueden escribirse en el almacenamiento principal y, opcionalmente, en un dispositivo de almacenamiento auxiliar.
- **Instalación de vertido.** Puede especificar áreas del almacenamiento principal para volcarlas en un conjunto de datos secuenciales, ya sea en cinta o en disco, para su posterior formateo e impresión sin conexión con un CICS programa de utilidad.
- **Diarios.** Con fines estadísticos o de supervisión, las instalaciones pueden crear entradas en conjuntos de datos especiales llamados diarios. El registro del sistema es un diario.
- **Recuperación.** Cuando se produce un error, CICS devuelve ciertos recursos a su estado original para que el operador pueda volver a enviar fácilmente una transacción para reiniciarla. Puede utilizar el comando SYNCPOINT para subdividir un programa de modo que solo tenga que volver a enviar la parte incompleta de una transacción.

## CICS instalación de diagnóstico de ejecución

El CICS función de diagnóstico de ejecución (EDF) rastrea las sentencias SQL en un modo de depuración interactivo, lo que permite a los programadores de aplicaciones probar y depurar programas en línea sin cambiar el programa o el procedimiento de preparación del programa.

EDF intercepta el programa de aplicación en ejecución en varios puntos y muestra información útil sobre el tipo de instrucción, las variables de entrada y salida, y cualquier condición de error después de que se ejecute la instrucción. También muestra las pantallas que envía el programa de aplicación, para que pueda conversar con el programa de aplicación durante la prueba tal como lo haría un usuario en un sistema de producción.

EDF muestra información esencial antes y después de que se ejecute una instrucción SQL, mientras la tarea está en modo EDF. Esto puede ser de gran ayuda para depurar CICS programas de transacción que contienen instrucciones SQL. La información SQL que muestra EDF es útil para depurar programas y para el análisis de errores después de un error o advertencia SQL. El uso de esta función reduce la cantidad de trabajo que necesita hacer para escribir controladores de errores especiales.

## EDF antes de la ejecución

La siguiente figura muestra un ejemplo de una pantalla EDF antes de ejecutar una instrucción SQL. Los nombres de los campos de información clave de este panel están en **negrita**.

```
TRANSACTION: XC05 PROGRAM: TESTC05 TASK NUMBER: 0000668 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER DSNCSQL
EXEC SQL INSERT
DBRM=TESTC05, STMT=00368, SECT=00004
IVAR 001: TYPE=CHAR, LEN=00007, IND=000 AT X'03C92810'
 DATA=X'F0F0F9F4F3F4F2'
IVAR 002: TYPE=CHAR, LEN=00007, IND=000 AT X'03C92817'
 DATA=X'F0F1F3F3F7F5F1'
IVAR 003: TYPE=CHAR, LEN=00004, IND=000 AT X'03C9281E'
 DATA=X'E7C3F0F5'
IVAR 004: TYPE=CHAR, LEN=00040, IND=000 AT X'03C92822'
 DATA=X'E3C5E2E3C3F0F540E2C9D4D7D3C540C4C2F240C9D5E2C5D9E3404040'...
IVAR 005: TYPE=SMALLINT, LEN=00002, IND=000 AT X'03C9284A'
 DATA=X'0001'

OFFSET:X'001ECE' LINE:UNKNOWN EIBFN=X'1002'

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : UNDEFINED PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK
```

Figura 77. Pantalla EDF antes de una instrucción SQL e Db2

La información SQL de la base de datos (Db2) en esta pantalla es la siguiente:

- *Tipo de instrucción EXEC SQL*

Este es el tipo de instrucción SQL que se debe ejecutar. La instrucción SQL puede ser cualquier instrucción SQL válida.

- *DBRM=nombre dbrm*

El nombre del módulo de solicitud de base de datos (DBRM) que se está procesando actualmente. El DBRM, creado por el precompilador de la base de datos Oracle (Db2), contiene información sobre una instrucción SQL.

- *STMT=número de extracto*

Este es el número de la instrucción generada por el precompilador de Db2. Las listas de origen y mensaje de error del precompilador utilizan este número de instrucción, y usted puede utilizar el número de instrucción para determinar qué instrucción se está procesando. Este número es un contador de líneas de origen que incluye declaraciones de lenguaje de host. Un número de extracto superior a 32.767 se muestra como 0.

- *SECT=número de sección*

El número de sección del plan que utiliza la instrucción SQL.

## Declaraciones SQL que contienen variables de host de entrada

La sección IVAR (variables de host de entrada) y sus campos correspondientes aparecen solo cuando la instrucción de ejecución contiene variables de host de entrada.

La sección de variables de host incluye las variables de los predicados, los valores utilizados para insertar o actualizar y el texto de las sentencias SQL dinámicas que se están preparando. La dirección de la variable de entrada es AT X'nnnnnnnnn' !

Información adicional de la variable de host:

- *TYPE=tipo de datos*

Especifica el tipo de datos para esta variable de host. Los tipos de datos básicos incluyen cadena de caracteres, cadena gráfica, entero binario, punto flotante, decimal, fecha, hora y marca de tiempo.

- *LEN = longitud*

Especifica la longitud de la variable de host.

- *IND=número de estado de la variable indicadora*

Especifica la variable indicadora que está asociada con esta variable host en particular. Un valor de cero indica que no existe ninguna variable indicadora. Si el valor de la columna seleccionada es nulo, Db2 pone un valor negativo en la variable indicadora para esta variable host.

- *DATA=datos de la variable de host*

Especifica los datos, mostrados en formato hexadecimal, que están asociados con esta variable de host. Si los datos exceden lo que se puede mostrar en una sola línea, aparecen tres puntos (...) en el extremo derecho para indicar que hay más datos.

## EDF después de la ejecución

La siguiente figura muestra un ejemplo de la primera pantalla de EDF que se muestra después de ejecutar una instrucción SQL. Los nombres de los campos de información clave de este panel están en **negrita**.

```
TRANSACTION: XC05 PROGRAM: TESTC05 TASK NUMBER: 0000698 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER DSNCSQL
EXEC SQL FETCH P.AUTH=SYSADM , S.AUTH=
PLAN=TESTC05, DBRM=TESTC05, STMT=00346, SECT=00001
SQL COMMUNICATION AREA:
 SQLCABC = 136 AT X'03C92789'
 SQLCODE = 000 AT X'03C9278D'
 SQLERRML = 000 AT X'03C92791'
 SQLERRMC = ' AT X'03C92793'
 SQLERRP = 'DSN' AT X'03C927D9'
 SQLERRD(1-6) = 000, 000, 00000, -1, 00000, 000 AT X'03C927E1'
 SQLWARN(0-A) = ' AT X'03C927F9'
 SQLSTATE = 00000 AT X'03C92804'
+ OVAR 001: TYPE=INTEGER, LEN=00004, IND=000 AT X'03C920A0'
 DATA=X'00000001'
 OFFSET:X'001D14' LINE:UNKNOWN EIBFN=X'1802'

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : UNDEFINED PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK
```

Figura 78. Pantalla de EDF después de una instrucción SQL de e Db2

La información SQL de la base de datos ( Db2 ) en esta pantalla es la siguiente:

- *P.AUTH=ID de autorización principal*

El ID de autorización principal de Db2 .

- *S.AUTH=ID de autorización secundaria*

El ID de autorización secundaria. Si la RACF lista de opciones de grupo no está activa, Db2 utiliza el nombre de grupo conectado que CICS función de adjuntar proporciona como ID de autorización secundaria. Si la RACF lista de opciones de grupo está activa, Db2 ignora el nombre de grupo conectado que CICS función de adjuntar proporciona, pero el valor se muestra en la lista de Db2 de ID de autorización secundaria.

- *PLAN=nombre del plan*

El nombre del plan que se está ejecutando actualmente. El PLAN representa la estructura de control que se produce durante el proceso de enlace y que utiliza Db2 para procesar las sentencias SQL que se encuentran mientras se ejecuta la aplicación.

- Área de comunicaciones de SQL (SQLCA)

Información en el SQLCA. El SQLCA contiene información sobre los errores, si se producen. Db2 utiliza el SQLCA para proporcionar a un programa de aplicación información sobre las instrucciones SQL que se están ejecutando.

Los signos más (+) a la izquierda de la pantalla indican que puede ver resultados adicionales de EDF utilizando las teclas PF para desplazarse hacia adelante o hacia atrás por la pantalla.

La sección OVAR (variables de host de salida) y sus campos correspondientes se muestran solo cuando la instrucción de ejecución devuelve variables de host de salida.

La siguiente figura contiene el resto de la salida EDF para este ejemplo.

Figura 79. Pantalla de EDF después de una instrucción SQL de e Db2 , continuación

La función de adjuntar muestra automáticamente la información SQL mientras está en el modo EDF. (Puede iniciar EDF como se indica en el CICS manual de referencia del programador de aplicaciones correspondiente) Si no se muestra esta información, póngase en contacto con la persona responsable de la instalación y migración de Db2.

## **Conceptos relacionados**

## Tipos de datos de columnas

Al crear una tabla de Db2, define cada columna para que tenga un tipo de datos específico. El tipo de datos de una columna determina qué puede y qué no puede hacer con la columna.

## Variables de indicación, matrices y estructuras

Una variable de indicación se asocia con una variable host concreta. Cada variable de indicación contiene un valor entero pequeño que indica alguna información sobre la variable host asociada. Las estructuras y matrices de indicador tienen el mismo propósito para las estructuras y matrices de variables de host.

#### **Información relacionada**

Ayudas de depuración de CICS (CICS Transaction Server for z/OS)

## Encontrar una referencia violada o una restricción de verificación

Cuando reciba un error SQL debido a una violación de restricción, consulte el SQLCA para obtener información específica.

## Acerca de esta tarea

**Pregunta :** Cuando se ha violado una restricción referencial o de verificación, ¿cómo puedo determinar cuál es?

**Respuesta :** Cuando reciba un error SQL debido a una violación de restricción, imprima el SQLCA. Puede utilizar la rutina DSNTIAR para formatear el SQLCA por usted. Compruebe el texto de inserción del mensaje de error SQL (SQLERRM) para ver el nombre de la restricción. Para obtener información sobre posibles infracciones, consulte los códigos SQL -530 a -548.

### Conceptos relacionados

[Códigos SQL de error \(-\) \(códigos de error de la base de datos \[ Db2 \]\)](#)

### Tareas relacionadas

[Visualización de campos SQLCA invocando DSNTIAR](#)

Si utiliza el SQLCA para comprobar si una sentencia SQL se ha ejecutado correctamente, el programa necesita leer los datos de los campos SQLCA adecuados. Una forma sencilla de leer estos campos es utilizar la subrutina de ensamblador DSNTIAR.



# Capítulo 12. Datos de muestra y aplicaciones suministrados con Db2 for z/OS

Puede utilizar las aplicaciones de muestra que se incluyen con Db2 for z/OS para aprender a programar aplicaciones que aprovechan las capacidades de Db2. Db2 también proporciona modelos para sus propias situaciones.

Para preparar y ejecutar las aplicaciones de muestra suministradas, utilice el JCL en *prefijo.SDSNSAMP* como modelo:

## Referencia relacionada

[Tablas de muestra de Db2 \(Introducción a Db2 para z/OS\)](#)

## Tablas de ejemplo de Db2

Gran parte de la información de Db2 hace referencia o se basa en tablas de ejemplo de Db2. Como grupo, las tablas incluyen información que describe empleados, departamentos, proyectos y actividades y forman una aplicación de ejemplo que ilustra muchas de las características de Db2.

### GUPI

El grupo de almacenamiento, las bases de datos, los espacios de tablas, las tablas y vistas de ejemplo se crean cuando se ejecutan los trabajos de ejemplo de instalación DSNTEJ1 y DSNTEJ7. Los objetos de ejemplo de Db2 que incluyen LOB se crean en el trabajo DSNTEJ7. Los demás objetos de ejemplo se crean en el trabajo DSNTEJ1. Las sentencias CREATE INDEX para las tablas de ejemplo no se muestran aquí; también se crean mediante los trabajos de ejemplo DSNTEJ1 y DSNTEJ7.

Se proporciona la autorización PUBLIC sobre todos los objetos de ejemplo para que los programas de ejemplo sean más fáciles de ejecutar. Puede revisar el contenido de cualquier tabla ejecutando una sentencia SQL, por ejemplo SELECT \* FROM DSN8C10.PROJ. Para facilitar la interpretación de los ejemplos, las tablas de departamentos y empleados se listan completas.

### GUPI

## Conceptos relacionados

[Fase 1: Creación y carga de tablas de muestra \(Db2 Installation and Migration\)](#)

## Tabla de actividades (DSN8C10.ACT)

La tabla de actividades describe las actividades que se pueden realizar durante un proyecto.

### GUPI

La tabla de actividades reside en la base de datos DSN8D12A y se crea con la siguiente sentencia:

```
CREATE TABLE DSN8C10.ACT
 (ACTNO SMALLINT NOT NULL,
 ACTKWD CHAR(6) NOT NULL,
 ACTDESC VARCHAR(20) NOT NULL,
 PRIMARY KEY (ACTNO)
)
IN DSN8D12A.DSN8S12P
CCSID EBCDIC;
```

### GUPI

## Contenido de la tabla de actividades

La tabla siguiente muestra el contenido de las columnas de la tabla de actividades.

Tabla 156. Columnas de la tabla de actividades

Columna	Nombre de columna	Descripción
1	ACTNO	ID de actividad (clave primaria)
2	ACTKWD	Palabra clave de actividad (seis caracteres como máximo)
3	ACTDESC	Descripción de actividad

La tabla de actividades contiene los índices siguientes.

Tabla 157. Índices de la tabla de actividades

Nombre	En la columna	Tipo de índice
DSN8C10.XACT1	ACTNO	Primario, ascendente
DSN8C10.XACT2	ACTKWD	Exclusivo, ascendente

## Relación con otras tablas

La tabla de actividades es una tabla padre de la tabla de actividades de un proyecto, mediante una clave foránea en la columna ACTNO.

## Tabla de departamentos (DSN8C10.DEPT)

La tabla de departamentos describe cada departamento de la empresa e identifica su director y el departamento al cual informa.



La tabla del departamento reside en el espacio de tabla DSN8D12A.DSN8S12D y se crea con la siguiente declaración:

```
CREATE TABLE DSN8C10.DEPT
 (DEPTNO CHAR(3) NOT NULL,
 DEPTNAME VARCHAR(36) NOT NULL,
 MGRNO CHAR(6) ,
 ADMRDEPT CHAR(3) NOT NULL,
 LOCATION CHAR(16) ,
 PRIMARY KEY (DEPTNO)
)
IN DSN8D12A.DSN8S12D
CCSID EBCDIC;
```

Debido a que la tabla de departamentos hace referencia a sí misma y también forma parte de un ciclo de dependencias, sus claves foráneas deben añadirse más adelante con las sentencias siguientes:

```
ALTER TABLE DSN8C10.DEPT
 FOREIGN KEY RDD (ADMRDEPT) REFERENCES DSN8C10.DEPT
 ON DELETE CASCADE;

ALTER TABLE DSN8C10.DEPT
 FOREIGN KEY RDE (MGRNO) REFERENCES DSN8C10.EMP
 ON DELETE SET NULL;
```



## Contenido de la tabla de departamentos

La tabla siguiente muestra el contenido de las columnas de la tabla de departamentos.

*Tabla 158. Columna de la tabla de departamentos*

<b>Columna</b>	<b>Nombre de columna</b>	<b>Descripción</b>
1	DEPTNO	ID de departamento, clave primaria.
2	DEPTNAME	Nombre que describe las actividades generales del departamento.
3	MGRNO	Número de empleado (EMPNO) del director de departamento.
4	ADMREDEPT	ID del departamento al cual informa este departamento; el departamento en el nivel superior informa a sí mismo.
5	UBICACIÓN	El nombre de ubicación remota.

La tabla siguiente muestra los índices de la tabla de departamentos.

*Tabla 159. Índices de la tabla de departamentos*

<b>Nombre</b>	<b>En la columna</b>	<b>Tipo de índice</b>
DSN8C10.XDEPT1	DEPTNO	Primario, ascendente
DSN8C10.XDEPT2	MGRNO	Ascendente
DSN8C10.XDEPT3	ADMREDEPT	Ascendente

La tabla siguiente muestra el contenido de la tabla de departamentos.

*Tabla 160. DSN8C10.DEPT: tabla de departamentos*

<b>DEPTNO</b>	<b>DEPTNAME</b>	<b>MGRNO</b>	<b>ADMREDEPT</b>	<b>UBICACIÓN</b>
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	-----
B01	PLANNING	000020	A00	-----
C01	INFORMATION CENTER	000030	A00	-----
D01	DEVELOPMENT CENTER	-----	A00	-----
E01	SUPPORT SERVICES	000050	A00	-----
D11	MANUFACTURING SYSTEMS	000060	D01	-----
D21	ADMINISTRATION SYSTEMS	000070	D01	-----
E11	OPERATIONS	000090	E01	-----
E21	SOFTWARE SUPPORT	000100	E01	-----
F22	BRANCH OFFICE F2	-----	E01	-----
G22	BRANCH OFFICE G2	-----	E01	-----
H22	BRANCH OFFICE H2	-----	E01	-----
I22	BRANCH OFFICE I2	-----	E01	-----
J22	BRANCH OFFICE J2	-----	E01	-----

La columna LOCATION contiene valores nulos hasta que el trabajo de ejemplo DSNTEJ6 actualiza esta columna con el nombre de ubicación.

## Relación con otras tablas

La tabla de departamentos hace referencia a sí misma: el valor del departamento de administración debe ser un ID de departamento válido.

La tabla de departamentos es una tabla padre de las siguientes:

- La tabla de empleados, mediante una clave foránea en la columna WORKDEPT
- La tabla de proyectos, mediante una clave foránea en la columna DEPTNO

La tabla de departamentos depende de la tabla de empleados, mediante su clave primaria en la columna MGRNO.

## Tabla de empleados (DSN8C10.EMP)

La tabla de empleados de ejemplo identifica todos los empleados mediante un número de empleado y lista la información de personal básica.

**GUPI** La tabla de empleados reside en el espacio de tabla particionado DSN8D12A.DSN8S12E. Debido a que esta tabla tiene una clave foránea que hace referencia a DEPT, la tabla y el índice de su clave primaria deben crearse primero. A continuación, EMP se crea con la sentencia siguiente:

```
CREATE TABLE DSN8C10.EMP
 (EMPNO CHAR(6) NOT NULL,
 FIRSTNME VARCHAR(12) NOT NULL,
 MIDINIT CHAR(1) NOT NULL,
 LASTNAME VARCHAR(15) NOT NULL,
 WORKDEPT CHAR(3) ,
 PHONENO CHAR(4) CONSTRAINT NUMBER CHECK
 (PHONENO >= '0000' AND
 PHONENO <= '9999') ,
 HIREDATE DATE ,
 JOB CHAR(8) ,
 EDLEVEL SMALLINT ,
 SEX CHAR(1) ,
 BIRTHDATE DATE ,
 SALARY DECIMAL(9,2) ,
 BONUS DECIMAL(9,2) ,
 COMM DECIMAL(9,2) ,
 PRIMARY KEY (EMPNO)
 FOREIGN KEY RED (WORKDEPT) REFERENCES DSN8C10.DEPT
 ON DELETE SET NULL
)
EDITPROC DSN8EAE1
IN DSN8D12A.DSN8S12E
CCSID EBCDIC;
```

**GUPI**

## Contenido de la tabla de empleados

La tabla siguiente muestra el tipo de contenido de cada una de las columnas de la tabla de empleados. La tabla tiene una restricción de comprobación, NUMBER, que comprueba que el número de teléfono de cuatro dígitos esté dentro del rango numérico de 0000 a 9999.

Tabla 161. Columnas de la tabla de empleados

Columna	Nombre de columna	Descripción
1	EMPNO	Número de empleado (clave primaria)
2	FIRSTNME	Nombre del empleado
3	MIDINIT	Inicial media del empleado
4	LASTNAME	Apellido del empleado
5	WORKDEPT	ID de departamento en el que trabaja el empleado
6	PHONENO	Número de teléfono del empleado

---

*Tabla 161. Columnas de la tabla de empleados (continuación)*

---

<b>Columna</b>	<b>Nombre de columna</b>	<b>Descripción</b>
7	HIREDATE	Fecha de contratación
8	JOB	Ocupación del empleado
9	EDLEVEL	Número de años de educación formal
10	SEX	Sexo del empleado (M o F)
11	BIRTHDATE	Fecha de nacimiento
12	SALARY	Salario anual en dólares
13	BONUS	Bonificación anual en dólares
14	COMM	Comisión anual en dólares

---

La tabla siguiente muestra los índices de la tabla de empleados.

*Tabla 162. Índices de la tabla de empleados*

---

<b>Nombre</b>	<b>En la columna</b>	<b>Tipo de índice</b>
DSN8C10.XEMP1	EMPNO	Primario, particionado, ascendente
DSN8C10.XEMP2	WORKDEPT	Ascendente

---

La tabla siguiente muestra la primera mitad (lado izquierdo) del contenido de la tabla de empleados.  
([Tabla 164 en la página 1058](#) muestra el contenido restante (lado derecho) de la tabla de empleados.)

*Tabla 163. Mitad izquierda de DSN8C10.EMP: tabla de empleados.* Observe que un espacio en blanco en la columna MIDINIT es un valor real de " " en lugar de un nulo.

---

<b>EMPNO</b>	<b>FIRSTNAME</b>	<b>MIDINIT</b>	<b>LASTNAME</b>	<b>WORKDEPT</b>	<b>PHONE NO</b>	<b>HIREDATE</b>
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10
000030	SALLY	A	KWAN	C01	4738	1975-04-05
000050	JOHN	B	GEYER	E01	6789	1949-08-17
000060	IRVING	F	STERN	D11	6423	1973-09-14
000070	EVA	D	PULASKI	D21	7831	1980-09-30
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19
000110	VINCENZO	G	LUCCHESI	A00	3490	1958-05-16
000120	SEAN		O'CONNELL	A00	2167	1963-12-05
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15
000150	BRUCE		ADAMSON	D11	4510	1972-02-12
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07
000190	JAMES	H	WALKER	D11	2986	1974-07-26

---

Tabla 163. Mitad izquierda de DSN8C10.EMP: tabla de empleados. Observe que un espacio en blanco en la columna MIDINIT es un valor real de " " en lugar de un nulo. (continuación)

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000200	DAVID		BROWN	D11	4501	1966-03-03
000210	WILLIAM	T	JONES	D11	0942	1979-04-11
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05
000250	DANIEL	S	SMITH	D21	0961	1969-10-30
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11
000270	MARIA	L	PEREZ	D21	9001	1980-09-30
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24
000290	JOHN	R	PARKER	E11	4502	1980-05-30
000300	PHILIP	X	SMITH	E11	2095	1972-06-19
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07
000330	WING		LEE	E21	2103	1976-02-23
000340	JASON	R	GOUNOT	E21	5698	1947-05-05
200010	DIAN	J	HEMMINGER	A00	3978	1965-01-01
200120	GREG		ORLANDO	A00	2167	1972-05-05
200140	KIM	N	NATZ	C01	1793	1976-12-15
200170	KIYOSHI		YAMAMOTO	D11	2890	1978-09-15
200220	REBA	K	JOHN	D11	0672	1968-08-29
200240	ROBERT	M	MONTEVERDE	D21	3780	1979-12-05
200280	EILEEN	R	SCHWARTZ	E11	8997	1967-03-24
200310	MICHELLE	F	SPRINGER	E11	3332	1964-09-12
200330	HELENA		WONG	E21	2103	1976-02-23
200340	ROY	R	ALONZO	E21	5698	1947-05-05

(Tabla 163 en la página 1057 muestra la primera mitad (lado derecho) del contenido de la tabla de empleados.)

Tabla 164. Mitad derecha de DSN8C10.EMP: tabla de empleados

(EMPNO)	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
(000010)	PRES	18	F	1933-08-14	52750.00	1000.00	4220.00
(000020)	MANAGER	18	M	1948-02-02	41250.00	800.00	3300.00
(000030)	MANAGER	20	F	1941-05-11	38250.00	800.00	3060.00
(000050)	MANAGER	16	M	1925-09-15	40175.00	800.00	3214.00
(000060)	MANAGER	16	M	1945-07-07	32250.00	600.00	2580.00
(000070)	MANAGER	16	F	1953-05-26	36170.00	700.00	2893.00

Tabla 164. Mitad derecha de DSN8C10.EMP: tabla de empleados (continuación)

(EMPNO)	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
(000090)	MANAGER	16	F	1941-05-15	29750.00	600.00	2380.00
(000100)	MANAGER	14	M	1956-12-18	26150.00	500.00	2092.00
(000110)	SALESREP	19	M	1929-11-05	46500.00	900.00	3720.00
(000120)	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
(000130)	ANALYST	16	F	1925-09-15	23800.00	500.00	1904.00
(000140)	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
(000150)	DESIGNER	16	M	1947-05-17	25280.00	500.00	2022.00
(000160)	DESIGNER	17	F	1955-04-12	22250.00	400.00	1780.00
(000170)	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
(000180)	DESIGNER	17	F	1949-02-21	21340.00	500.00	1707.00
(000190)	DESIGNER	16	M	1952-06-25	20450.00	400.00	1636.00
(000200)	DESIGNER	16	M	1941-05-29	27740.00	600.00	2217.00
(000210)	DESIGNER	17	M	1953-02-23	18270.00	400.00	1462.00
(000220)	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
(000230)	CLERK	14	M	1935-05-30	22180.00	400.00	1774.00
(000240)	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
(000250)	CLERK	15	M	1939-11-12	19180.00	400.00	1534.00
(000260)	CLERK	16	F	1936-10-05	17250.00	300.00	1380.00
(000270)	CLERK	15	F	1953-05-26	27380.00	500.00	2190.00
(000280)	OPERADOR	17	F	1936-03-28	26250.00	500.00	2100.00
(000290)	OPERADOR	12	M	1946-07-09	15340.00	300.00	1227.00
(000300)	OPERADOR	14	M	1936-10-27	17750.00	400.00	1420.00
(000310)	OPERADOR	12	F	1931-04-21	15900.00	300.00	1272.00
(000320)	FIELDREP	16	M	1932-08-11	19950.00	400.00	1596.00
(000330)	FIELDREP	14	M	1941-07-18	25370.00	500.00	2030.00
(000340)	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00
(200010)	SALESREP	18	F	1933-08-14	46500.00	1000.00	4220.00
(200120)	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
(200140)	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
(200170)	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
(200220)	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
(200240)	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
(200280)	OPERADOR	17	F	1936-03-28	26250.00	500.00	2100.00
(200310)	OPERADOR	12	F	1931-04-21	15900.00	300.00	1272.00
(200330)	FIELDREP	14	F	1941-07-18	25370.00	500.00	2030.00
(200340)	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00

## Relación con otras tablas

La tabla de empleados es una tabla padre de:

- La tabla de departamentos, mediante una clave foránea en la columna MGRNO
- La tabla de proyectos, mediante una clave foránea en la columna RESPEMP

La tabla de empleados depende de la tabla de departamentos, mediante su clave foránea en la columna WORKDEPT.

## Tabla de fotografías y currículums de empleados (DSN8C10.EMP\_PHOTO\_RESUME)

La tabla de fotografías y currículums de empleados de ejemplo complementa la tabla de empleados.

**GUPI** Cada fila de la tabla de fotos y currículum contiene una foto del empleado, en dos formatos, y el currículum del empleado. La tabla de fotografías y currículums de empleados en el espacio de tablas de DSN8D12L.DSN8S12B. La sentencia siguiente crea la tabla:

```
CREATE TABLE DSN8C10.EMP_PHOTO_RESUME
 (EMPNO CHAR(06) NOT NULL,
 EMP_ROWID ROWID NOT NULL GENERATED ALWAYS,
 PSEG_PHOTO BLOB(500K),
 BMP_PHOTO BLOB(100K),
 RESUME CLOB(5K),
 PRIMARY KEY (EMPNO))
IN DSN8D12L.DSN8S12B
CCSID EBCDIC;
```

Db2 requiere una tabla auxiliar para cada columna LOB de una tabla. Las sentencias siguientes definen las tablas auxiliares para las tres columnas LOB en DSN8C10.EMP\_PHOTO\_RESUME:

```
CREATE AUX TABLE DSN8C10.AUX_BMP_PHOTO
 IN DSN8D12L.DSN8S12M
 STORES DSN8C10.EMP_PHOTO_RESUME
 COLUMN BMP_PHOTO;

CREATE AUX TABLE DSN8C10.AUX_PSEG_PHOTO
 IN DSN8D12L.DSN8S12L
 STORES DSN8C10.EMP_PHOTO_RESUME
 COLUMN PSEG_PHOTO;

CREATE AUX TABLE DSN8C10.AUX_EMP_RESUME
 IN DSN8D12L.DSN8S12N
 STORES DSN8C10.EMP_PHOTO_RESUME
 COLUMN RESUME;
```

**GUPI**

## Contenido de la tabla de fotografías y currículums

La tabla siguiente muestra el contenido de las columnas de la tabla de fotografías y currículums de empleados.

Tabla 165. Columnas de la tabla de fotografías y currículums de empleados

Columna	Nombre de columna	Descripción
1	EMPNO	ID de empleado (clave primaria).
2	EMP_ROWID	ID de fila para identificar exclusivamente cada fila de la tabla. Db2 proporciona los valores de esta columna.
3	PSEG_PHOTO	Fotografía del empleado, en formato PSEG.
4	BMP_PHOTO	Fotografía del empleado, en formato BMP.
5	RESUME	Currículum del empleado.

La tabla siguiente muestra los índices para la tabla de fotografías y currículums de empleados.

*Tabla 166. Índices de la tabla de fotografías y currículums de empleados*

Nombre	En la columna	Tipo de índice
DSN8C10.XEMP_PHOTO_RESUME	EMPNO	Primario, ascendente

La tabla siguiente muestra los índices para las tablas auxiliares que dan soporte a la tabla de fotografía y currículums de empleados.

*Tabla 167. Índices de las tablas auxiliares para la tabla de fotografías y currículums de empleados*

Nombre	En la tabla	Tipo de índice
DSN8C10.XAUX_BMP_PHOTO	DSN8C10.AUX_BMP_PHOTO	Exclusivo
DSN8C10.XAUX_PSEG_PHOTO	DSN8C10.AUX_PSEG_PHOTO	Exclusivo
DSN8C10.XAUX_EMP_RESUME	DSN8C10.AUX_EMP_RESUME	Exclusivo

## Relación con otras tablas

La tabla de fotografías y currículums de empleados es una tabla padre de la tabla de proyectos, mediante una clave foránea en la columna RESPEMP.

## Tabla de proyectos (DSN8C10.PROJ)

La tabla de proyectos de ejemplo describe cada proyecto que la empresa está llevando a cabo actualmente. Los datos de cada fila de la tabla incluyen el número de proyecto, el nombre, la persona responsable y las fechas de planificación.

La tabla de proyectos reside en la base de datos DSN8D12A. Dado que esta tabla tiene claves foráneas que hacen referencia a DEPT y EMP, estas tablas y los índices de sus claves primarias deben crearse primero. A continuación, PROJ se crea con la sentencia siguiente:

GUPI

```
CREATE TABLE DSN8C10.PROJ
 (PROJNO CHAR(6) PRIMARY KEY NOT NULL,
 PROJNAME VARCHAR(24) NOT NULL WITH DEFAULT
 'PROJECT NAME UNDEFINED',
 DEPTNO CHAR(3) NOT NULL REFERENCES
 DSN8C10.DEPT ON DELETE RESTRICT,
 RESPEMP CHAR(6) NOT NULL REFERENCES
 DSN8C10.EMP ON DELETE RESTRICT,
 PRSTAFF DECIMAL(5, 2) ,
 PRSTDATE DATE ,
 PRENDATE DATE ,
 MAJPROJ CHAR(6))
IN DSN8D12A.DSN8S12P
CCSID EBCDIC;
```

Debido a que la tabla de proyectos hace referencia a sí misma la clave foránea para esta restricción debe añadirse más adelante con la sentencia siguiente:

```
ALTER TABLE DSN8C10.PROJ
 FOREIGN KEY RPP (MAJPROJ) REFERENCES DSN8C10.PROJ
 ON DELETE CASCADE;
```

GUPI

## Contenido de la tabla de proyectos

La tabla siguiente muestra el contenido de las columnas de la tabla de proyectos.

Tabla 168. Columnas de la tabla de proyectos

Columna	Nombre de columna	Descripción
1	PROJNO	ID de proyecto (clave primaria)
2	PROJNAME	Nombre de proyecto
3	DEPTNO	ID del departamento responsable del proyecto
4	RESPEMP	ID del empleado responsable del proyecto
5	PRSTAFF	Número medio estimado de personas necesarias entre PRSTDAT y PRENDAT para completar el proyecto entero, incluidos los subproyectos
6	PRSTDAT	Fecha de inicio estimada del proyecto
7	PRENDAT	Fecha de finalización estimada del proyecto
8	MAJPROJ	ID de cualquier proyecto del que forme parte este proyecto

La tabla siguiente muestra los índices para la tabla de proyectos:

Tabla 169. Índices de la tabla de proyectos

Nombre	En la columna	Tipo de índice
DSN8C10.XPROJ1	PROJNO	Primario, ascendente
DSN8C10.XPROJ2	RESPEMP	Ascendente

## Relación con otras tablas

La tabla hace referencia a sí misma: un valor no nulo de MAJPROJ debe ser un número de proyecto válido. La tabla es una tabla padre de la tabla de actividades de proyectos, mediante una clave foránea en la columna PROJNO. Depende de las tablas siguientes:

- La tabla de departamentos, mediante su clave foránea en DEPTNO
- La tabla de empleados, mediante su clave foránea en RESPEMP

## Tabla de actividades de proyectos (DSN8C10.PROJECT)

La tabla de actividades de proyectos de ejemplo lista las actividades que se realizan en cada proyecto.

La tabla de actividades de proyectos reside en la base de datos DSN8D12A. Debido a que esta tabla tiene claves foráneas que hacen referencia a PROJ y ACT, estas tablas y los índices de sus claves primarias deben crearse primero. A continuación, PROJECT se crea con la sentencia siguiente:

GUPI

```
CREATE TABLE DSN8C10.PROJECT
 (PROJNO CHAR(6) NOT NULL,
 ACTNO SMALLINT NOT NULL,
 ACSTAFF DECIMAL(5,2),
 ACSTDAT DATE NOT NULL,
 ACENDAT DATE ,
 PRIMARY KEY (PROJNO, ACTNO, ACSTDAT),
 FOREIGN KEY RPAP (PROJNO) REFERENCES DSN8C10.PROJ
 ON DELETE RESTRICT,
 FOREIGN KEY RPAA (ACTNO) REFERENCES DSN8C10.ACT
 ON DELETE RESTRICT)
IN DSN8D12A.DSN8S12P
CCSID EBCDIC;
```

GUPI

## Contenido de la tabla de actividades de proyectos

La tabla siguiente muestra el contenido de las columnas de la tabla de actividades de proyectos.

Tabla 170. Columnas de la tabla de actividades de proyectos

Columna	Nombre de columna	Descripción
1	PROJNO	ID de proyecto
2	ACTNO	ID de actividad
3	ACSTAFF	Número medio estimado de empleados que se necesitan para realizar la actividad
4	ACSTDAT	Fecha de inicio estimada de la actividad
5	ACENDAT	Fecha de finalización estimada de la actividad

La tabla siguiente muestra el índice de la tabla de actividades de proyectos:

Tabla 171. Índice de la tabla de actividades de proyectos

Nombre	En columnas	Tipo de índice
DSN8C10.XPROJAC1	PROJNO, ACTNO, ACSTDAT	primario, ascendente

## Relación con otras tablas

La tabla de actividades de proyectos es una tabla padre de la tabla de empleados de actividades de proyectos, mediante una clave foránea en las columnas PROJNO, ACTNO y EMSTDAT. Depende de las tablas siguientes:

- La tabla de actividades, mediante su clave foránea en la columna ACTNO
- La tabla de proyectos, mediante su clave foránea en la columna PROJNO

### Referencia relacionada

[Tabla de actividades \( DSN8C10.ACT \) \(Introducción a Db2 for z/OS \)](#)

[Tabla de proyectos \( DSN8C10 .PROJ \) \(Introducción a Db2 for z/OS \)](#)

## Tabla de empleados de actividades de proyectos (DSN8C10.EMPPROJECT)

La tabla de empleados de actividades de proyectos de ejemplo identifica los empleados que realizan una actividad para un proyecto, indica la proporción de tiempo necesario del empleado y proporciona una planificación para la actividad.

### GUPI

La tabla de empleados de actividades de proyectos reside en la base de datos DSN8D12A. Dado que esta tabla tiene claves foráneas que hacen referencia a EMP y PROJECT, estas tablas y los índices de sus claves primarias deben crearse primero. A continuación, EMPPROJECT se crea con la sentencia siguiente:

```
CREATE TABLE DSN8C10.EMPPROJECT
 (EMPNO CHAR(6) NOT NULL,
 PROJNO CHAR(6) NOT NULL,
 ACTNO SMALLINT NOT NULL,
 EMPTIME DECIMAL(5,2) ,
 EMSTDAT DATE ,
 EMENDAT DATE ,
 FOREIGN KEY REPAPA (PROJNO, ACTNO, EMSTDAT)
 REFERENCES DSN8C10.PROJECT
 ON DELETE RESTRICT,
 FOREIGN KEY REPAE (EMPNO) REFERENCES DSN8C10.EMP
 ON DELETE RESTRICT)
IN DSN8D12A.DSN8S12P
CCSID EBCDIC;
```

## Contenido de la tabla de empleados de actividades de proyectos

La tabla siguiente muestra el contenido de las columnas de la tabla de empleados de actividades de proyectos.

*Tabla 172. Columnas de la tabla de empleados de actividades de proyectos*

Columna	Nombre de columna	Descripción
1	EMPNO	Número de ID del empleado
2	PROJNO	ID de proyecto del proyecto
3	ACTNO	ID de actividad dentro del proyecto
4	EMPTIME	Proporción del tiempo completo del empleado (entre 0,00 y 1,00) que debe dedicarse a la actividad
5	EMSTDATE	Fecha de inicio de la actividad
6	EMENDATE	Fecha de finalización de la actividad

La tabla siguiente muestra los índices para la tabla de empleados de actividades de proyectos:

*Tabla 173. Índices de la tabla de empleados de actividades de proyectos*

Nombre	En columnas	Tipo de índice
DSN8C10.XEMPPROJECT1	PROJNO, ACTNO, EMSTDATE, EMPNO	Exclusivo, ascendente
DSN8C10.XEMPPROJECT2	EMPNO	Ascendente

## Relación con otras tablas

La tabla de empleados de actividades de proyectos depende de las tablas siguientes:

- La tabla de empleados, mediante su clave foránea en la columna EMPNO
- La tabla de actividades de proyectos, mediante su clave foránea en las columnas PROJNO, ACTNO y EMSTDATE.

### Referencia relacionada

[Tabla de empleados \( DSN8C10. EMP \) \(Introducción a Db2 for z/OS \)](#)

[Tabla de actividades del proyecto \( DSN8C10 .PROJECT \) \(Introducción a Db2 for z/OS \)](#)

## Tabla de ejemplo Unicode (DSN8C10.DEMO\_UNICODE)

La tabla de ejemplo Unicode se utiliza para comprobar que las conversiones de datos entre EBCDIC y Unicode funcionen como se espera.

La tabla reside en la base de datos DSN8D12A, y se define con la siguiente sentencia:

```
CREATE TABLE DSN8C10.DEMO_UNICODE
 (LOWER_A_TO_Z CHAR(26)
 UPPER_A_TO_Z CHAR(26)
 ZERO_TO_NINE CHAR(10)
 X00_TO_XFF VARCHAR(256) FOR BIT DATA)
 ,
 IN DSN8D81E.DSN8S81U
 CCSID UNICODE;
```

## Contenido de la tabla de ejemplo Unicode

La tabla siguiente muestra el contenido de las columnas de la tabla de ejemplo Unicode:

Tabla 174. Columnas de la tabla de ejemplo Unicode

Columna	Nombre de columna	Descripción
1	LOWER_A_TO_Z	Matriz de caracteres, de 'a' a 'z'
2	UPPER_A_TO_Z	Matriz de caracteres, de 'A' a 'Z'
3	ZERO_TO_NINE	Matriz de caracteres, de '0' a '9'
4	X00_TO_XFF	Matriz de caracteres, de x'00' a x'FF'

Esta tabla no tiene índices.

## Relación con otras tablas

Esta tabla no tiene ninguna relación con otras tablas.

## Relaciones entre las tablas de ejemplo

Las relaciones entre las tablas de ejemplo se establecen mediante claves foráneas en tablas dependientes que hacen referencia a claves primarias de tablas padre.

La figura siguiente muestra las relaciones entre las tablas de ejemplo. Encontrará las descripciones de las columnas en las descripciones de las tablas.

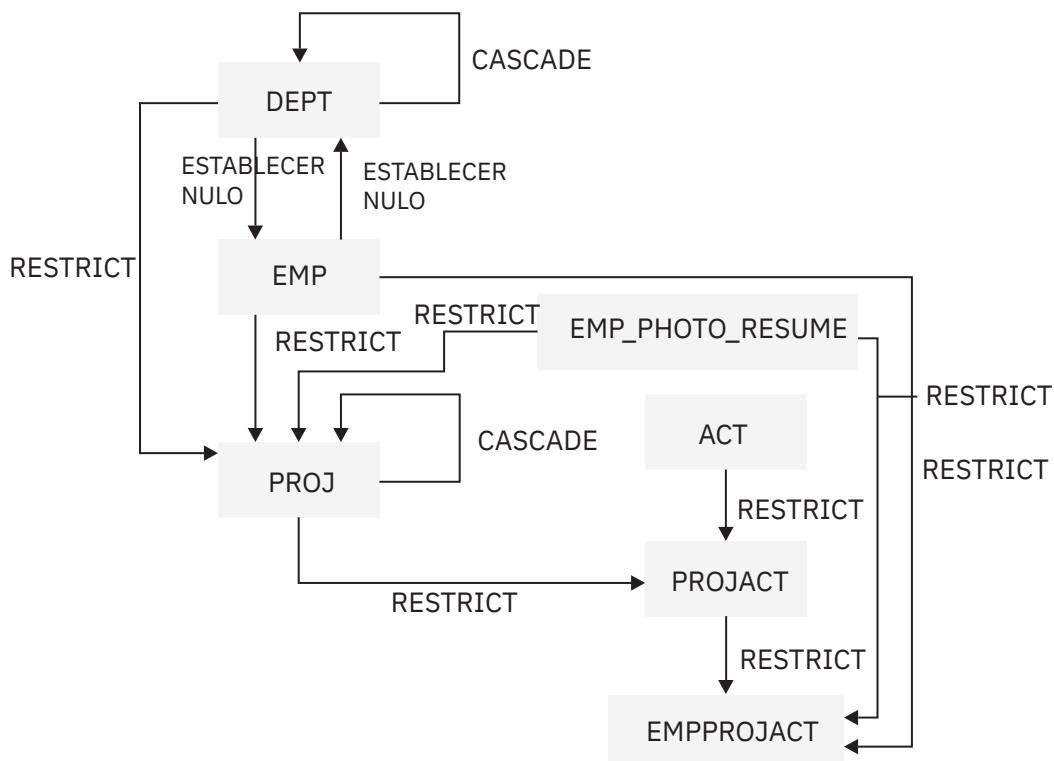


Figura 80. Relaciones entre tablas en la aplicación de ejemplo

### Referencia relacionada

Tabla de actividades ( DSN8C10.ACT ) (Introducción a Db2 for z/OS )

Tabla de departamentos ( DSN8C10. DEPT ) (Introducción a Db2 for z/OS )

[Tabla de fotos y currículums de empleados \( DSN8C10. EMP\\_PHOTO\\_RESUME \) \(Introducción a Db2 for z/OS \)](#)

[Tabla de empleados \( DSN8C10. EMP \) \(Introducción a Db2 for z/OS \)](#)

[Tabla de actividad de empleado a proyecto \( DSN8C10. EMPPROJECT \) \(Introducción a Db2 for z/OS \)](#)

[Tabla de actividades del proyecto \( DSN8C10 .PROJACT \) \(Introducción a Db2 for z/OS \)](#)

[Tabla de proyectos \( DSN8C10 .PROJ \) \(Introducción a Db2 for z/OS \)](#)

[Tabla de muestra Unicode \( DSN8C10.DEMO\\_UNICODE\) \(Introducción a Db2 for z/OS \)](#)

## Vistas en las tablas de ejemplo

Db2 crea un número de vistas en las tablas de ejemplo para utilizarlas en las aplicaciones de ejemplo.

La tabla siguiente indica las tablas en las que se define cada vista y las aplicaciones de ejemplo que utilizan la vista. Todos los nombres de vista tienen el calificador DSN8C10.

*Tabla 175. Vistas en tablas de ejemplo*

Nombre de vista	En tablas o vistas	Utilizada en la aplicación
VDEPT	DEPT	Organización Proyecto
VHDEPT	DEPT	Organización distribuida
VEMP	EMP	Organización distribuida Organización Proyecto
VPROJ	PROJ	Proyecto
VACT	ACT	Proyecto
VPROJACT	PROJECT	Proyecto
VEMPPROJECT	EMPPROJECT	Proyecto
VDEPMG1	DEPT EMP	Organización
VEMPDPT1	DEPT EMP	Organización
VASTRDE1	DEPT	
VASTRDE2	VDEPMG1 EMP	Organización
VPROJRE1	PROJ EMP	Proyecto
VPSTRDE1	VPROJRE1 VPROJRE2	Proyecto
VPSTRDE2	VPROJRE1	Proyecto
VFORPLA	VPROJRE1 EMPPROJECT	Proyecto

Tabla 175. Vistas en tablas de ejemplo (continuación)

Nombre de vista	En tablas o vistas	Utilizada en la aplicación
VSTAFAC1	PROJACT ACT	Proyecto
VSTAFAC2	EMPPROJACT ACT EMP	Proyecto
VPHONE	EMP DEPT	Teléfono
VEMPLP	EMP	Teléfono



La sentencia de SQL siguiente crea la vista denominada VDEPT.

```
CREATE VIEW DSN8C10.VDEPT
 AS SELECT ALL DEPTNO ,
 DEPTNAME,
 MGRNO ,
 ADMRDEPT
 FROM DSN8C10.DEPT;
```

La sentencia de SQL siguiente crea la vista denominada VHDEPT.

```
CREATE VIEW DSN8C10.VHDEPT
 AS SELECT ALL DEPTNO ,
 DEPTNAME,
 MGRNO ,
 ADMRDEPT,
 LOCATION
 FROM DSN8C10.DEPT;
```

La sentencia de SQL siguiente crea la vista denominada VEMP.

```
CREATE VIEW DSN8C10.VEMP
 AS SELECT ALL EMPNO ,
 FIRSTNAME,
 MIDINIT ,
 LASTNAME,
 WORKDEPT
 FROM DSN8C10.EMP;
```

La sentencia de SQL siguiente crea la vista denominada VPROJ.

```
CREATE VIEW DSN8C10.VPROJ
 AS SELECT ALL PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF,
 PRSTDATE, PRENDATE, MAJPROJ
 FROM DSN8C10.PROJ ;
```

La sentencia de SQL siguiente crea la vista denominada VACT.

```
CREATE VIEW DSN8C10.VACT
 AS SELECT ALL ACTNO ,
 ACTKWD ,
 ACTDESC
 FROM DSN8C10.ACT ;
```

La sentencia de SQL siguiente crea la vista denominada VPROJECT.

```
CREATE VIEW DSN8C10.VPROJECT
AS SELECT ALL
 PROJNO,ACTNO, ACSTAFF, ACSTDAT, ACENDAT
 FROM DSN8C10.PROJECT ;
```

La sentencia de SQL siguiente crea la vista denominada VEMPPROJECT.

```
CREATE VIEW DSN8C10.VEMPPROJECT
AS SELECT ALL
 EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDAT, EMENDAT
 FROM DSN8C10.EMPPROJECT ;
```

La sentencia de SQL siguiente crea la vista denominada VDEPMG1.

```
CREATE VIEW DSN8C10.VDEPMG1
(DEPTNO, DEPTNAME, MGRNO, FIRSTNME, MIDINIT,
 LASTNAME, ADMRDEPT)
AS SELECT ALL
 DEPTNO, DEPTNAME, EMPNO, FIRSTNME, MIDINIT,
 LASTNAME, ADMRDEPT
 FROM DSN8C10.DEPT LEFT OUTER JOIN DSN8C10.EMP
 ON MGRNO = EMPNO ;
```

La sentencia de SQL siguiente crea la vista denominada VEMPDPT1.

```
CREATE VIEW DSN8C10.VEMPDPT1
(DEPTNO, DEPTNAME, EMPNO, FRSTINIT, MIDINIT,
 LASTNAME, WORKDEPT)
AS SELECT ALL
 DEPTNO, DEPTNAME, EMPNO, SUBSTR(FIRSTNME, 1, 1), MIDINIT,
 LASTNAME, WORKDEPT
 FROM DSN8C10.DEPT RIGHT OUTER JOIN DSN8C10.EMP
 ON WORKDEPT = DEPTNO ;
```

La sentencia de SQL siguiente crea la vista denominada VASTRDE1.

```
CREATE VIEW DSN8C10.VASTRDE1
(DEPT1NO,DEPT1NAM,EMP1NO,EMP1FN,EMP1MI,EMP1LN,TYPE2,
 DEPT2NO,DEPT2NAM,EMP2NO,EMP2FN,EMP2MI,EMP2LN)
AS SELECT ALL
 D1.DEPTNO,D1.DEPTNAME,D1.MGRNO,D1.FIRSTNME,D1.MIDINIT,
 D1.LASTNAME, '1',
 D2.DEPTNO,D2.DEPTNAME,D2.MGRNO,D2.FIRSTNME,D2.MIDINIT,
 D2.LASTNAME
 FROM DSN8C10.VDEPMG1 D1, DSN8C10.VDEPMG1 D2
 WHERE D1.DEPTNO = D2.ADMRDEPT ;
```

La sentencia de SQL siguiente crea la vista denominada VASTRDE2.

```
CREATE VIEW DSN8C10.VASTRDE2
(DEPT1NO,DEPT1NAM,EMP1NO,EMP1FN,EMP1MI,EMP1LN,TYPE2,
 DEPT2NO,DEPT2NAM,EMP2NO,EMP2FN,EMP2MI,EMP2LN)
AS SELECT ALL
 D1.DEPTNO,D1.DEPTNAME,D1.MGRNO,D1.FIRSTNME,D1.MIDINIT,
 D1.LASTNAME, '2',
 D1.DEPTNO,D1.DEPTNAME,E2.EMPNO,E2.FIRSTNME,E2.MIDINIT,
 E2.LASTNAME
 FROM DSN8C10.VDEPMG1 D1, DSN8C10.EMP E2
 WHERE D1.DEPTNO = E2.WORKDEPT;
```

La figura siguiente muestra la sentencia de SQL que crea la vista denominada VPROJRE1.

```

CREATE VIEW DSN8C10.VPROJRE1
(PROJNO,PROJNAME,PROJDEP,RESPEMP,FIRSTNME,MIDINIT,
LASTNAME,MAJPROJ)
AS SELECT ALL
 PROJNO,PROJNAME,DEPTNO,EMPNO,FIRSTNME,MIDINIT,
 LASTNAME,MAJPROJ
 FROM DSN8C10.PROJ, DSN8C10.EMP
 WHERE RESPEMP = EMPNO ;

```

*Figura 81. VPROJRE1*

La sentencia de SQL siguiente crea la vista denominada VPSTRDE1.

```

CREATE VIEW DSN8C10.VPSTRDE1
(PROJ1NO,PROJ1NAME,RESP1NO,RESP1FN,RESP1MI,RESP1LN,
PROJ2NO,PROJ2NAME,RESP2NO,RESP2FN,RESP2MI,RESP2LN)
AS SELECT ALL
 P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
 P1.LASTNAME,
 P2.PROJNO,P2.PROJNAME,P2.RESPEMP,P2.FIRSTNME,P2.MIDINIT,
 P2.LASTNAME
 FROM DSN8C10.VPROJRE1 P1,
 DSN8C10.VPROJRE1 P2
 WHERE P1.PROJNO = P2.MAJPROJ ;

```

La sentencia de SQL siguiente crea la vista denominada VPSTRDE2.

```

CREATE VIEW DSN8C10.VPSTRDE2
(PROJ1NO,PROJ1NAME,RESP1NO,RESP1FN,RESP1MI,RESP1LN,
PROJ2NO,PROJ2NAME,RESP2NO,RESP2FN,RESP2MI,RESP2LN)
AS SELECT ALL
 P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
 P1.LASTNAME,
 P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
 P1.LASTNAME
 FROM DSN8C10.VPROJRE1 P1
 WHERE NOT EXISTS
 (SELECT * FROM DSN8C10.VPROJRE1 P2
 WHERE P1.PROJNO = P2.MAJPROJ) ;

```

La sentencia de SQL siguiente crea la vista denominada VFORPLA.

```

CREATE VIEW DSN8C10.VFORPLA
(PROJNO,PROJNAME,RESPEMP,PROJDEP,FRSTINIT,MIDINIT,lastname)
AS SELECT ALL
 F1.PROJNO,PROJNAME,RESPEMP,PROJDEP, SUBSTR(FIRSTNME, 1, 1),
 MIDINIT, lastname
 FROM DSN8C10.VPROJRE1 F1 LEFT OUTER JOIN DSN8C10.EMPPROJACT F2
 ON F1.PROJNO = F2.PROJNO;

```

La sentencia de SQL siguiente crea la vista denominada VSTAFAC1.

```

CREATE VIEW DSN8C10.VSTAFAC1
(PROJNO, ACTNO, ACTDESC, EMPNO, FIRSTNME, MIDINIT, LASTNAME,
EMPTIME,STDATETIME,ENDATE, TYPE)
AS SELECT ALL
 PA.PROJNO, PA.ACTNO, AC.ACTDESC,' ', ' ', ' ', ' ',
 PA.ACSTAFF, PA.ACSTDATETIME,
 PA.ACENDATE,'1'
 FROM DSN8C10.PROJECT PA, DSN8C10.ACT AC
 WHERE PA.ACTNO = AC.ACTNO ;

```

La sentencia de SQL siguiente crea la vista denominada VSTAFAC2.

```

CREATE VIEW DSN8C10.VSTAFAC2
(PROJNO, ACTNO, ACTDESC, EMPNO, FIRSTNME, MIDINIT, LASTNAME,
EMPTIME,STDATETIME,ENDATE, TYPE)
AS SELECT ALL
 EP.PROJNO, EP.ACTNO, AC.ACTDESC, EP.EMPNO, EM.FIRSTNME,
 EM.MIDINIT, EM.lastname, EP.EMPTIME, EP.EMSTDATETIME,
 EP.EMENDATE,'2'
 FROM DSN8C10.EMPPROJACT EP, DSN8C10.ACT AC, DSN8C10.EMP EM
 WHERE EP.ACTNO = AC.ACTNO AND EP.EMPNO = EM.EMPNO ;

```

La sentencia de SQL siguiente crea la vista denominada VPHONE.

```
CREATE VIEW DSN8C10.VPHONE
 (LASTNAME,
 FIRSTNAME,
 MIDDLEINITIAL,
 PHONENUMBER,
 EMPLOYEENUMBER,
 DEPTNUMBER,
 DEPTNAME)
AS SELECT ALL LASTNAME,
 FIRSTNAME,
 MIDINIT ,
 VALUE(PHONENO, ' '),
 EMPNO,
 DEPTNO,
 DEPTNAME
 FROM DSN8C10.EMP, DSN8C10.DEPT
 WHERE WORKDEPT = DEPTNO;
```

La sentencia de SQL siguiente crea la vista denominada VEMPLP.

```
CREATE VIEW DSN8C10.VEMPLP
 (EMPLOYEENUMBER,
 PHONENUMBER)
AS SELECT ALL EMPNO ,
 PHONENO
 FROM DSN8C10.EMP ;
```

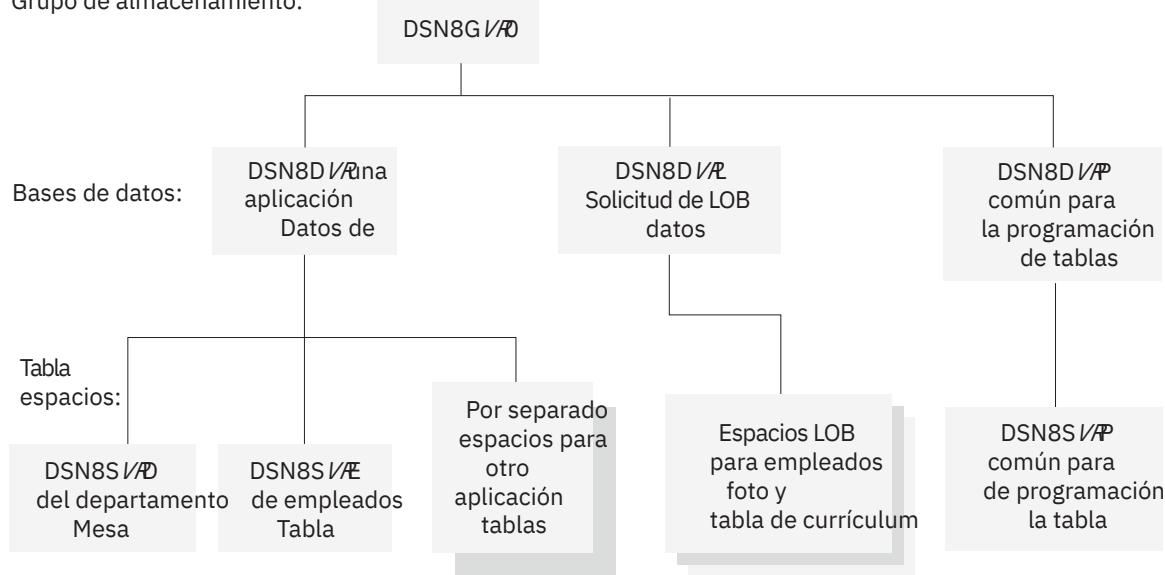
GUPI

## Almacenamiento de tablas de aplicaciones de ejemplo

Normalmente, los datos relacionados se almacenan en la misma base de datos.

La figura siguiente muestra cómo se relacionan las tablas de ejemplo con bases de datos y grupos de almacenamientos. Se utilizan dos bases de datos para ilustrar la posibilidad.

Grupo de almacenamiento:



V<sup>R</sup> es un identificador de versión de 2 dígitos.

Figura 82. Relación entre bases de datos y espacios de tablas de ejemplo

Además del grupo de almacenamiento y las bases de datos que se muestran en la figura anterior, el grupo de almacenamiento DSN8G12U y la base de datos DSN8D12U se crean al ejecutar DSNTEJ2A.

## Grupo de almacenamiento para datos de aplicaciones de ejemplo

Los datos de aplicaciones de ejemplo se almacenan en el grupo de almacenamiento DSN8G120. El grupo de almacenamiento predeterminado, SYSDEFLT, que se crea cuando se instala Db2, no se utiliza para almacenar datos de aplicación de ejemplo.



El grupo de almacenamiento que se utiliza para almacenar datos de aplicaciones de ejemplo se define con la sentencia siguiente:

```
CREATE STOGROUP DSN8G120
 VOLUMES (DSNV01)
 VCAT DSNC111;
```



## Bases de datos para datos de aplicaciones de ejemplo

Los datos de aplicaciones de ejemplo se almacenan en varias bases de datos. La base de datos predeterminada que se crea cuando se instala Db2 no se utiliza para almacenar los datos de aplicación de ejemplo.

DSN8D12P es la base de datos que se utiliza para las tablas que están relacionadas con programas. Las otras bases de datos se utilizan para tablas que están relacionadas con aplicaciones. Las bases de datos se definen mediante las sentencias siguientes:

```
CREATE DATABASE DSN8D12A
 STOGROUP DSN8G120
 BUFFERPOOL BP0
 CCSID EBCDIC;

CREATE DATABASE DSN8D12P
 STOGROUP DSN8G120
 BUFFERPOOL BP0
 CCSID EBCDIC;

CREATE DATABASE DSN8D12L
 STOGROUP DSN8G120
 BUFFERPOOL BP0
 CCSID EBCDIC;

CREATE DATABASE DSN8D12E
 STOGROUP DSN8G120
 BUFFERPOOL BP0
 CCSID UNICODE;

CREATE DATABASE DSN8D12U
 STOGROUP DSN8G12U
 CCSID EBCDIC;
```



## Espacios de tablas para datos de aplicaciones de ejemplo

Los espacios de tablas que no se definen explícitamente se crean implícitamente en la base de datos DSN8D12A, utilizando los atributos de espacio por omisión.



Las sentencias de SQL siguiente definen explícitamente una serie de espacios de tablas.

```
CREATE TABLESPACE DSN8S12D
 IN DSN8D12A
 USING STOGROUP DSN8G120
 PRIQTY 20
 SECQTY 20
 ERASE NO
 LOCKSIZE PAGE LOCKMAX SYSTEM
 BUFFERPOOL BP0
```

```
CLOSE NO
CCSID EBCDIC;

CREATE TABLESPACE DSN8S12E
IN DSN8D12A
USING STOGROUP DSN8G120
PRIQTY 20
SECQTY 20
ERASE NO
NUMPARTS 4
(PART 1 USING STOGROUP DSN8G120
PRIQTY 12
SECQTY 12,
PART 3 USING STOGROUP DSN8G120
PRIQTY 12
SECQTY 12)
LOCKSIZE PAGE LOCKMAX SYSTEM
BUFFERPOOL BP0
CLOSE NO
COMPRESS YES
CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S12B
IN DSN8D12L
USING STOGROUP DSN8G120
PRIQTY 20
SECQTY 20
ERASE NO
LOCKSIZE PAGE
LOCKMAX SYSTEM
BUFFERPOOL BP0
CLOSE NO
CCSID EBCDIC;
```

```
CREATE LOB TABLESPACE DSN8S12M
IN DSN8D12L
LOG NO;

CREATE LOB TABLESPACE DSN8S12L
IN DSN8D12L
LOG NO;

CREATE LOB TABLESPACE DSN8S12N
IN DSN8D12L
LOG NO;
```

```
CREATE TABLESPACE DSN8S12C
IN DSN8D12P
USING STOGROUP DSN8G120
PRIQTY 160
SECQTY 80
SEGSIZE 4
LOCKSIZE TABLE
BUFFERPOOL BP0
CLOSE NO
CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S12P
IN DSN8D12A
USING STOGROUP DSN8G120
PRIQTY 160
SECQTY 80
SEGSIZE 4
LOCKSIZE ROW
BUFFERPOOL BP0
CLOSE NO
CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S12R
IN DSN8D12A
USING STOGROUP DSN8G120
PRIQTY 20
SECQTY 20
ERASE NO
LOCKSIZE PAGE LOCKMAX SYSTEM
BUFFERPOOL BP0
```

```
CLOSE NO
CCSID EBCDIC;

CREATE TABLESPACE DSN8S12S
IN DSN8D12A
USING STOGROUP DSN8G120
PRIQTY 20
SECQTY 20
ERASE NO
LOCKSIZE PAGE LOCKMAX SYSTEM
BUFFERPOOL BP0
CLOSE NO
CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S81Q
IN DSN8D81P
USING STOGROUP DSN8G810
PRIQTY 160
SECQTY 80
SEGSIZE 4
LOCKSIZE PAGE
BUFFERPOOL BP0
CLOSE NO
CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S81U
IN DSN8D81E
USING STOGROUP DSN8G810
PRIQTY 5
SECQTY 5
ERASE NO
LOCKSIZE PAGE LOCKMAX SYSTEM
BUFFERPOOL BP0
CLOSE NO
CCSID UNICODE;
```

GUPI

## Tablas SYSDUMMYx

Db2 for z/OS proporciona un conjunto de tablas de catálogo SYSDUMMYx.

GUPI

El último carácter del nombre de tabla identifica el esquema de codificación de la siguiente manera:

- SYSIBM.SYSDUMMY1 utiliza el esquema de codificación EBCDIC.
- SYSIBM.SYSDUMMYE utiliza el esquema de codificación EBCDIC.
- SYSIBM.SYSDUMMYA utiliza el esquema de codificación ASCII.
- SYSIBM.SYSDUMMYU utiliza el esquema de codificación UNICODE.

Aunque las tablas SYSDUMMYx se implementan como tablas de catálogo, son similares a las tablas de ejemplo y se utilizan en algunos ejemplos de la documentación de Db2 for z/OS.

Puede utilizar cualquiera de las tablas SYSDUMMYx cuando tenga que escribir una consulta, pero no los datos de una tabla está referenciada. En cualquier consulta, debe especificar una referencia de tabla en la cláusula FROM, pero si la consulta no referencia a los datos de la tabla, no importa qué tabla está referenciada en la cláusula FROM. Cada una de las tablas SYSDUMMYx contiene una fila, por lo que la tabla SYSDUMMYx puede ser referenciada en una sentencia SELECT INTO sin necesidad de un predicado para limitar el resultado a una sola fila de datos.

Por ejemplo, cuando desee recuperar el valor de un registro especial o variable global, puede utilizar una consulta que haga referencia a una tabla SYSDUMMYx.

```
SELECT CURRENT PATH -- Retrieve the value of a special register
INTO :myvar
FROM SYSIBM.SYSDUMMY1;
SELECT TEMPORAL_LOGICAL_TRANSACTION_TIME -- Retrieve the value of a special register
```

```
INTO :myvar
FROM SYSIBM.SYSDUMMY1;
```

A veces, cuando Db2 for z/OS procesa una sentencia SQL, la sentencia se reescribe y se añade una referencia a una tabla SYSDUMMYx. Por ejemplo, algunas de las reescrituras internas que vuelven a añadir una referencia a una tabla SYSDUMMYx son para procesar la condición de búsqueda de un desencadenante, o las sentencias de control SQL PL IF, REPEAT, WHILE o RETURN. En estas situaciones, el conjunto de privilegios debe incluir el privilegio SELECT en la tabla SYSDUMMYx que está referenciada.

#### GUPI

#### Conceptos relacionados

[Objetos con CCSID diferentes en la misma sentencia SQL \(Db2 Internationalization Guide\)](#)

#### Tareas relacionadas

[Cómo impedir la conversión de caracteres de los localizadores de LOB](#)

En algunas situaciones, Db2 materializa el valor de LOB completo y lo convierte al esquema de codificación de una sentencia de SQL concreta. Este proceso adicional puede degradar el rendimiento y debe evitarse.

#### Referencia relacionada

[Tabla de catálogo SYSDUMMY1 \(Db2 SQL\)](#)

[Tabla de catálogo SYSDUMMYA \(Db2 SQL\)](#)

[Tabla de catálogo SYSDUMMYE \(Db2 SQL\)](#)

[Tabla de catálogo SYSDUMMYU \(Db2 SQL\)](#)

## Db2 programas de muestra de ayuda a la productividad

Db2 proporciona cuatro programas de muestra que muchos usuarios encuentran útiles como ayudas a la productividad. Estos programas se envían como código fuente, por lo que puede modificarlos para adaptarlos a sus necesidades.

#### DSNTIAUL

DSNTIAUL es un programa de ejemplo para descargar datos, como alternativa a la utilidad UNLOAD. DSNTIAUL está escrito en lenguaje ensamblador. DSNTIAUL puede descargar algunas o todas las filas de hasta 100 tablas de datos de SQL ( Db2 ). Con DSNTIAUL, puede descargar datos de cualquier tipo de datos integrado o distinto de Db2 . DSNTIAUL descarga las filas en un formulario que es compatible con la utilidad LOAD y genera sentencias de control de utilidad para LOAD. También puede utilizar DSNTIAUL para ejecutar cualquier instrucción SQL que no sea SELECT que pueda ejecutarse dinámicamente.

#### DSNTIAD

El programa de muestra DSNTIAD puede emitir cualquier instrucción SQL que pueda ejecutarse dinámicamente, excepto las instrucciones SELECT. DSNTIAD está escrito en lenguaje ensamblador.

#### DSNTEP2

DSNTEP2 es un programa SQL dinámico de muestra que puede emitir cualquier instrucción SQL que pueda ejecutarse dinámicamente. DSNTEP2 está escrito en lenguaje PL/I y está disponible en dos versiones: una versión fuente que puede modificar para satisfacer sus necesidades o una versión de código objeto que puede utilizar sin necesidad de un compilador PL/I.

#### DSNTEP4

El programa de muestra DSNTEP4 es idéntico a DSNTEP2, excepto que utiliza la captura de varias filas para aumentar el rendimiento. DSNTEP4 está escrito en lenguaje PL/I y está disponible en dos versiones: una versión fuente que puede modificar para satisfacer sus necesidades o una versión de código objeto que puede utilizar sin necesidad de un compilador PL/I.

Dado que estos cuatro programas también aceptan las sentencias SQL estáticas CONNECT, SET CONNECTION y RELEASE, puede utilizarlos para acceder a tablas de Db2 s en ubicaciones remotas.

## Preparación de los programas de muestra de ayuda a la productividad

DSNTIAUL y DSNTIAD se envían únicamente como código fuente, por lo que debe precompilarlos, ensamblarlos, vincularlos y enlazarlos antes de poder utilizarlos. Si desea utilizar la versión del código fuente de DSNTEP2 o DSNTEP4, debe precompilarla, compilarla, vincularla y enlazarla. Debe vincular la versión de código objeto de DSNTEP2 o DSNTEP4 antes de poder utilizarlas. Por lo general, un administrador del sistema prepara los programas como parte del proceso de instalación.

**Importante:** Envuelva siempre el paquete para el programa de muestra DSNTIAUL con la opción de envoltura REOPT(ALWAYS), y no especifique la opción de envoltura CONCENTRATESTMT(YES) para este paquete.

La siguiente tabla indica los trabajos de instalación que preparan cada programa de muestra. Todos los trabajos de instalación se encuentran en el conjunto de datos DSN1210.SDSNSAMP.

Tabla 176. Trabajos que preparan DSNTIAUL, DSNTIAD, DSNTEP2 y DSNTEP4

Nombre de programa	Trabajo de preparación del programa
DSNTIAUL	<a href="#">DSNTEJ2A</a>
DSNTIAD	<a href="#">DSNTIJTM</a>
DSNTEP2 (fuente)	<a href="#">DSNTEJ1P</a>
DSNTEP2 (objeto)	<a href="#">DSNTEJ1L</a>
DSNTEP4 (fuente)	<a href="#">DSNTEJ1P</a>
DSNTEP4 (objeto)	<a href="#">DSNTEJ1L</a>

## Compatibilidad de aplicaciones para los programas de muestra de ayuda a la productividad

Si está preparado para que los programas de muestra de ayuda a la productividad empiecen a utilizar nuevas capacidades de aplicación en niveles de función de e Db2 12 s posteriores, debe volver a vincular los paquetes para estos programas con la opción APPLCOMPAT correspondiente.

Los trabajos de Db2 12 para preparar los programas de muestra de ayuda a la productividad enlazan los paquetes con APPLCOMPAT(V12R1) de forma predeterminada, que equivale a APPLCOMPAT(V12R1M500).

Para obtener más información, consulte [Controlar el nivel de compatibilidad de la aplicación Db2 \(Db2 for z/OS ¿Qué hay de nuevo?\)](#) y [Niveles de función y niveles relacionados en Db2 12 \(Db2 for z/OS ¿Qué hay de nuevo?\)](#).

## Ejecución de los programas de muestra de ayuda a la productividad

Para ejecutar los programas de muestra, utilice el comando RUN (DSN). Para más información, consulte [RUN subcomando \(DSN\) \(Comandos de Db2\)](#) y los siguientes temas.

## Recuperación de datos Unicode de UTF-16 por DSNTEP2, DSNTEP4 y DSNTIAUL

Puede utilizar DSNTEP2, DSNTEP4 y DSNTIAUL para recuperar datos gráficos Unicode UTF-16. Sin embargo, es posible que estos programas no puedan mostrar algunos caracteres, si dichos caracteres no tienen correspondencia en el CCSID EBCDIC SBCS de destino.

### Referencia relacionada

[RUN subcomando \(DSN\) \(Comandos de Db2\)](#)

[Db2 for z/OS Cambio](#)

## Programa de ejemplo DSNTIAUL

Puede utilizar el programa DSNTIAUL para descargar datos de tablas de Db2 es en conjuntos de datos secuenciales. Los datos se copian en los conjuntos de datos y no se eliminan de la tabla.

DSNTIAUL es un programa de ejemplo para descargar datos, como alternativa a la utilidad UNLOAD. DSNTIAUL está escrito en lenguaje ensamblador. DSNTIAUL puede descargar algunas o todas las filas de hasta 100 tablas de datos de SQL ( Db2 ). Con DSNTIAUL, puede descargar datos de cualquier tipo de datos integrado o distinto de Db2 . DSNTIAUL descarga las filas en un formulario que es compatible con la utilidad LOAD y genera sentencias de control de utilidad para LOAD. También puede utilizar DSNTIAUL para ejecutar cualquier instrucción SQL que no sea SELECT que pueda ejecutarse dinámicamente.

Cuando se utiliza la recuperación de varias filas, el paralelismo puede desactivarse en el último grupo paralelo del bloque de consulta de nivel superior de una consulta. Para consultas muy sencillas, el paralelismo podría desactivarse para toda la consulta cuando se utiliza la recuperación de varias filas. Para obtener un paralelismo completo al ejecutar DSNTIAUL, cambie DSNTIAUL al modo de obtención de una sola fila especificando 1 para el parámetro *número de filas por obtención*.

DSNTIAUL utiliza SQL para acceder a Db2. Las operaciones en una tabla de control de acceso de nivel de fila o de nivel de columna están sujetas a las reglas especificadas para el control de acceso. Si la tabla tiene control de acceso a nivel de fila, DSNTIAUL recibe y devuelve solo las filas de la tabla que satisfacen los permisos de fila para el usuario. Si la tabla tiene control de acceso a nivel de columna, DSNTIAUL recibe y devuelve los valores en los valores de columna modificados por las máscaras de columna para el usuario.

**Importante:** Para evitar caracteres de sustitución en datos descargados, no utilice DSNTIAUL para descargar una tabla EBCDIC que contenga una columna Unicode.

### Preparación del programa de muestra DSNTIAUL

Antes de poder utilizar el programa de muestra DSNTIAUL, primero debe precompilarlo, ensamblarlo, enlazarlo y vincularlo. Además, cuando esté listo para que DSNTIAUL comience a utilizar nuevas capacidades en niveles de función posteriores de Db2 12 , debe volver a vincular los paquetes en el nivel APPLCOMPAT correspondiente.

**Importante:** Envuelva siempre el paquete para el programa de muestra DSNTIAUL con la opción de envoltura REOPT(ALWAYS), y no especifique la opción de envoltura CONCENTRATESTMT(YES) para este paquete.

Para obtener más información, consulte “[Db2 programas de muestra de ayuda a la productividad](#)” en la [página 1074](#).

### Ejecución del programa de muestra DSNTIAUL

Para ejecutar el programa de ejemplo DSNTIAUL, utilice el comando RUN (DSN) y especifique el siguiente módulo de carga y nombre de plan.

Nombre de módulo de carga	DSNTIAUL
Nombre de plan	DSNTIBC1

Para obtener más información sobre el comando RUN, consulte [RUN subcomando \(DSN\) \(Comandos de Db2\)](#).

### Parámetros DSNTIAUL

#### **PKGSET(colección)**

Especifica que DSNTIAUL ejecuta implícitamente una instrucción SET CURRENT PACKAGESET para asignar un valor al registro especial CURRENT PACKAGESET antes de procesar las instrucciones SQL dinámicas en SYSIN.

### **colección**

El valor que se asignará al registro especial CURRENT PACKAGESET. Puede especificar hasta 40 caracteres.

### **SQL**

Especifique SQL para indicar que su conjunto de datos de entrada contiene una o más sentencias SQL completas, cada una de las cuales termina con un punto y coma. Puede incluir cualquier instrucción SQL que pueda ejecutarse dinámicamente en su conjunto de datos de entrada. Además, puede incluir las sentencias SQL estáticas CONNECT, SET CONNECTION o RELEASE. Las sentencias SQL estáticas deben estar en mayúsculas.

DSNTIAUL utiliza las sentencias SELECT para determinar qué tablas descargar y ejecuta dinámicamente todas las demás sentencias excepto CONNECT, SET CONNECTION y RELEASE. DSNTIAUL ejecuta CONNECT, SET CONNECTION y RELEASE de forma estática para conectarse a ubicaciones remotas.

### **número de filas por búsqueda**

Especifique un número del 1 al 32767 para indicar el número de filas que DSNTIAUL recupera para cada operación SQL FETCH. Si no especifica este número, DSNTIAUL recupera 100 filas para cada FETCH. Este parámetro puede especificarse con el parámetro SQL.

Si también se especifica el parámetro LOBFILE y el conjunto de resultados de una operación FETCH puede contener valores LOB NULL, el *número de filas por obtención* debe ser 1.

### **AVISO LEGAL**

Especifique NO (el valor predeterminado) o SÍ para indicar si DSNTIAUL continúa recuperando filas después de recibir una advertencia SQL:

#### **(NO)**

Si se produce una advertencia cuando DSNTIAUL ejecuta un OPEN o FETCH para recuperar filas, DSNTIAUL deja de recuperar filas. Si la marca SQLWARN1, SQLWARN2, SQLWARN6 o SQLWARN7 está activada cuando DSNTIAUL ejecuta un FETCH para recuperar filas, DSNTIAUL continúa recuperando filas.

#### **(SÍ)**

Si se produce una advertencia cuando DSNTIAUL ejecuta un OPEN o FETCH para recuperar filas, DSNTIAUL continúa recuperando filas.

#### **(SILENCIO)**

Igual que SÍ, excepto que el programa suprime todos los mensajes de advertencia SQL de las sentencias OPEN o FETCH si el SQLCODE es 0 o mayor.

### **LOBFILE(prefijo)**

Especifique LOBFILE para indicar que desea que DSNTIAUL asigne dinámicamente conjuntos de datos, cada uno para recibir el contenido completo de una celda LOB. (Una celda LOB es la intersección de una fila y una columna LOB) Si no especifica la opción LOBFILE, solo podrá descargar hasta 32 KB de datos de una columna LOB.

#### **prefijo**

Especifique un calificador de alto nivel para estos conjuntos de datos asignados dinámicamente. Puede especificar hasta 17 caracteres. El calificador debe cumplir con las reglas para los nombres de conjuntos de datos TSO.

DSNTIAUL utiliza una convención de nomenclatura para estos conjuntos de datos asignados dinámicamente de *prefix.Qiiiiiii.Cjjjjjjj.Rkkkkkkk*, donde estos calificadores tienen los siguientes valores:

#### **prefijo**

El calificador de alto nivel que especifique en la opción LOBFILE.

#### **Qiiiiiii**

El número de secuencia (a partir de 0) de una consulta que devuelve una o más columnas LOB

#### **Cjjjjjjj**

El número de secuencia (empezando por 0) de una columna en una consulta que devuelve una o más columnas LOB

## **Rkkkkkkk**

El número de secuencia (a partir de 0) de una fila de un conjunto de resultados que tiene una o más columnas LOB.

La sentencia LOAD generada contiene variables de referencia de archivos LOB que pueden utilizarse para cargar datos de estos conjuntos de datos asignados dinámicamente.

Si no especifica el parámetro SQL, su conjunto de datos de entrada debe contener una o más sentencias de una sola línea (sin punto y coma) que utilicen la siguiente sintaxis:

```
table or view name [WHERE conditions] [ORDER BY columns]
```

Cada instrucción de entrada debe ser una instrucción SQL SELECT válida con la cláusula SELECT \* FROM omitida y sin punto y coma final. DSNTIAUL genera una instrucción SELECT para cada instrucción de entrada añadiendo su línea de entrada a SELECT \* FROM, y luego utiliza el resultado para determinar qué tablas descargar. Para este formato de entrada, el texto de cada especificación de tabla puede tener un máximo de 72 bytes y no debe abarcar varias líneas.

Puede utilizar las sentencias de entrada para especificar sentencias SELECT que unen dos o más tablas o seleccionar columnas específicas de una tabla. Si especifica columnas, debe modificar la instrucción LOAD que genera DSNTIAUL.

## **Conjuntos de datos DSNTIAUL**

### **Conjunto de datos**

#### **Descripción**

#### **SYSIN**

Conjunto de datos de entrada.

Si especifica el parámetro SQL, puede introducir comentarios entre corchetes en la entrada DSNTIAUL que incluye sentencias SQL dinámicas. Los comentarios entre corchetes no son compatibles si la entrada incluye las sentencias SQL estáticas CONNECT, SET CONNECTION o RELEASE. Los comentarios entre corchetes comienzan con /\* y terminan con \*/.

La longitud de registro para el conjunto de datos de entrada debe ser de al menos 72 bytes. DSNTIAUL lee solo los primeros 72 bytes de cada registro.

#### **SYSPRINT**

Conjunto de datos de salida. DSNTIAUL escribe mensajes informativos y de error en este conjunto de datos.

La longitud de registro para el conjunto de datos SYSPRINT es de 121 bytes.

#### **SYSPUNCH**

Conjunto de datos de salida. DSNTIAUL escribe las sentencias de control de la utilidad LOAD en este conjunto de datos.

#### **SYSRECnn**

Conjuntos de datos de salida. El valor nn va de 00 a 99. Puede tener un máximo de 100 conjuntos de datos de salida para una sola ejecución de DSNTIAUL. Cada conjunto de datos contiene los datos que se descargan cuando DSNTIAUL procesa una instrucción SELECT del conjunto de datos de entrada.

Por lo tanto, el número de conjuntos de datos de salida debe coincidir con el número de instrucciones SELECT (si especifica el parámetro SQL) o las especificaciones de la tabla en su conjunto de datos de entrada.

Definir todos los conjuntos de datos como conjuntos de datos secuenciales. Puede especificar la longitud de registro y el tamaño de bloque de los conjuntos de datos SYSPUNCH y SYSRECnn. La longitud máxima de registro para los conjuntos de datos SYSPUNCH y SYSRECnn es de 32760 bytes.

## Códigos de devolución DSNTIAUL

Tabla 177. Códigos de devolución DSNTIAUL

Código de retorno	Significado
0	Realización satisfactoria.
4	<p>Una instrucción SQL recibió un código de advertencia.</p> <ul style="list-style-type: none"><li>Si se especifica TOLWARN(YES) y la advertencia se produjo en un FETCH o OPEN durante el procesamiento de una sentencia SELECT, Db2 realiza la operación de descarga.</li><li>De lo contrario, si la instrucción SQL era una instrucción SELECT, Db2 no realizaba la operación de descarga asociada.</li></ul> <p>Si Db2 devuelve un +394, lo que indica que está utilizando sugerencias de optimización, o un +395, lo que indica una o más sugerencias de optimización no válidas, Db2 realiza la operación de descarga.</p>
8	Una instrucción SQL ha recibido un código de error. Si la instrucción SQL era una instrucción SELECT, Db2 no realizó la operación de descarga asociada o no la completó.
12	DSNTIAUL no pudo abrir un conjunto de datos, una instrucción SQL devolvió un código de error grave (-144, -302, -804, -805, -818, -902, -906, -911, -913, -922, -923, -924 o -927), o se produjo un error en la rutina de formato de mensajes SQL.

### ejemplos

#### Ejemplo: uso de DSNTIAUL para descargar un subconjunto de filas de una tabla

Supongamos que desea descargar las filas del departamento de D01 de la tabla del proyecto. Como puede ajustar la especificación de la tabla en una línea y no desea ejecutar ninguna instrucción que no sea SELECT, no necesita el parámetro SQL. Su invocación se parece a la que se muestra en la siguiente figura:

```
//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAUL) PLAN(DSNTIBC1) -
 LIB('DSN1210.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DSN=DSN8UNLD.SYSREC00,
// UNIT=SYSDA,SPACE=(32760,(1000,500)),DISP=(,CATLG),
// VOL=SER=SCR03
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
// UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
// VOL=SER=SCR03,RECFM=FB,LRECL=120,BLKSIZE=1200
//SYSIN DD *
DSN8C10.PROJ WHERE DEPTNO='D01'
```

#### Ejemplo: uso de DSNTIAUL para descargar filas en más de una tabla

Supongamos que también desea utilizar DSNTIAUL para realizar las siguientes acciones:

- Descargar todas las filas de la tabla del proyecto
- Descargar solo las filas de la tabla de empleados para los empleados de departamentos con números de departamento que comiencen por D, y ordenar las filas descargadas por número de empleado
- Bloquee ambas mesas en modo compartido antes de descargarlas
- Recuperar 250 filas por recuperación

Para estas actividades, debe especificar el parámetro SQL y el parámetro *número de filas por obtención* cuando ejecute DSNTIAUL. Su invocación DSNTIAUL se muestra en la siguiente figura:

```
//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAUL) PLAN(DSNTIBC1) PARMS('SQL,250') -
 LIB('DSN1210.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DSN=DSN8UNLD.SYSREC00,
 // UNIT=SYSDA,SPACE=(32760,(1000,500)),DISP=(,CATLG),
 // VOL=SER=SCR03
//SYSREC01 DD DSN=DSN8UNLD.SYSREC01,
 // UNIT=SYSDA,SPACE=(32760,(1000,500)),DISP=(,CATLG),
 // VOL=SER=SCR03
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
 // UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
 // VOL=SER=SCR03,RECFM=FB,LRECL=120,BLKSIZE=1200
//SYSIN DD *
LOCK TABLE DSN8C10.EMP IN SHARE MODE;
LOCK TABLE DSN8C10.PROJ IN SHARE MODE;
SELECT * FROM DSN8C10.PROJ;
SELECT * FROM DSN8C10.EMP
 WHERE WORKDEPT LIKE 'D%'
 ORDER BY EMPNO;
```

### Ejemplo: uso de DSNTIAUL para obtener instrucciones de control de la utilidad LOAD

Si desea obtener las sentencias de control de la utilidad LOAD para cargar filas en una tabla, pero no desea descargar las filas, puede establecer los nombres de los conjuntos de datos para *los conjuntos de datos SYSRECnn* en DUMMY. Por ejemplo, para obtener las sentencias de control de utilidad para cargar filas en la tabla de departamentos, invoque DSNTIAUL como se muestra en la siguiente figura:

```
//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAUL) PLAN(DSNTIBC1) -
 LIB('DSN1210.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DUMMY
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
 // UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
 // VOL=SER=SCR03,RECFM=FB,LRECL=120,BLKSIZE=1200
//SYSIN DD *
DSN8C10.DEPT
```

### Ejemplo: uso de DSNTIAUL para descargar datos LOB

Este ejemplo utiliza la tabla LOB de muestra con la siguiente estructura:

```
CREATE TABLE DSN8C10.EMP_PHOTO_RESUME
(EMPNO CHAR(06) NOT NULL,
 EMP_ROWID ROWID NOT NULL GENERATED ALWAYS,
 PSEG_PHOTO BLOB(500K),
 BMP_PHOTO BLOB(100K),
 RESUME CLOB(5K),
 PRIMARY KEY (EMPNO))
IN DSN8D12L.DSN8S12B
CCSID EBCDIC;
```

La siguiente llamada a DSNTIAUL descarga la tabla LOB de muestra. Los parámetros para DSNTIAUL indican las siguientes opciones:

- El parámetro SQL especifica que el conjunto de datos de entrada (SYSIN) contiene SQL.
- El valor del parámetro *número de filas por recuperación* de 1 especifica que DSNTIAUL debe recuperar una fila por cada operación FETCH. Es necesario un valor de 1 si las columnas LOB que descarga pueden contener valores NULL.

- El valor del parámetro LOBFILE de LOBFILE( DSN8UNLD ) especifica que DSNTIAUL coloca los datos LOB en conjuntos de datos con un calificador de alto nivel de DSN8UNLD.

```
//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB91) -
 PARMS('SQL,1,LOBFILE(DSN8UNLD)') -
 LIB('DSN1210.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DSN=DSN8UNLD.SYSREC00,
// UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
// VOL=SER=SCR03,RECFM=FB
//SYSPUNCH DD DSN=DSN8UNLD.SYSPUNCH,
// UNIT=SYSDA,SPACE=(800,(15,15)),DISP=(,CATLG),
// VOL=SER=SCR03,RECFM=FB
//SYSIN DD *
 SELECT * FROM DSN8C10.EMP_PHOTO_RESUME;
```

Dado que la tabla LOB de muestra tiene 4 filas de datos, DSNTIAUL produce el siguiente resultado:

- Los datos de las columnas EMPNO y EMP\_ROWID se colocan en el conjunto de datos que se asigna de acuerdo con la sentencia DD de la base de datos ( SYSREC00 ). El nombre del conjunto de datos es DSN8UNLD.SYSREC00
- Se coloca una sentencia LOAD generada en el conjunto de datos que se asigna de acuerdo con la sentencia DD de SYSPUNCH. El nombre del conjunto de datos es DSN8UNLD.SYSPUNCH
- Los siguientes conjuntos de datos se crean dinámicamente para almacenar datos LOB:
  - DSN8UNLD.Q0000000.C0000002.R0000000
  - DSN8UNLD.Q0000000.C0000002.R0000001
  - DSN8UNLD.Q0000000.C0000002.R0000002
  - DSN8UNLD.Q0000000.C0000002.R0000003
  - DSN8UNLD.Q0000000.C0000003.R0000000
  - DSN8UNLD.Q0000000.C0000003.R0000001
  - DSN8UNLD.Q0000000.C0000003.R0000002
  - DSN8UNLD.Q0000000.C0000003.R0000003
  - DSN8UNLD.Q0000000.C0000004.R0000000
  - DSN8UNLD.Q0000000.C0000004.R0000001
  - DSN8UNLD.Q0000000.C0000004.R0000002
  - DSN8UNLD.Q0000000.C0000004.R0000003

Por ejemplo, DSN8UNLD.Q0000000.C0000004.R0000001 significa que el conjunto de datos contiene datos que se descargan de la segunda fila ( R0000001 ) y la quinta columna ( C0000004 ) del conjunto de resultados de la primera consulta ( Q0000000 ).

## Programa de ejemplo DSNTIAD

Puede utilizar el programa DSNTIAD para ejecutar sentencias de SQL dinámico que no sean sentencias SELECT.

El programa de muestra DSNTIAD puede emitir cualquier instrucción SQL que pueda ejecutarse dinámicamente, excepto las instrucciones SELECT. DSNTIAD está escrito en lenguaje ensamblador.

### Preparación del programa de muestra DSNTIAD

Antes de poder utilizar el programa de muestra DSNTIAD, primero debe precompilarlo, ensamblarlo, enlazarlo y vincularlo. Además, cuando esté listo para que DSNTIAD comience a utilizar nuevas

capacidades en niveles de función posteriores de Db2 12 , debe volver a vincular los paquetes en el nivel APPLCOMPAT correspondiente.

Para obtener más información, consulte “[Db2 programas de muestra de ayuda a la productividad](#)” en la [página 1074](#)

## Ejecución del programa de muestra DSNTIAD

Para ejecutar el programa de muestra DSNTIAD, utilice el comando RUN (DSN) y especifique el siguiente módulo de carga y nombre de plan.

Nombre de módulo de carga	DSNTIAD
Nombre de plan	DSNTIAC1

Para obtener más información sobre el comando RUN, consulte [RUN subcomando \(DSN\) \(Comandos de Db2 \)](#).

## Parámetros DSNTIAD

### **PKGSET(*colección*)**

Especifica que DSNTIAD ejecuta implícitamente una instrucción SET CURRENT PACKAGESET para asignar un valor al registro especial CURRENT PACKAGESET antes de procesar las instrucciones SQL dinámicas en SYSIN.

#### ***colección***

El valor que se asignará al registro especial CURRENT PACKAGESET. Puede especificar hasta 40 caracteres.

### **RC0**

Si especifica este parámetro, DSNTIAD finaliza con el código de retorno 0, incluso si el programa encuentra errores SQL. Si no se especifica RC0, DSNTIAD finaliza con un código de retorno que refleja la gravedad de los errores que se producen. Sin RC0, DSNTIAD finaliza si se producen más de 10 errores SQL durante una sola ejecución.

### **SQL TERM(*termchar*)**

Especifique este parámetro para indicar el carácter que utiliza para finalizar cada instrucción SQL. Puede utilizar cualquier carácter especial **excepto** uno de los que figuran en la siguiente tabla. SQLTERM(;) es el valor predeterminado.

*Tabla 178. Caracteres especiales no válidos para el terminador SQL*

Nombre	Carácter	Representación hexadecimal
en blanco		X'40'
coma	,	X'6B'
comillas dobles	"	X'7F'
abrir paréntesis	(	X'4D'
cerrar paréntesis	)	X'5D'
comillas simples	'	X'7D'
subrayado	_	X'6D'

Utilice un carácter que no sea punto y coma si tiene previsto ejecutar una instrucción que contenga puntos y comas incrustados.

Por ejemplo, supongamos que especifica el parámetro SQLTERM(#) para indicar que el carácter # es el terminador de la instrucción. Entonces, una declaración CREATE TRIGGER con punto y coma incrustado tiene este aspecto:

```
CREATE TRIGGER NEW_HIRE
 AFTER INSERT ON EMP
 FOR EACH ROW MODE DB2SQL
 BEGIN ATOMIC
 UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
 END#
```

Una instrucción CREATE PROCEDURE con punto y coma incrustado tiene el siguiente aspecto:

```
CREATE PROCEDURE PROC1 (IN PARM1 INT, OUT SCODE INT)
 LANGUAGE SQL
 BEGIN
 DECLARE SQLCODE INT;
 DECLARE EXIT HANDLER FOR SQLEXCEPTION
 SET SCODE = SQLCODE;
 UPDATE TBL1 SET COL1 = PARM1;
 END #
```

Tenga cuidado de elegir un carácter para el terminador de la instrucción que no se utilice dentro de la instrucción.

## Conjuntos de datos DSNTIAD

### Conjunto de datos

#### Descripción

#### SYSIN

Conjunto de datos de entrada. En este conjunto de datos, puede introducir cualquier número de sentencias SQL no SELECT. Cada instrucción SQL debe terminar con el carácter de terminación SQL. Si especifica el parámetro SQLTERM(*termchar*), *termchar* es el carácter de terminación SQL. De lo contrario, el carácter de terminación SQL es un punto y coma. Una declaración puede abarcar varias líneas, pero DSNTIAD solo lee los primeros 72 bytes de cada línea.

Los comentarios en la entrada DSNTIAD no son compatibles.

#### SYSPRINT

Conjunto de datos de salida. DSNTIAD escribe mensajes informativos y de error en este conjunto de datos. DSNTIAD establece la longitud de registro de este conjunto de datos en 121 bytes y el tamaño de bloque en 1210 bytes.

Definir todos los conjuntos de datos como conjuntos de datos secuenciales.

## Códigos de devolución DSNTIAD

Tabla 179. Códigos de devolución DSNTIAD

Código de retorno	Significado
0	Finalización con éxito, o el parámetro especificado por el usuario RCO.
4	Una instrucción SQL recibió un código de advertencia.
8	Una instrucción SQL ha recibido un código de error.
12	DSNTIAD no pudo abrir un conjunto de datos, la longitud de una instrucción SQL era superior a 2 MB, una instrucción SQL devolvió un código de error grave ((-8 nn o -9nn ), o se produjo un error en la rutina de formato de mensajes SQL.

## Ejemplo: invocar el programa DSNTIAD

Supongamos que desea ejecutar 20 sentencias UPDATE y no quiere que DSNTIAD finalice si se producen más de 10 errores. Su invocación se parece a la que se muestra en la siguiente figura:

```
//RUNTIAD EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIAC1) PARMS('RC0') -
 LIB('DSN1210.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
UPDATE DSN8C10.PROJ SET DEPTNO='J01' WHERE DEPTNO='A01';
UPDATE DSN8C10.PROJ SET DEPTNO='J02' WHERE DEPTNO='A02';
:
UPDATE DSN8C10.PROJ SET DEPTNO='J20' WHERE DEPTNO='A20';
```

## Programas de ejemplo DSNTEP2 y DSNTEP4

Puede utilizar los programas DSNTEP2 o DSNTEP4 para ejecutar sentencias SQL de forma dinámica.

DSNTEP2 es un programa SQL dinámico de muestra que puede emitir cualquier instrucción SQL que pueda ejecutarse dinámicamente. DSNTEP2 está escrito en lenguaje PL/I y está disponible en dos versiones: una versión fuente que puede modificar para satisfacer sus necesidades o una versión de código objeto que puede utilizar sin necesidad de un compilador PL/I.

El programa de muestra DSNTEP4 es idéntico a DSNTEP2, excepto que utiliza la captura de varias filas para aumentar el rendimiento. DSNTEP4 está escrito en lenguaje PL/I y está disponible en dos versiones: una versión fuente que puede modificar para satisfacer sus necesidades o una versión de código objeto que puede utilizar sin necesidad de un compilador PL/I.

Cuando se utiliza la recuperación de varias filas, el paralelismo puede desactivarse para el último grupo paralelo en el bloque de consulta de nivel superior, o desactivarse por completo para consultas muy simples. Para obtener un paralelismo completo, utilice DSNTEP2 o especifique la opción de control SET MULT\_FETCH 1 para DSNTEP4.

DSNTEP2 y DSNTEP4 escriben sus resultados en el conjunto de datos definido por la sentencia DD de SYSPRINT. Los datos SYSPRINT deben tener una longitud de registro lógico que coincida con el valor PAGEWIDTH en el programa fuente DSNTEP2 o DSNTEP4. Si utiliza la versión original del programa que se envía con Db2, la longitud del registro lógico es de 133 bytes. Si los datos SYSPRINT no tienen la misma longitud de registro lógico que el valor PAGEWIDTH, el programa emite el código de retorno 12 con abend U4038 y el código de razón 1. Este error se produce debido al error de excepción de archivo PL/I IBM0201S ONCODE=81. Se emite el siguiente mensaje de error:

La condición UNDEFINEDFILE se planteó debido a un conflicto DECLARE y atributos OPEN (FILE= SYSPRINT).

Si utiliza aplicaciones u otros sistemas de automatización para procesar los resultados de DSNTEP2 o DSNTEP4, tenga en cuenta que pueden producirse pequeños cambios en el formato como resultado de mejoras o del servicio. Dichos cambios podrían requerir que usted ajuste sus procesos que utilizan el resultado de estos programas.

**Importante:** Cuando asigne un nuevo conjunto de datos con la sentencia SYSPRINT DD, especifique un DCB con RECFM=FBA y LRECL=nnn, donde nnn coincide con el valor PAGEWIDTH en el programa fuente, o no especifique el parámetro DCB.

## Preparación de los programas de muestra DSNTEP2 y DSNTEP4

Antes de poder utilizar los programas de muestra DSNTEP2 y DSNTEP4, debe prepararlos. Si utiliza las versiones de código fuente de DSNTEP2 o DSNTEP4, primero debe precompilarlas, compilarlas, vincularlas y enlazarlas. Si utiliza las versiones de código objeto de DSNTEP2 o DSNTEP4, primero debe vincularlas.

Además, cuando esté listo para que DSNTEP2 o DSNTEP4 empiecen a utilizar nuevas capacidades en niveles de función posteriores de Db2 12 , debe volver a vincular los paquetes en el nivel APPLCOMPAT correspondiente.

Para más información, consulte "Preparación de los programas de muestra de ayuda a la productividad" en "["Db2 programas de muestra de ayuda a la productividad"](#)" en la página 1074.

## Ejecutar los programas de muestra DSNTEP2 y DSNTEP4

Para ejecutar los programas de muestra DSNTEP2 y DSNTEP4, utilice el comando RUN (DSN) y especifique el siguiente módulo de carga y nombre de plan.

### DSNTEP2 cargar módulo y plan

Módulo de carga	DSNTEP2
Plan	DSNTEPC1

### DSNTEP4 cargar módulo y plan

Módulo de carga	DSNTEP4
Plan	DSNTP412

Para obtener más información sobre el comando RUN, consulte [RUN subcomando \(DSN\) \(Comandos de Db2 \)](#).

## DSNTEP2 y parámetros de DSNTEP4

### PKGSET(*colección*)

Especifica que DSNTEP2 o DSNTEP4 ejecutan implícitamente una instrucción SET CURRENT PACKAGESET para asignar un valor al registro especial CURRENT PACKAGESET antes de procesar las instrucciones SQL dinámicas en SYSIN.

### *colección*

El valor que se asignará al registro especial CURRENT PACKAGESET. Puede especificar hasta 40 caracteres.

Por ejemplo, consulte: "[Ejemplo: Ejecutar sentencias SQL dinámicas en diferentes niveles de compatibilidad de aplicaciones en el mismo SYSIN](#)" en la página 1090.

### ALIGN(MID) o ALIGN(LHS)

Especifica la alineación.

### ALINEAR (MEDIO)

Especifica que la salida DSNTEP2 o DSNTEP4 debe estar centrada. [ALIGN \(MID\)](#) es el valor predeterminado.

### ALINEAR (IZQ)

Especifica que la salida DSNTEP2 o DSNTEP4 debe estar justificada a la izquierda.

### NOMIXED o MIXED

Especifica si DSNTEP2 o DSNTEP4 contienen caracteres DBCS.

### NOMIXED

Especifica que la entrada DSNTEP2 o DSNTEP4 no contiene caracteres DBCS. [NOMIXED](#) es el valor predeterminado.

### MIXED

Especifica que la entrada DSNTEP2 o DSNTEP4 contiene algunos caracteres DBCS.

### PRECAUCIÓN

Especifica que DSNTEP2 o DSNTEP4 debe mostrar detalles sobre cualquier advertencia SQL que se encuentre en el momento de PREPARAR.

Independientemente de si especifica PREPWARN, cuando se encuentra una advertencia SQL en el momento de PREPARAR, el programa muestra el mensaje "SQLWARNING ON PREPARE" y establece el código de retorno en 4. Cuando especifique PREPWARN, el programa también mostrará los detalles sobre cualquier advertencia SQL.

#### **SQLFORMAT**

Especifica cómo DSNTEP2 o DSNTEP4 preprocesan las sentencias SQL antes de pasárlas a Db2. Seleccione una de las siguientes opciones:

##### **SQL**

Este es el modo preferido para las sentencias SQL que no sean de lenguaje de procedimiento SQL. Cuando se utiliza esta opción, que es la predeterminada, DSNTEP2 o DSNTEP4 contraen cada línea de una instrucción SQL en una sola línea antes de pasar la instrucción a Db2. DSNTEP2 o DSNTEP4 también descarta todos los comentarios SQL.

##### **SQLCOMNT**

Este modo es adecuado para todos los SQL, pero está pensado principalmente para el procesamiento de lenguaje procedural SQL. Cuando esta opción está activa, el comportamiento es similar al modo SQL, excepto que DSNTEP2 o DSNTEP4 no descartan los comentarios SQL. En su lugar, termina automáticamente cada comentario SQL con un carácter de avance de línea (hex 25), a menos que el comentario ya esté terminado por uno o más caracteres de formato de línea. Utilice esta opción para procesar el lenguaje de procedimientos SQL con una modificación mínima por DSNTEP2 o DSNTEP4.

##### **SQLPL**

Este modo es adecuado para todos los SQL, pero está pensado principalmente para el procesamiento de lenguaje procedural SQL. Cuando esta opción está activa, DSNTEP2 o DSNTEP4 retienen los comentarios SQL y terminan cada línea de una instrucción SQL con un carácter de avance de línea (hex 25) antes de pasar la instrucción a Db2. Las líneas que terminan con un token de división no terminan con un carácter de avance de línea. Utilice este modo para obtener mejores diagnósticos y depuración del lenguaje de procedimientos SQL.

#### **SQLTER M(*termchar*)**

Especifica el carácter que se utiliza para finalizar cada instrucción SQL. Puede utilizar cualquier carácter, excepto uno de los que figuran en [Tabla 178 en la página 1082](#). **SQLTERM( ; )** es el valor predeterminado.

Utilice un carácter que no sea punto y coma si tiene previsto ejecutar una instrucción que contenga puntos y comas incrustados.

Consulte “[Ejemplo: cambiar el terminador SQL](#)” en la página 1091.

#### **AVISO LEGAL**

Indica si DSNTEP2 o DSNTEP4 continúan procesando instrucciones SQL SELECT después de recibir una advertencia SQL. Puede especificar uno de los valores siguientes:

##### **NEE**

Indica que el programa deja de procesar la instrucción SELECT si se produce una advertencia cuando el programa ejecuta un OPEN o FETCH para una instrucción SELECT. NO es el valor predeterminado para TOLWARN.

Existen las siguientes excepciones:

- Si se produce SQLCODE +445 o SQLCODE +595 cuando DSNTEP2 o DSNTEP4 ejecuta un FETCH para una sentencia SELECT, el programa continúa procesando la sentencia SELECT.
- Si se produce SQLCODE +354 cuando DSNTEP4 ejecuta un FETCH para una sentencia SELECT, el programa continúa procesando la sentencia SELECT.
- Si se produce SQLCODE +802 cuando DSNTEP2 o DSNTEP4 ejecuta un FETCH para una sentencia SELECT, el programa continúa procesando la sentencia SELECT si la sentencia de control TOLARTHWRN se establece en YES.

**YES**

Indica que el programa continúa procesando la instrucción SELECT si se produce una advertencia cuando el programa ejecuta un OPEN o FETCH para una instrucción SELECT.

**quiet**

Igual que SÍ, excepto que el programa suprime todos los mensajes de advertencia SQL de las sentencias OPEN o FETCH si el SQLCODE es 0 o mayor.

## Configuración que puede cambiar en los programas fuente DSNTEP2 o DSNTEP4 para modificar el formato de salida

Si el formato de salida DSNTEP2 o DSNTEP4 no cumple con sus requisitos, puede cambiar algunas variables en el código fuente para modificar el formato.

Por ejemplo, si desea aumentar el ancho máximo de una página, el tamaño máximo de una línea de impresión y el número máximo de caracteres en la salida de una columna de caracteres, puede aumentar los valores de las variables PAGEWIDTH, MAXPAGWD y MAXCOLWD.

Para una descripción completa de los ajustes que puede cambiar, consulte la información en TAMAÑOS DE PROGRAMA en los prólogos de DSNTEP2 y DSNTEP4.

## DSNTEP2 y conjuntos de datos de la Oficina de Estadísticas Laborales ( DSNTEP4 )

DSNTEP2 y DSNTEP4:

**SYSIN**

Conjunto de datos de entrada. En este conjunto de datos, puede introducir cualquier número de sentencias SQL, cada una terminada con un punto y coma. Una declaración puede abarcar varias líneas, pero DSNTEP2 o DSNTEP4 solo lee los primeros 72 bytes de cada línea. Debe confirmar explícitamente todas las sentencias SQL excepto la última.

Puede introducir comentarios en DSNTEP2 o DSNTEP4 introduciendo un asterisco (\*) en la columna 1 o dos guiones (--) en cualquier lugar de una línea. El texto que sigue al asterisco se considera texto de comentario. El texto que sigue a dos guiones puede ser texto de comentario o una instrucción de control. Los comentarios no se tienen en cuenta en el almacenamiento en caché dinámico de sentencias. Los comentarios y las instrucciones de control no pueden abarcar varias líneas.

Puede introducir instrucciones de control de la siguiente forma en el conjunto de datos de entrada DSNTEP2 y DSNTEP4 :

```
--#SET control-option value
```

Puede especificar las siguientes declaraciones de opciones de control. Si especifica un valor de NO para cualquiera de las opciones de esta lista, el programa se comporta como si no hubiera especificado el parámetro.

**--#SET PKGSET valor**

Especifica que DSNTEP2 o DSNTEP4 ejecuta implícitamente una instrucción SET CURRENT PACKAGESET para asignar un valor al registro especial CURRENT PACKAGESET antes de procesar instrucciones SQL dinámicas después de esta instrucción de control en SYSIN. *es el valor* que se asignará al registro especial CURRENT PACKAGESET. Puede especificar hasta 40 caracteres.

Por ejemplo, consulte: [“Ejemplo: Ejecutar sentencias SQL dinámicas en diferentes niveles de compatibilidad de aplicaciones en el mismo SYSIN”](#) en la página 1090.

**--#SET TERMINATOR value**

El terminador de la instrucción SQL. *es* cualquier carácter de un solo byte que no sea uno de los que figuran en [“Programa de ejemplo DSNTIAD”](#) en la página 1081. El valor predeterminado es el valor del parámetro SQLTERM.

Consulte [“Ejemplo: cambiar el terminador SQL dentro de una serie de sentencias SQL”](#) en la página 1091.

**--#SET ROWS\_FETCH value**

El número de filas que se van a obtener de la tabla de resultados. es un literal numérico entre -1 y el número de filas de la tabla de resultados. -1 significa que se deben recuperar todas las filas. El valor por omisión es -1.

**--#SET ROWS\_OUT valor**

El número de filas recuperadas que se enviarán al conjunto de datos de salida. es un literal numérico entre -1 y el número de filas obtenidas. -1 significa que todas las filas obtenidas deben enviarse al conjunto de datos de salida. El valor por omisión es -1.

**--#SET MULT\_FETCH value**

Esta opción solo es válida para DSNTEP4. Utilice MULT\_FETCH para especificar el número de filas que se van a recuperar a la vez de la tabla de resultados. La cantidad de recuperación predeterminada para DSNTEP4 es de 100 filas, pero puede especificar de 1 a 32676 filas.

**--#SET TOLWARN valor**

Indica si DSNTEP2 o DSNTEP4 continúan procesando instrucciones SQL SELECT después de recibir una advertencia SQL. Puede especificar uno de los valores siguientes:

**NEE**

Indica que el programa deja de procesar la instrucción SELECT si se produce una advertencia cuando el programa ejecuta un OPEN o FETCH para una instrucción SELECT. NO es el valor predeterminado para TOLWARN.

Existen las siguientes excepciones:

- Si se produce SQLCODE +445 o SQLCODE +595 cuando DSNTEP2 o DSNTEP4 ejecuta un FETCH para una sentencia SELECT, el programa continúa procesando la sentencia SELECT.
- Si se produce SQLCODE +354 cuando DSNTEP4 ejecuta un FETCH para una sentencia SELECT, el programa continúa procesando la sentencia SELECT.
- Si se produce SQLCODE +802 cuando DSNTEP2 o DSNTEP4 ejecuta un FETCH para una sentencia SELECT, el programa continúa procesando la sentencia SELECT si la sentencia de control TOLARTHWRN se establece en YES.

**YES**

Indica que el programa continúa procesando la instrucción SELECT si se produce una advertencia cuando el programa ejecuta un OPEN o FETCH para una instrucción SELECT.

**quiet**

Igual que SÍ, excepto que el programa suprime todos los mensajes de advertencia SQL de las sentencias OPEN o FETCH si el SQLCODE es 0 o mayor.

**--#SET TOLARTHWRN valor**

Indica si DSNTEP2 y DSNTEP4 continúan procesando una instrucción SQL SELECT después de que se devuelva una advertencia aritmética SQL (SQLCODE +802). es NO (valor predeterminado) o SÍ.

**--#SET PREPWARN value**

Especifica que DSNTEP2 o DSNTEP4 debe mostrar detalles sobre cualquier advertencia SQL que se encuentre en el momento de PREPARAR.

Independientemente de si especifica PREPWARN, cuando se encuentra una advertencia SQL en el momento de PREPARAR, el programa muestra el mensaje " SQLWARNING ON PREPARE " y establece el código de retorno en 4. Cuando especifique PREPWARN, el programa también mostrará los detalles sobre cualquier advertencia SQL.

**--#SET SQLFORMAT valor**

Especifica cómo DSNTEP2 o DSNTEP4 preprocesan las sentencias SQL antes de pasárlas a Db2. Seleccione una de las siguientes opciones:

**SQL**

Este es el modo preferido para las sentencias SQL que no sean de lenguaje de procedimiento SQL. Cuando se utiliza esta opción, que es la predeterminada, DSNTEP2 o DSNTEP4 contraen

cada línea de una instrucción SQL en una sola línea antes de pasar la instrucción a Db2. DSNTEP2 o DSNTEP4 también descarta todos los comentarios SQL.

#### **SQLCOMNT**

Este modo es adecuado para todos los SQL, pero está pensado principalmente para el procesamiento de lenguaje procedimental SQL. Cuando esta opción está activa, el comportamiento es similar al modo SQL, excepto que DSNTEP2 o DSNTEP4 no descartan los comentarios SQL. En su lugar, termina automáticamente cada comentario SQL con un carácter de avance de línea (hex 25), a menos que el comentario ya esté terminado por uno o más caracteres de formato de línea. Utilice esta opción para procesar el lenguaje de procedimientos SQL con una modificación mínima por DSNTEP2 o DSNTEP4.

#### **SQLPL**

Este modo es adecuado para todos los SQL, pero está pensado principalmente para el procesamiento de lenguaje procedimental SQL. Cuando esta opción está activa, DSNTEP2 o DSNTEP4 retienen los comentarios SQL y terminan cada línea de una instrucción SQL con un carácter de avance de línea (hex 25) antes de pasar la instrucción a Db2. Las líneas que terminan con un token de división no terminan con un carácter de avance de línea. Utilice este modo para obtener mejores diagnósticos y depuración del lenguaje de procedimientos SQL.

#### **--#SET MAXERRORS value**

*el valor* especifica el número de errores que DSNTEP2 y DSNTEP4 manejan antes de que se detenga el procesamiento. El valor predeterminado es 10. Utilice un valor de -1 para indicar que un programa debe tolerar un número ilimitado de errores.

#### **SYSPRINT**

Conjunto de datos de salida. DSNTEP2 y DSNTEP4 escriben mensajes informativos y de error en este conjunto de datos. DSNTEP2 y DSNTEP4 escriben registros de salida de no más de 133 bytes.

Definir todos los conjuntos de datos como conjuntos de datos secuenciales.

### **DSNTEP2 y códigos de retorno de DSNTEP4**

Tabla 180. DSNTEP2 y DSNTEP4 códigos de retorno

Código de retorno	Significado
0	Realización satisfactoria.
4	Una instrucción SQL recibió un código de advertencia.
8	Una instrucción SQL recibió un código de error.
12	La longitud de una declaración SQL fue más de 2097152 bytes, una declaración SQL devolvió un código de error grave (-8 nn o -9 nn ) o se produjo un error en la rutina de formato del mensaje SQL.

### **ejemplos**

#### **Ejemplo: invocar DSNTEP2**

Supongamos que desea utilizar DSNTEP2 para ejecutar instrucciones SQL SELECT que podrían contener caracteres DBCS. También desea una salida alineada a la izquierda. Su invocación se parece al siguiente ejemplo.

```
//RUNTEP2 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTEP2) PLAN(DSNTEPC1) PARMS(' /ALIGN(LHS) MIXED TOLWARN(YES) ') -
 LIB('DSN1210.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
 SELECT * FROM DSN8C10.PROJ;
```

### Ejemplo: invocar DSNTEP4

Supongamos que desea utilizar DSNTEP4 para ejecutar instrucciones SQL SELECT que podrían contener caracteres DBCS, y desea una salida alineada al centro. También quieras que DSNTEP4 recupere 250 filas a la vez. Su invocación se parece a la de la siguiente figura:

```
//RUNTEP2 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTEP4) PLAN(DSNTEPC1) PARMS(' /ALIGN(MID) MIXED') -
 LIB('DSN1210.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
--#SET MULT_FETCH 250
SELECT * FROM DSN8C10.EMP;
```

### Ejemplo: Cambiar el nivel de compatibilidad de la aplicación de las sentencias SQL dinámicas

Supongamos que Db2 está en el nivel de función V12R1M508 o superior, pero desea utilizar DSTNEP2 para ejecutar instrucciones SQL dinámicas en un nivel de compatibilidad de aplicación inferior. Puede utilizar los siguientes comandos de ejemplo para vincular paquetes para DSNTEP2 con dos opciones APPLCOMPAT diferentes:

```
BIND PACKAGE (M503TEP2) MEMBER(DSN@EP2L) APPLCOMPAT(V12R1M503) +
 CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC)
BIND PACKAGE (M508TEP2) MEMBER(DSN@EP2L) APPLCOMPAT(V12R1M508) +
 CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC)
BIND PLAN(DSNTEP2) PKLIST(M508TEP2.,M503TEP2.) +
 ACTION(REPLACE) RETAIN +
 CURRENTDATA(NO) ISO(CS) ENCODING(EBCDIC) SQLRULES(DB2)
```

En este ejemplo, V12R1M508 es el nivel de compatibilidad de aplicación predeterminado para DSNTEP2 porque el paquete llamado M508TEP2 está vinculado con APPLCOMPAT(V12R1M508) y aparece en primer lugar en el paso BIND PLAN. Sin embargo, también puede utilizar DSNTEP2 para emitir sentencias SQL dinámicas en el nivel de compatibilidad de aplicaciones V12R1M503. Para hacerlo en este ejemplo, especifique los siguientes parámetros cuando ejecute DSNTEP2.

```
RUN PROGRAM(DSNTEP2) PARMS(' /PKGSET(M503TEP2)')
```

DSNTEP2 emite implícitamente la siguiente declaración antes de ejecutar las sentencias SQL dinámicas, y ejecuta sentencias SQL dinámicas en el nivel de compatibilidad de la aplicación V12R1M503:

```
SET CURRENT PACKAGESET = 'M503TEP2'
```

### Ejemplo: Ejecutar sentencias SQL dinámicas en diferentes niveles de compatibilidad de aplicaciones en el mismo SYSIN

Suponga que emitió los mismos comandos BIND del ejemplo anterior y que desea crear un nuevo espacio de tabla segmentado multitable, que solo se puede crear en la versión de compatibilidad de la aplicación V12R1M503 o inferior. Como en el ejemplo anterior, el nivel de compatibilidad de aplicación predeterminado es V12R1M508, pero puede utilizar el siguiente ejemplo --#SET PKGSET instrucción de control para ejecutar algunas instrucciones a una compatibilidad de aplicación inferior en el mismo SYSIN.

```
--#SET PKGSET M503TEP2
CREATE TABLESPACE SEGTS1...
CREATE TABLE TB1 IN SEGTS1...
CREATE TABLE TB2 IN SEGTS1...
--#SET PKGSET M508TEP2
ALTER TABLESPACE SEGTS1 MOVE TABLE TB1 TO TABLESPACE PBGTS1...
ALTER TABLESPACE SEGTS1 MOVE TABLE TB2 TO TABLESPACE PBGTS2...
```

En este ejemplo, las tres primeras sentencias tienen éxito porque la sentencia de control --#SET PKGSET le dice a DSNTEP2 que las ejecute en el nivel de compatibilidad de la aplicación V12R1M503. Otra instrucción de control --#SET PKGSET cambia el nivel de compatibilidad de la aplicación a V12R1M508 porque las instrucciones ALTER TABLESPACE deben ejecutarse en este nivel o superior para especificar la cláusula MOVE TABLE.

### Ejemplo: cambiar el terminador SQL

Supongamos que especifica el parámetro SQLTERM(#) para indicar que el carácter # es el terminador de la sentencia. Entonces, una instrucción CREATE TRIGGER con punto y coma incrustado tiene este aspecto:

```
CREATE TRIGGER NEW_HIRE
 AFTER INSERT ON EMP
 FOR EACH ROW MODE DB2SQL
 BEGIN ATOMIC
 UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
 END#
```

Una instrucción CREATE PROCEDURE con punto y coma incrustado tiene el siguiente aspecto:

```
CREATE PROCEDURE PROC1 (IN PARM1 INT, OUT SCODE INT)
 LANGUAGE SQL
 BEGIN
 DECLARE SQLCODE INT;
 DECLARE EXIT HANDLER FOR SQLEXCEPTION
 SET SCODE = SQLCODE;
 UPDATE TBL1 SET COL1 = PARM1;
 END #
```

Tenga cuidado de elegir un carácter para el terminador de la instrucción que no se utilice dentro de la instrucción.

### Ejemplo: cambiar el terminador SQL dentro de una serie de sentencias SQL

Supongamos que tiene un conjunto existente de sentencias SQL al que desea agregar una sentencia CREATE TRIGGER que tiene puntos y comas incrustados. Puede utilizar la instrucción de control --#SET TERMINATOR. Puede utilizar el valor predeterminado de SQLTERM, que es un punto y coma, para todas las instrucciones SQL existentes.

Antes de ejecutar la instrucción CREATE TRIGGER, incluya la instrucción de control --#SET TERMINATOR # para cambiar el terminador SQL al carácter #:

```
SELECT * FROM DEPT;
SELECT * FROM ACT;
SELECT * FROM EMPPROJACT;
SELECT * FROM PROJ;
SELECT * FROM PROJECT;
--#SET TERMINATOR #
CREATE TRIGGER NEW_HIRE
 AFTER INSERT ON EMP
 FOR EACH ROW MODE DB2SQL
 BEGIN ATOMIC
 UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
 END#
```

Consulte el siguiente análisis del conjunto de datos SYSIN para obtener más información sobre la instrucción de control --#SET.

## Aplicaciones de ejemplo suministradas con Db2 for z/OS

Db2 proporciona ejemplos de aplicaciones para ayudarle con las técnicas de programación e Db2 es y las prácticas de codificación dentro de cada uno de los cuatro entornos: batch, TSO, IMS, y CICS. Las aplicaciones de ejemplo contienen varias aplicaciones que pueden aplicarse a la gestión de una empresa.

Este tema describe las aplicaciones de muestra de Db2 , así como los entornos en los que se ejecuta cada aplicación. También proporciona información sobre cómo utilizar las aplicaciones y cómo imprimir los listados de aplicaciones.

Puede examinar el código fuente de los programas de aplicación de muestra en la biblioteca de muestras en línea incluida con el producto Db2 . El nombre de esta biblioteca de muestras es DSN1210.SDSNSAMP.

## Utilización de las aplicaciones de ejemplo

Puede utilizar las aplicaciones de forma interactiva accediendo a los datos de las tablas de muestra en las pantallas (paneles). También puede acceder a las tablas de muestra en lote cuando utilice las aplicaciones del teléfono. Todos los objetos de muestra tienen autorización PÚBLICA, lo que facilita su ejecución.

### Conceptos relacionados

[Ejemplos de programación de Db2 \(Db2 Programming samples\)](#)

### Referencia relacionada

[Db2 for z/OS Cambio](#)

## Tipos de aplicaciones de muestra

Db2 proporciona varias aplicaciones de muestra que gestionan información de muestra de la empresa. Estas aplicaciones también demuestran cómo utilizar procedimientos almacenados, funciones definidas por el usuario y LOB.

### Solicitud de organización:

La aplicación de la organización gestiona la siguiente información de la empresa:

- Estructura administrativa del departamento
- Departamentos individuales
- Empleados individuales.

La gestión de la información sobre las estructuras administrativas de los departamentos implica la forma en que los departamentos se relacionan con otros departamentos. Puede ver o cambiar la estructura organizativa de un departamento individual, así como la información sobre empleados individuales en cualquier departamento. La aplicación de la organización se ejecuta de forma interactiva en el ISPF /TSO, IMS, y CICS y está disponible en PL/I y COBOL.

### Solicitud de proyecto:

La aplicación de proyectos gestiona información sobre las actividades de proyectos de una empresa, incluyendo lo siguiente:

- Estructuras de proyectos
- Listados de actividades del proyecto
- Procesamiento de proyectos individuales
- Procesamiento de estimación de actividad de proyecto individual
- Procesamiento de dotación de personal para proyectos individuales.

Cada departamento trabaja en proyectos que contienen conjuntos de actividades relacionadas. La información disponible sobre estas actividades incluye las asignaciones de personal, las estimaciones de tiempo de finalización para el proyecto en su conjunto y las actividades individuales dentro de un proyecto. La aplicación del proyecto se ejecuta de forma interactiva en IMS y CICS y solo está disponible en PL/I.

### Aplicación para teléfono:

La aplicación del teléfono le permite ver o actualizar los números de teléfono de los empleados. Existen diferentes versiones de la solicitud de ISPF /TSO, CICS, IMS y por lotes:

- ISPF las aplicaciones /TSO utilizan COBOL y PL/I.
- CICS y IMS aplicaciones utilizan PL/I.
- Las aplicaciones por lotes utilizan C, C++, COBOL, FORTRAN y PL/I.

## **Aplicaciones de procedimientos almacenados:**

Hay tres conjuntos de aplicaciones de procedimientos almacenados:

### **aplicaciones IFI**

Estas aplicaciones le permiten pasar comandos de e Db2 encia desde un programa cliente a un procedimiento almacenado, que ejecuta los comandos en un servidor de e Db2 encia utilizando la interfaz de instrumentación (IFI). Hay dos conjuntos de programas de cliente y procedimientos almacenados. Un conjunto tiene un cliente PL/I y un procedimiento almacenado; el otro conjunto tiene un cliente C y un procedimiento almacenado.

### **Solicitud de ODBA**

Esta aplicación muestra cómo puede utilizar la IMS Interfaz ODBA para acceder a IMS bases de datos desde procedimientos almacenados. El procedimiento almacenado accede a la IMS base de datos DL/I de muestra. El programa cliente y el procedimiento almacenado están escritos en COBOL.

### **Aplicación de procedimiento almacenado de servicios públicos**

Esta aplicación muestra cómo llamar al procedimiento almacenado de utilidades.

### **Aplicaciones de procedimientos SQL**

Hay disponibles ejemplos de aplicaciones tanto para procedimientos SQL externos como para procedimientos SQL nativos:

- Las aplicaciones para procedimientos SQL externos demuestran cómo escribir, preparar e invocar dichos procedimientos. Un conjunto de aplicaciones muestra cómo preparar procedimientos SQL utilizando JCL. El otro conjunto de aplicaciones muestra cómo preparar procedimientos SQL utilizando el procesador de procedimientos SQL. Los programas del cliente están escritos en C.
- El ejemplo de trabajo para un procedimiento SQL nativo muestra cómo preparar un procedimiento SQL nativo, cómo gestionar versiones de procedimientos SQL nativos y, opcionalmente, cómo implementar un procedimiento SQL nativo en un servidor remoto. La muestra también prepara y ejecuta una muestra de llamada en el lenguaje C

### **Actualizar aplicación WLM**

Esta aplicación es un programa cliente que llama al procedimiento almacenado WLM\_REFRESH proporcionado por Db2para actualizar un entorno WLM. Este programa está escrito en C.

### **Aplicación de informes de parámetros del sistema**

Esta aplicación es un programa cliente que llama al Db2 –Procedimiento almacenado ADMIN\_INFO\_SYSPARM proporcionado para mostrar la configuración actual de los parámetros del sistema. Este programa está escrito en C.

Todas las aplicaciones de procedimientos almacenados se ejecutan en el entorno por lotes TSO.

## **Aplicaciones de funciones definidas por el usuario:**

Las aplicaciones de funciones definidas por el usuario consisten en un programa cliente que invoca las funciones de muestra definidas por el usuario y un conjunto de funciones definidas por el usuario que realizan las siguientes funciones:

- Convertir la fecha actual a un formato especificado por el usuario
- Convertir una fecha de un formato a otro
- Convertir la hora actual a un formato especificado por el usuario
- Convertir una fecha de un formato a otro
- Devuelve el día de la semana para una fecha especificada por el usuario
- Volver al mes para una fecha especificada por el usuario

- Dar formato a un número de punto flotante como valor de moneda
- Devolver el nombre de la tabla para una tabla, vista o alias
- Devolver el calificador para una tabla, vista o alias
- Devolver la ubicación de una mesa, vista o alias
- Volver a la tabla de información meteorológica

Todos los programas están escritos en C o C++ y se ejecutan en el entorno por lotes TSO.

### **Solicitud de LOB:**

La aplicación LOB muestra cómo realizar las siguientes tareas:

- Definir objetos de tipo LOB ( Db2 ) para contener datos LOB
- Rellene las tablas de Db2 con datos LOB utilizando la utilidad LOAD, o utilizando sentencias INSERT y UPDATE cuando los datos sean demasiado grandes para utilizarlos con la utilidad LOAD
- Manipular los datos LOB utilizando localizadores LOB

Los programas que crean y rellenan los objetos LOB utilizan DSNTIAD y se ejecutan en el entorno por lotes de TSO. El programa que manipula los datos LOB está escrito en C y se ejecuta en un entorno de tiempo compartido ( ISPF, TSO ).

## **Entornos y lenguajes de aplicación para las aplicaciones de ejemplo**

Las aplicaciones de muestra demuestran cómo ejecutar aplicaciones de Db2 en el TSO, IMS, o CICS.

La siguiente tabla muestra los entornos en los que se ejecuta cada aplicación y los lenguajes que utilizan las aplicaciones para cada entorno.

*Tabla 181. Idiomas y entornos de aplicación*

<b>Programas</b>	<b>ISPF /TSO</b>	<b>IMS</b>	<b>CICS</b>	<b>Despliegue</b>	<b>SPUFI</b>
Programas SQL dinámicos				Assembler PL/I	
Rutinas de salida	Assembler	Assembler	Assembler	Assembler	Assembler
Organización	COBOL	COBOL PL/I	COBOL PL/I		
Teléfono	COBOL PL/I Montador1	PL/I	PL/I	COBOL FORTRAN PL/I C C++	
Proyecto		PL/I	PL/I		
Rutinas de formato SQLCA		Assembler	Assembler	Assembler	Assembler
Procedimientos almacenados		COBOL		PL/I C SQL	

Tabla 181. Idiomas y entornos de aplicación (continuación)

Programas	ISPF / TSO	IMS	CICS	Despliegue	SPUFI
Funciones definidas por el usuario				C C++	
LOB	C				

**Notas:**

1. Subrutina ensambladora DSN8CA.

## Ejemplos de aplicaciones en TSO

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

Tabla 182. Ejemplos de solicitudes de Db2 s para TSO

Aplicación	Nombre de programa	Preparación Nombre del miembro de JCL	recurso de conexión	Descripción
Teléfono	<a href="#">DSN8BC3</a>	DSNTEJ2C	DSNELI	Este programa por lotes COBOL enumera los números de teléfono de los empleados y los actualiza si se solicita.
Teléfono	<a href="#">DSN8BD3</a>	DSNTEJ2D	DSNELI	Este programa por lotes C enumera los números de teléfono de los empleados y los actualiza si se solicita.
Teléfono	<a href="#">DSN8BE3</a>	DSNTEJ2E	DSNELI	Este programa por lotes en C++ enumera los números de teléfono de los empleados y los actualiza si se solicita.
Teléfono	<a href="#">DSN8BP3</a>	DSNTEJ2P	DSNELI	Este programa por lotes PL/I enumera los números de teléfono de los empleados y los actualiza si se solicita.
Teléfono	<a href="#">DSN8BF3</a>	DSNTEJ2F	DSNELI	Este programa FORTRAN enumera los números de teléfono de los empleados y los actualiza si se solicita.
Organización	<a href="#">DSN8HC3</a>	DSNTEJ3C o DSNTEJ6	DSNALI	Este programa COBOL ISPF, muestra y actualiza información sobre un departamento local. También puede mostrar y actualizar información sobre un empleado en una ubicación local o remota.
Teléfono	<a href="#">DSN8SC3</a>	DSNTEJ3C	DSNALI	Este programa COBOL ( ISPF ) enumera los números de teléfono de los empleados y los actualiza si se solicita.
Teléfono	<a href="#">DSN8SP3</a>	DSNTEJ3P	DSNALI	Este programa de PL/I ISPF, enumera los números de teléfono de los empleados y los actualiza si se solicita.
UNLOAD	<a href="#">DSNTIAUL</a>	DSNTEJ2A	DSNELI	Este programa de lenguaje ensamblador descarga los datos de una tabla o vista y produce sentencias de control de utilidad LOAD para los datos.

Tabla 182. Ejemplos de solicitudes de Db2 para TSO (continuación)

Aplicación	Nombre de programa	Preparación Nombre del miembro de JCL	recurso de conexión	Descripción
SQL dinámico	<u>DSNTIAD</u>	DSNTIJTM	DSNELI	Este programa de lenguaje ensamblador ejecuta dinámicamente sentencias no SELECT leídas desde SYSIN; es decir, utiliza SQL dinámico para ejecutar sentencias SQL no SELECT.
SQL dinámico	<u>DSNTEP2</u>	DSNTEJ1P o DSNTEJ1L	DSNELI	Este programa PL/I ejecuta dinámicamente sentencias SQL leídas desde SYSIN. A diferencia de DSNTIAD, esta aplicación también puede ejecutar sentencias SELECT.
Procedimientos almacenados 1	<u>DSN8EP1</u>	DSNTEJ6P	DSNELI	Los trabajos DSNTEJ6P y DSNTEJ6S preparan una versión PL/I de la aplicación. Este ejemplo ejecuta comandos de Db2 a utilizando la interfaz de instrumentación (IFI).
Procedimiento almacenado1	<u>DSN8EP2</u>	DSNTEJ6S	DSNRLI	
Procedimientos almacenados 1	<u>DSN8EPU</u>	DSNTEJ6U	DSNELI	La muestra que prepara el trabajo de DSNTEJ6U, invoca el procedimiento almacenado de utilidades.
Procedimientos almacenados 1	<u>DSN8ED1</u>	DSNTEJ6D	DSNELI	Los trabajos DSNTEJ6D y DSNTEJ6T preparan una versión C de la aplicación. El procedimiento almacenado C utiliza conjuntos de resultados para devolver comandos al cliente. Este ejemplo ejecuta comandos de Db2 a utilizando la interfaz de instrumentación (IFI).
Procedimientos almacenados 1	<u>DSN8ED2</u>	DSNTEJ6T	DSNRLI	
Procedimientos almacenados 1	<u>DSN8EC1</u>	DSNTEJ61	DSNRLI	La muestra preparada por jobs DSNTEJ61 y DSNTEJ62 muestra un procedimiento almacenado que accede a IMS bases de datos a través de la interfaz ODBA.
Procedimientos almacenados 1	<u>DSN8EC2</u>	DSNTEJ62	DSNELI	
Procedimientos almacenados 1	<u>DSN8ES1</u>	DSNTEJ63	DSNRLI	La muestra preparada por jobs DSNTEJ63 y DSNTEJ64 muestra cómo preparar un procedimiento SQL utilizando JCL.
Procedimientos almacenados 1	<u>DSN8ED3</u>	DSNTEJ64	DSNELI	

Tabla 182. Ejemplos de solicitudes de Db2 s para TSO (continuación)

Aplicación	Nombre de programa	Preparación Nombre del miembro de JCL	recurso de conexión	Descripción
Procedimientos almacenados 1	<a href="#">DSN8ES2</a>	DSNTEJ65	DSNRLI	La muestra preparada por job DSNTEJ65, muestra cómo preparar un procedimiento SQL utilizando el procesador de procedimientos SQL.
Procedimientos almacenados 1	<a href="#">DSN8ED6</a>	DSNTEJ6W	DSNELI	La muestra preparada por job DSNTEJ6W muestra cómo preparar y ejecutar un programa de cliente que llama a un procedimiento almacenado suministrado por WLM (Db2) para actualizar un entorno WLM.
Procedimientos almacenados 1	<a href="#">DSN8ED7</a>	DSNTEJ6Z	DSNELI	La muestra preparada por job DSNTEJ6Z muestra cómo preparar y ejecutar un programa de cliente que llama a un procedimiento almacenado suministrado por el sistema (Db2) para mostrar la configuración actual de los parámetros del sistema.
Procedimientos almacenados 1	<a href="#">DSN8ED9</a>	DSNTEJ66	DSNELI	La muestra preparada por job DSNTEJ66, muestra cómo preparar y ejecutar un programa de cliente que llama a un procedimiento SQL nativo, gestiona versiones de ese procedimiento y, opcionalmente, lo implementa en un servidor remoto. DSN8ES3 es el procedimiento SQL nativo de muestra y DSN8ED9 es el llamador de lenguaje C de muestra de DSN8ES3.
Funciones definidas por el usuario	<a href="#">DSN8DUAD</a>	DSNTEJ2U	DSNRLI	Estas aplicaciones C consisten en un conjunto de funciones escalares definidas por el usuario que pueden invocarse a través de SPUFI o DSNTEP2.
Funciones definidas por el usuario	<a href="#">DSN8DUAT</a>	DSNTEJ2U	DSNRLI	
Funciones definidas por el usuario	<a href="#">DSN8DUCD</a>	DSNTEJ2U	DSNRLI	
Funciones definidas por el usuario	<a href="#">DSN8DUCT</a>	DSNTEJ2U	DSNRLI	
Funciones definidas por el usuario	<a href="#">DSN8DUCY</a>	DSNTEJ2U	DSNRLI	
Funciones definidas por el usuario	<a href="#">DSN8DUTI</a>	DSNTEJ2U	DSNRLI	

Tabla 182. Ejemplos de solicitudes de Db2 s para TSO (continuación)

Aplicación	Nombre de programa	Preparación Nombre del miembro de JCL	recurso de conexión	Descripción
Funciones definidas por el usuario	<u>DSN8DUWC</u>	DSNTEJ2U	DSNRLI	La función de tabla definida por el usuario DSN8DUWF puede ser invocada por el programa cliente C DSN8DUWC.
Funciones definidas por el usuario	<u>DSN8DUWF</u>	DSNTEJ2U	DSNRLI	
Funciones definidas por el usuario	<u>DSN8EUDN</u>	DSNTEJ2U	DSNRLI	Estas aplicaciones C++ consisten en un conjunto de funciones escalares definidas por el usuario que pueden invocarse a través de SPUFI o DSNTEP2.
Funciones definidas por el usuario	<u>DSN8EUMN</u>	DSNTEJ2U	DSNRLI	
Funciones definidas por el usuario	<u>DSN8HDFS</u>	DSNTEJBI	DSNRLI	La función de tabla definida por el usuario HDFS_READ, que está preparada por DSNTEJBI, lee datos de un archivo separado por delimitadores en el Sistema de Archivos Distribuidos de Oracle ( Hadoop Distributed File System, HDFS ). Esta función definida por el usuario puede invocarse a través de SPUFI o DSNTEP2.
Funciones definidas por el usuario	<u>DSN8JAQL</u>	DSNTEJBI	DSNRLI	La función escalar definida por el usuario JAQL_SUBMIT, que está preparada por DSNTEJBI, invoca un IBMInfoSphereBigInsights Consulta Jaql. Esta función definida por el usuario puede invocarse a través de SPUFI o DSNTEP2.
LOB	<u>DSN8DLPL</u>	DSNTEJ71	DSNELI	Estas aplicaciones muestran cómo llenar una columna LOB que supera los 32 KB, manipular los datos utilizando las funciones integradas
LOB	<u>DSN8DLCT</u>	DSNTEJ71	DSNELI	POSSTR y SUBSTR, y mostrar los datos en un
LOB	<u>DSN8DLRV</u>	DSNTEJ73	DSNELI	ISPF e utilizando GDDM.
LOB	<u>DSN8DLPV</u>	DSNTEJ75	DSNELI	

**Nota:**

1. Todas las aplicaciones de procedimientos almacenados constan de un programa de llamada, un programa de procedimiento almacenado o ambos.

**Referencia relacionada**

Conjuntos de datos que utiliza el precompilador

Cuando invoca el precompilador, necesita proporcionar conjuntos de datos que contienen entrada para el precompilador, como las sentencias de programación de host o sentencias SQL. También necesita proporcionar conjuntos de datos donde el precompilador pueda almacenar la salida, como el código de origen modificado o los mensajes de diagnóstico.

## DSN8BC3

ESTE MÓDULO LISTA LOS NÚMEROS DE TELÉFONO DE EMPLEADOS Y LOS ACTUALIZA SI LO DESEA.

```

IDENTIFICATION DIVISION.

PROGRAM-ID. DSN8BC3.

***** DSN8BC3 - DB2 SAMPLE PHONE APPLICATION - COBOL - BATCH ****
*
* MODULE NAME = DSN8BC3
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
* PHONE APPLICATION
* BATCH
* COBOL
*
*LICENSED MATERIALS - PROPERTY OF IBM
*5605-DB2
*(C) COPYRIGHT 1982, 2010 IBM CORP. ALL RIGHTS RESERVED.
*
*STATUS = VERSION 10
*
* FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND
* UPDATES THEM IF DESIRED.
*
* NOTES = NONE
*
* MODULE TYPE = COBOL PROGRAM
* PROCESSOR = DB2 PRECOMPILER, VS COBOL
* MODULE SIZE = SEE LINK EDIT
* ATTRIBUTES = NOT REENTRANT OR REUSABLE
*
*
* ENTRY POINT = DSN8BC3
* PURPOSE = SEE FUNCTION
* LINKAGE = INVOKED FROM DSN RUN
* INPUT =
*
* SYMBOLIC LABEL/NAME = CARDIN
* DESCRIPTION = INPUT REQUEST FILE
*
* SYMBOLIC LABEL/NAME = VPHONE
* DESCRIPTION = VIEW OF TELEPHONE
* INFORMATION
*
*
* OUTPUT =
*
* SYMBOLIC LABEL/NAME = REPORT
* DESCRIPTION = REPORT OF EMPLOYEE
* PHONE NUMBERS
*
* SYMBOLIC LABEL/NAME = VEMPLP
* DESCRIPTION = VIEW OF EMPLOYEE
* INFORMATION
*
* EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION
*
* EXIT-ERROR =
*
* RETURN CODE = NONE
*
* ABEND CODES = NONE
*
* ERROR-MESSAGES =
* DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED
* DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE
* DSN8008I - NO EMPLOYEE FOUND IN TABLE
* DSN8053I - ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED
* DSN8060E - SQL ERROR, RETURN CODE IS:
* DSN8061E - ROLLBACK FAILED, RETURN CODE IS:
* DSN8068E - INVALID REQUEST, SHOULD BE 'L' OR 'U'
* DSN8075E - MESSAGE FORMAT ROUTINE ERROR,
* RETURN CODE IS:
*
* EXTERNAL REFERENCES =
* ROUTINES/SERVICES =
* DSNTIAR - TRANSLATE SQLCA INTO MESSAGES
* DSN8MCG - ERROR MESSAGE ROUTINE
*
* DATA-AREAS = NONE
*
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA

```

```

* TABLES = NONE
*
*
* CHANGE-ACTIVITY = NONE
*
*
* *PSEUDOCODE*
*
* PROCEDURE
* GET FIRST INPUT
* DO WHILE MORE INPUT
* CREATE REPORT HEADING
*
* CASE (ACTION)
*
* SUBCASE ('L')
* IF LASTNAME IS '*' THEN
* LIST ALL EMPLOYEES
* ELSE
* IF LASTNAME CONTAINS '%' THEN
* LIST EMPLOYEES GENERIC
* ELSE
* LIST EMPLOYEES SPECIFIC
* ENDIF
* ENDIF
* ENDSUB
*
* SUBCASE ('U')
* UPDATE PHONENUMBER FOR EMPLOYEE
* WRITE CONFIRMATION MESSAGE
* OTHERWISE
* INVALID REQUEST
* ENDSUB
*
* ENDCASE
* GET NEXT INPUT
* END
*
* IF SQL ERROR OCCURS THEN
* DO
* FORMAT ERROR MESSAGE
* ROLLBACK
* END
* END.

/
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SPECIAL-NAMES. C01 IS TO-TOP-OF-PAGE.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
 SELECT CARDIN
 ASSIGN TO DA-S-CARDIN.
 SELECT REPOUT
 ASSIGN TO UT-S-REPORT.

DATA DIVISION.

FILE SECTION.
FD CARDIN
 RECORD CONTAINS 80 CHARACTERS
 BLOCK CONTAINS 0 RECORDS
 LABEL RECORDS ARE OMITTED.
01 CARDREC PIC X(80).

FD REPOUT
 RECORD CONTAINS 120 CHARACTERS
 LABEL RECORDS ARE OMITTED
 DATA RECORD IS REPREC.
01 REPREC PIC X(120).
/
WORKING-STORAGE SECTION.

* STRUCTURE FOR INPUT

01 IOAREA.
 02 ACTION PIC X(01).
 02 LNAME PIC X(15).
 02 FENAME PIC X(12).

```

```

02 ENO PIC X(06).
02 NEWNO PIC X(04).
02 FILLER PIC X(42).

* REPORT HEADER STRUCTURE *

01 REPHDR1.
 02 FILLER PIC X(29)
 VALUE '-----'.
 02 FILLER PIC X(21)
 VALUE ' TELEPHONE DIRECTORY '.
 02 FILLER PIC X(29)
 VALUE '-----'.

01 REPHDR2.
 02 FILLER PIC X(09) VALUE 'LAST NAME'.
 02 FILLER PIC X(07) VALUE SPACES.
 02 FILLER PIC X(10) VALUE 'FIRST NAME'.
 02 FILLER PIC X(03) VALUE SPACES.
 02 FILLER PIC X(08) VALUE 'INITIAL'.
 02 FILLER PIC X(07) VALUE 'PHONE'.
 02 FILLER PIC X(09) VALUE 'EMPLOYEE'.
 02 FILLER PIC X(05) VALUE 'WORK'.
 02 FILLER PIC X(04) VALUE 'WORK'.

01 REPHDR3.
 02 FILLER PIC X(37) VALUE SPACES.
 02 FILLER PIC X(07) VALUE 'NUMBER'.
 02 FILLER PIC X(09) VALUE 'NUMBER'.
 02 FILLER PIC X(05) VALUE 'DEPT'.
 02 FILLER PIC X(05) VALUE 'DEPT'.
 02 FILLER PIC X(04) VALUE 'NAME'.

* REPORT STRUCTURE *

01 REPDATA.
 02 RLNAME PIC X(15).
 02 FILLER PIC X(01) VALUE SPACES.
 02 RFNAME PIC X(12).
 02 FILLER PIC X(04) VALUE SPACES.
 02 RMIDINIT PIC X(01).
 02 FILLER PIC X(04) VALUE SPACES.
 02 RPHONE PIC X(04).
 02 FILLER PIC X(03) VALUE SPACES.
 02 REMPNO PIC X(06).
 02 FILLER PIC X(03) VALUE SPACES.
 02 RDEPTNO PIC X(03).
 02 FILLER PIC X(02) VALUE SPACES.
 02 RDEPTNAME PIC X(36).

* WORKAREAS *

01 LNAME-WORK.
 49 LNAME-WORKL PIC S9(4) COMP.
 49 LNAME-WORKC PIC X(15).

01 FNAME-WORK.
 49 FNAME-WORKL PIC S9(4) COMP.
 49 FNAME-WORKC PIC X(12).

77 INPUT-SWITCH PIC X VALUE 'Y'.
 88 NOMORE-INPUT PIC X VALUE 'N'.
77 NOT-FOUND PIC S9(9) COMP VALUE +100.

* VARIABLES FOR ERROR-HANDLING *

01 ERROR-MESSAGE.
 02 ERROR-LEN PIC S9(4) COMP VALUE +960.
 02 ERROR-TEXT PIC X(120) OCCURS 10 TIMES
 INDEXED BY ERROR-INDEX.
77 ERROR-TEXT-LEN PIC S9(9) COMP VALUE +120.

* SQL INCLUDE FOR SQLCA *

EXEC SQL INCLUDE SQLCA END-EXEC.

* SQL DECLARATION FOR VIEW VPHONE *

EXEC SQL DECLARE VPHONE TABLE
 (LASTNAME VARCHAR(15) NOT NULL,

```

```

 FIRSTNAME VARCHAR(12) NOT NULL,
 MIDDLEINITIAL CHAR(01) NOT NULL,
 PHONENUMBER CHAR(04),
 EMPLOYEENUMBER CHAR(06) NOT NULL,
 DEPTNUMBER CHAR(03) NOT NULL,
 DEPTNAME VARCHAR(36) NOT NULL)
END-EXEC.

* STRUCTURE FOR PPHONE RECORD

01 PPHONE.
 02 LASTNAME.
 49 LASTNAMEL PIC S9(4) COMP.
 49 LASTNAMEC PIC X(15) VALUE SPACES.
 02 FIRSTNAME.
 49 FIRSTNAMEL PIC S9(4) COMP.
 49 FIRSTNAMEC PIC X(12) VALUE SPACES.
 02 MIDDLEINITIAL PIC X(01).
 02 PHONENUMBER PIC X(04).
 02 EMPLOYEENUMBER PIC X(06).
 02 DEPTNUMBER PIC X(03).
 02 DEPTNAME.
 49 DEPTNAMEL PIC S9(4) COMP.
 49 DEPTNAMEC PIC X(36) VALUE SPACES.
*
* 77 PERCENT-COUNTER PIC S9(4) COMP.

* SQL DECLARATION FOR VIEW VEMPLP

EXEC SQL DECLARE VEMPLP TABLE
 (EMPLOYEENUMBER CHAR(06) NOT NULL,
 PHONENUMBER CHAR(04))
END-EXEC.

* SQL CURSORS

*** CURSOR LISTS ALL EMPLOYEE NAMES

EXEC SQL DECLARE TELE1 CURSOR FOR
 SELECT *
 FROM VPHONE
END-EXEC.

*** CURSOR LISTS ALL EMPLOYEE NAMES WITH A PATTERN (%) OR (_)
*** FOR LAST NAME

EXEC SQL DECLARE TELE2 CURSOR FOR
 SELECT *
 FROM VPHONE
 WHERE LASTNAME LIKE :LNAME-WORK
 AND FIRSTNAME LIKE :FNAME-WORK
END-EXEC.

*** CURSOR LISTS ALL EMPLOYEES WITH A SPECIFIC
*** LAST NAME

EXEC SQL DECLARE TELE3 CURSOR FOR
 SELECT *
 FROM VPHONE
 WHERE LASTNAME = :LNAME
 AND FIRSTNAME LIKE :FNAME-WORK
END-EXEC.

/
***** FIELDS SENT TO MESSAGE ROUTINE *****
01 MAJOR PIC X(07) VALUE 'DSN8BC3'.
01 MSGCODE PIC X(4).
01 OUTMSG PIC X(69).
01 MSG-REC1.
 02 OUTMSG1 PIC X(69).
 02 RETCODE PIC S9(9).
01 MSG-REC2.
 02 OUTMSG2 PIC X(69).

```

```

PROCEDURE DIVISION.
*-----

***** * SQL RETURN CODE HANDLING *
***** EXEC SQL WHENEVER SQLERROR GOTO DBERROR END-EXEC.
EXEC SQL WHENEVER SQLWARNING GOTO DBERROR END-EXEC.
EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.

***** * MAIN PROGRAM ROUTINE *
***** PROG-START.
* **OPEN FILES
OPEN INPUT CARDIN
 OUTPUT REPOUT.

* **GET FIRST INPUT
READ CARDIN RECORD INTO IOAREA
 AT END MOVE 'N' TO INPUT-SWITCH.

* **MAIN ROUTINE
PERFORM PROCESS-INPUT
 UNTIL NOMORE-INPUT.
PROG-END.
* **CLOSE FILES
CLOSE CARDIN
 REPOUT.
GOBACK.

***** * CREATE REPORT HEADING *
***** SELECT ACTION *
***** PROCESS-INPUT.
* **PRINT HEADING
WRITE REPREC FROM REPHDR1
 AFTER ADVANCING TO-TOP-OF-PAGE.
WRITE REPREC FROM REPHDR2
 AFTER ADVANCING 2 LINES.
WRITE REPREC FROM REPHDR3.

* **SELECT ACTION
IF ACTION = 'L'
 PERFORM LIST-FUNCTION
ELSE
 IF ACTION = 'U'
 PERFORM TELEPHONE-UPDATE
 ELSE
 **INVALID REQUEST
 **PRINT ERROR MESSAGE
MOVE '068E' TO MSGCODE
CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
MOVE OUTMSG TO OUTMSG2
WRITE REPREC FROM MSG-REC2
 AFTER ADVANCING 2 LINES.
READ CARDIN RECORD INTO IOAREA
 AT END MOVE 'N' TO INPUT-SWITCH.
/
***** * DETERMINE FORM OF NAME USED TO LIST EMPLOYEES *
***** LIST-FUNCTION.
* **NO LAST NAME GIVEN
IF LNAME = SPACES
 MOVE '%' TO LNAME.
* **NO FIRST NAME GIVEN
IF FNAME = SPACES
 MOVE '%' TO FNAME.
* **LIST ALL EMPLOYEES
IF LNAME = '*'
 PERFORM LIST-ALL
ELSE
* **UNSTRING LAST NAME
 UNSTRING LNAME
 DELIMITED BY SPACE
 INTO LNAME-WORKC
 COUNT IN LNAME-WORKL
*
* **UNSTRING FIRST NAME

```



```

 EXEC SQL OPEN TELE3 END-EXEC.

*
* **GET EMPLOYEES
 EXEC SQL FETCH TELE3 INTO :PPHONE END-EXEC.

 IF SQLCODE = NOT-FOUND
*
* **NO EMPLOYEE FOUND
* **PRINT ERROR MESSAGE
 MOVE '008I' TO MSGCODE
 CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
 MOVE OUTMSG TO OUTMSG2
 WRITE REPREC FROM MSG-REC2
 AFTER ADVANCING 2 LINES
 ELSE
*
* **LIST SPECIFIC EMPLOYEE(S)
 PERFORM PRINT-AND-GET3
 UNTIL SQLCODE IS NOT EQUAL TO ZERO.

*
* **CLOSE CURSOR
 EXEC SQL CLOSE TELE3 END-EXEC.

PRINT-AND-GET3.
 PERFORM PRINT-A-LINE.
 EXEC SQL FETCH TELE3 INTO :PPHONE END-EXEC.
/
***** * PRINT A LINE OF INFORMATION FROM DIRECTORY *
***** PRINT-A-LINE.
*
* **GET INFORMATION
 MOVE LASTNAMEC TO RLNAME.
 MOVE FIRSTNAMEC TO RFNAME.
 MOVE MIDDLEINITIAL TO RMIDINIT.
 MOVE PHONENUMBER OF PPHONE TO RPHONE.
 MOVE EMPLOYEENUMBER OF PPHONE TO REMPNO.
 MOVE DEPTNUMBER TO RDEPTNO.
 MOVE DEPTNAMEC TO RDEPTNAME.
*
* **PRINT INFORMATION
 WRITE REPREC FROM REpdata
 AFTER ADVANCING 2 LINES.

 MOVE SPACES TO LASTNAMEC
 FIRSTNAMEC
 DEPTNAMEC.
/
***** * UPDATES PHONE NUMBERS FOR EMPLOYEES *
***** TELEPHONE-UPDATE.
 EXEC SQL UPDATE VEMPLP
 SET PHONENUMBER = :NEWNO
 WHERE EMPLOYEENUMBER = :ENO END-EXEC.
 IF SQLCODE = ZERO
*
* **EMPLOYEE FOUND
* **UPDATE SUCCESSFUL
* **PRINT CONFIRMATION
* **MESSAGE
 MOVE '004I' TO MSGCODE
 ELSE
*
* **NO EMPLOYEE FOUND
* **UPDATE FAILED
* **PRINT ERROR MESSAGE
 MOVE '007E' TO MSGCODE.
 CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
 MOVE OUTMSG TO OUTMSG2.
 WRITE REPREC FROM MSG-REC2
 AFTER ADVANCING 2 LINES.
/
***** * SQL ERROR OCCURRED - GET ERROR MESSAGE *
***** DBERROR.
*
* **SQL ERROR
* **PRINT ERROR MESSAGE
 MOVE '060E' TO MSGCODE
 CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
 MOVE OUTMSG TO OUTMSG1 OF MSG-REC1.
 MOVE SQLCODE TO RETCODE OF MSG-REC1.
 WRITE REPREC FROM MSG-REC1
 AFTER ADVANCING 2 LINES.
 CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
 IF RETURN-CODE = ZERO

```

```

 PERFORM ERROR-PRINT VARYING ERROR-INDEX
 FROM 1 BY 1 UNTIL ERROR-INDEX GREATER THAN 10
 ELSE

 * **MESSAGE FORMAT
 * **ROUTINE ERROR
 * **PRINT ERROR MESSAGE
 MOVE '075E' TO MSGCODE
 CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
 MOVE OUTMSG TO OUTMSG1 OF MSG-REC1
 MOVE RETURN-CODE TO RETCODE OF MSG-REC1
 WRITE REPREC FROM MSG-REC1
 AFTER ADVANCING 2 LINES.

* SQL RETURN CODE HANDLING WHEN PROCESSING CANNOT PROCEED *

 EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
 EXEC SQL WHENEVER SQLWARNING CONTINUE END-EXEC.
 EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.

 * **PERFORM ROLLBACK
 EXEC SQL ROLLBACK END-EXEC.

 IF SQLCODE = ZERO

 * **ROLLBACK SUCCESSFUL
 * **PRINT CONFIRMATION
 * **MESSAGE
 MOVE '053I' TO MSGCODE
 ELSE

 * **ROLLBACK FAILED
 * **PRINT ERROR MESSAGE
 MOVE '061E' TO MSGCODE.
 CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
 MOVE OUTMSG TO OUTMSG1 OF MSG-REC1.
 MOVE SQLCODE TO RETCODE OF MSG-REC1.
 WRITE REPREC FROM MSG-REC1
 AFTER ADVANCING 2 LINES.
 GO TO PROG-END.

* PRINT MESSAGE TEXT

 ERROR-PRINT.
 WRITE REPREC FROM ERROR-TEXT (ERROR-INDEX)
 AFTER ADVANCING 1 LINE.

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8BD3

Este módulo lista los números de teléfono de los empleados y, opcionalmente, los actualiza.

```

/*****
*/
/* Module name = DSN8BD3
*/
/* Descriptive name = DB2 SAMPLE APPLICATION
 PHONE APPLICATION
 BATCH
 C LANGUAGE
*/
/* LICENSED MATERIALS - PROPERTY OF IBM
 5695-DB2
 (C) COPYRIGHT 1982, 1995 IBM CORP. ALL RIGHTS RESERVED.
*/
/* STATUS = VERSION 4
*/
/* Function = This module lists employee phone numbers and
 optionally updates them.
*/
/* Notes = none
*/
*/

```

```

/* Module type = C program */
/* Processor = DB2 precompiler, C compiler */
/* Module size = see link edit */
/* Attributes = not reentrant or reusable */
/*
/* Entry point = DSN8BD3 */
/* Purpose = see function */
/* Linkage = invoked from DSN command processor subcommand RUN */
/* Input = */
/*
/* symbolic label/name = CARDIN
/* description = INPUT REQUEST FILE
/*
/* symbolic label/name = VPHONE
/* description = VIEW OF TELEPHONE TABLE: PHONE
/*
/* Output =
/*
/* symbolic label/name = REPORT
/* description = PRINTED REPORT AND RESULTS
/*
/* symbolic label/name = VEMPLP
/* description = VIEW OF EMPLOYEE INFORMATION
/*
/*
/* Exit-normal = return code 0 normal completion */
/*
/* Exit-error =
/*
/* Return code = none
/*
/* Abend codes = none
/*
/* Error-messages =
/*
/* DSN8000I - REQUEST IS: ...
/* DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED
/* DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE
/* DSN8008I - NO EMPLOYEE FOUND IN TABLE
/* DSN8053I - ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED
/* DSN8060E - SQL ERROR, RETURN CODE IS:
/* DSN8061E - ROLLBACK FAILED, RETURN CODE IS:
/* DSN8068E - INVALID REQUEST, SHOULD BE 'L' OR 'U'
/* DSN8075E - MESSAGE FORMAT ROUTINE ERROR,
/* RETURN CODE IS:
/*
/* External references =
/*
/* Routines/services =
/* DSNTIAR - translate sqlca into messages
/*
/* Data-areas = none
/*
/* Control-blocks =
/* SQLCA - sql communication area
/*
/* Tables = none
/*
/* Change-activity =
/*
/* 10/03/94 Updated cardin statement to prevent looping. KEW1351 @51*/
/* PN61293 @51*/
/*
/* *Pseudocode*
/*
/* main:
/* do while more input
/* get input
/* display request
/* process request
/* end
/*
/* Do_req:
/* case (action)
/*
/* subcase ('L')
/* create report heading
/* if lastname is '*' then
/* list all employees
/* else
/* if lastname contains '%' then
/* list employees generic
/* else
/* list employees specific
/* endsub
/*

```

```

/*
/* subcase ('U')
/* update phonenumber for employee
/* write confirmation message
*/
/* otherwise
/* invalid request
/* endsub
*/
/* endcase
*/
/* Prt_row:
/* print a row of the report
*/
/* Sql_err:
/* if sql error occurs then
/* rollback
*/
/*
***** */

/*****
/* Include C library definitions
*/
***** */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*****
/* General declarations
*/
***** */
#define NOTFOUND 100

/*****
/* Input / Output files
*/
***** */
FILE *cardin; /* Input control cards */
FILE *report; /* Output phone report */

/*****
/* Input record structure
*/
***** */
EXEC SQL BEGIN DECLARE SECTION;
struct {
 char action[2]; /* L for list or U for update */
 char lname[16]; /* last name or pattern- L mode*/
 char fname[13]; /* first name or pattern-L mode*/
 char eno[7]; /* employee number- U mode */
 char newno[5]; /* new phone number- U mode */
 } ioarea;

char trail[43]; /* unused portion of input rec */
char slname[16]; /* unmodified last name pattern*/
EXEC SQL END DECLARE SECTION;

/*****
/* Report headings
*/
***** */
struct {
 char hdr011[30];
 char hdr012[32];
 } hdr0 = {
 " REQUEST LAST NAME ",
 "FIRST NAME EMPNO NEW XT.NO"};
#define rpthdr0 hdr0.hdr011

struct {
 char hdr111[29];
 char hdr112[21];
 char hdr113[30];
 } hdr1 = {
 "-----",
 " TELEPHONE DIRECTORY ",
 "-----"};
#define rpthdr1 hdr1.hdr111

struct {
 char hdr211[10];
 char hdr212[11];
 char hdr213[8];
 char hdr214[6];
 char hdr215[9];
 char hdr216[5];
}

```

```

char hdr217[5];
char hdr221[7];
char hdr222[7];
char hdr223[5];
char hdr224[5];
char hdr225[5];
} hdr2 = {
 "LAST NAME",
 "FIRST NAME",
 "INITIAL",
 "PHONE",
 "EMPLOYEE",
 "WORK",
 "WORK",
 "NUMBER",
 "NUMBER",
 "DEPT",
 "DEPT",
 "NAME"
};

#define rpthdr2 hdr2.hdr211,hdr2.hdr212,hdr2.hdr213,hdr2.hdr214,\n
 hdr2.hdr215,hdr2.hdr216,hdr2.hdr217,hdr2.hdr221,\n
 hdr2.hdr222,hdr2.hdr223,hdr2.hdr224,hdr2.hdr225\n

/**/\n
/* Report formats */\n
/**/\n
static char fmt1[] = "\n %s\n";
static char fmt2[] = " %s%17s%10s%6s%10s%5s%5s\n%43s%7s%7s%5s%5s\n";
static char fmt3[] = " %-16s%-16s%-5s%-7s%-9s%-5s%-36s\n";
static char fmt4[] = " %1c%15c%12c%6c%4c%43c";
static char fmt5[] =
 "\n\n %s\n %s\n --%-7s--%-15s--%-12s--%-5s--%-9s--\n";

/**/\n
/* Fields sent to message routine */\n
/**/\n
char outmsg[70]; /* error/information msg buffer*/
char module[8] = "DSN8BD3"; /* module name for message rtn */

extern DSN8MDG(); /* message routine */

/**/\n
/* SQL communication area */\n
/**/\n
EXEC SQL INCLUDE SQLCA;

/**/\n
/* SQL declaration for view VPHONE */\n
/**/\n
EXEC SQL DECLARE VPHONE TABLE
(
 LASTNAME VARCHAR(15) NOT NULL,
 FIRSTNAME VARCHAR(12) NOT NULL,
 MIDDLEINITIAL CHAR(1) NOT NULL,
 PHONENUMBER CHAR(4),
 EMPLOYEENUMBER CHAR(6) NOT NULL,
 DEPTNUMBER CHAR(3) NOT NULL,
 DEPTNAME VARCHAR(36) NOT NULL
);

/**/\n
/* Structure for pphone record */\n
/**/\n
/* Note: since the sample program data does not contain imbedded */
/* nulls, the C language null terminated string can be used to */
/* receive the varchar fields from DB2. */

EXEC SQL BEGIN DECLARE SECTION;
struct {
 char lastname[16];
 char firstname[13];
 char middleinitial[2];
 char phonenumber[5];
 char employeenumber[7];
 char deptnumber[4];
 char deptname[37];
} pphone;
EXEC SQL END DECLARE SECTION;

/**/\n
/* SQL declaration for view VEMPLP (used for update processing) */\n
/**/

```

```

EXEC SQL DECLARE VEMPLP TABLE
 (EMPLOYEENUMBER CHAR(6) NOT NULL,
 PHONENUMBER CHAR(4));
/******
* Structure for pemplp record
***** */
/******
EXEC SQL BEGIN DECLARE SECTION;
struct {
 char employeeNumber[7];
 char phoneNumber[5];
} pemplp;
EXEC SQL END DECLARE SECTION;

/******
/* SQL cursors
***** */
/* cursor to list all employee names */
EXEC SQL DECLARE TELE1 CURSOR FOR
 SELECT *
 FROM VPHONE;

/* cursor to list all employee names with a pattern */
/* (%) or (_) in last name */
EXEC SQL DECLARE TELE2 CURSOR FOR
 SELECT *
 FROM VPHONE
 WHERE LASTNAME LIKE :lname;

/* cursor to list all employees with a specific last name */
EXEC SQL DECLARE TELE3 CURSOR FOR
 SELECT *
 FROM VPHONE
 WHERE LASTNAME = :slname
 AND FIRSTNAME LIKE :fname;

/******
/* SQL return code handling
***** */
/******
EXEC SQL WHENEVER SQLERROR GOTO DBERROR;
EXEC SQL WHENEVER SQLWARNING GOTO DBERROR;
EXEC SQL WHENEVER NOT FOUND CONTINUE;

/******
/* main program routine
***** */
extern main()
{
 /* Open the input and output files */
 cardin = fopen("DD:CARDIN","r",recfm=FB,lrecl=80,blksize=80"); /*@51*/
 report = fopen("DD:REPORT","w");

 /* While more input, process */
 while (!feof(cardin))
 {
 /* Read the next request */
 if (fscanf(cardin, fmt4,
 ioarea.action,
 ioarea.lname,
 ioarea.fname,
 ioarea.eno,
 ioarea.newno,
 trail) == 6)
 {
 /* Display the request */
 DSN8MDG(module, "000I", outmsg);
 fprintf(report, fmt5,
 outmsg,
 rpthdr0,
 ioarea.action,
 ioarea.lname,
 ioarea.fname,
 ioarea.eno,
 ioarea.newno);
 Do_req();
 }
 } /* endwhile */
 fclose(report);
} /* end main */

/******
/* Process the current request
***** */

```

```

/***********************/
Do_req()
{
 char *blankloc; /* string translation pointer */
 strcpy(slname, ioarea.lname); /* save untranslated last name */
 while (blankloc = strpbrk(ioarea.lname, " "))
 blankloc = '%'; / translate blanks into % */
 while (blankloc = strpbrk(ioarea.fname, " "))
 blankloc = '%'; / translate blanks into % */

 /* Determine request type */
 switch (ioarea.action[0])
 {

 /* Process LIST request */
 case 'L':
 /* Print the report headings */
 fprintf(report, fmt1, rpthdr1);
 fprintf(report, fmt2, rpthdr2);

 /* List all employees */
 if (!strcmp(slname, "*"))
 EXEC SQL OPEN TELE1;
 EXEC SQL FETCH TELE1 INTO :pphone;
 if (sqlca.sqlcode == NOTFOUND){ /* If no employees */
 DSN8MDG(module, "008I", outmsg); /* found, display */
 fprintf(report, "%s\n", outmsg); /* error message */
 } /* endif */
 while (sqlca.sqlcode == 0){
 Prt_row();
 EXEC SQL FETCH TELE1 INTO :pphone;
 } /* endwhile */
 EXEC SQL CLOSE TELE1;

 /* List generic employees */
 } else {
 if (strpbrk(slname, "%")){
 EXEC SQL OPEN TELE2;
 EXEC SQL FETCH TELE2 INTO :pphone;
 if (sqlca.sqlcode == NOTFOUND){ /* If no employees */
 DSN8MDG(module, "008I", outmsg); /* found, display */
 fprintf(report, "%s\n", outmsg); /* error message */
 } else {
 while (sqlca.sqlcode == 0){
 Prt_row();
 EXEC SQL FETCH TELE2 INTO :pphone;
 } /* endwhile */
 } /* endif */
 EXEC SQL CLOSE TELE2;
 }

 /* List specific employee */
 } else {
 EXEC SQL OPEN TELE3;
 EXEC SQL FETCH TELE3 INTO :pphone;
 if (sqlca.sqlcode == NOTFOUND){ /* If no employee */
 DSN8MDG(module, "008I", outmsg); /* found, display */
 fprintf(report, "%s\n", outmsg); /* error message */
 } else {
 while (sqlca.sqlcode == 0){
 Prt_row();
 EXEC SQL FETCH TELE3 INTO :pphone;
 } /* endwhile */
 } /* endif */
 EXEC SQL CLOSE TELE3;
 } /* endif */
 break; /* end of 'L' request */

 /* Update an employee phone number */
 case 'U':
 EXEC SQL UPDATE VEMPLP
 SET PHONENUMBER = :ioarea.newno
 WHERE EMPLOYEENUMBER = :ioarea.eno;
 if (sqlca.sqlcode == 0){ /* If employee */
 DSN8MDG(module, "004I", outmsg); /* updated, display */
 fprintf(report, "%s\n", outmsg); /* confirmation msg */
 } else {
 DSN8MDG(module, "007E", outmsg); /* otherwise, display */
 fprintf(report, "%s\n", outmsg); /* error message */
 } /* endif */
 break;
 }
}

```

```

/* Invalid request type */
default:
 DSN8MDG(module, "068E", outmsg); /* Display error msg */
 fprintf(report, "%s\n", outmsg);
} /* endswitch */
return;

DBERROR:
 Sql_err();
} /* end Do_req */

/***/
/* Print a single employee on the report */
/***/
Prt_row()
{
 fprintf(report, fmt3, pphone.lastname,
 pphone.firstname,
 pphone.middleinitial,
 pphone.phonenumber,
 pphone.employeenumber,
 pphone.deptnumber,
 pphone.deptname);
}

/***/
/* SQL error handler */
/***/
#pragma linkage(dsntiar, OS)
Sql_err()
#define data_len 120
#define data_dim 10
struct error_struct {
 short int error_len;
 char error_text[data_dim][data_len];
 } error_message = {data_dim * data_len};
extern short int dsntiar(struct sqlca *sqlca,
 struct error_struct *msg,
 int *len);

short int rc;
int i;
static int lrecl = data_len;

DSN8MDG(module, "060E", outmsg);
fprintf(report, "%s %i\n", outmsg, sqlca.sqlcode);
rc = dsntiar(&sqlca, &error_message, &lrecl); /* Format the sqlca */
if (rc == 0){ /* Print formatted */
 for (i=0;i<=7;i++){ /* sqlca */
 fprintf(report, "%.120s\n", error_message.error_text [i]);
 } /* endfor */
} else {
 DSN8MDG(module, "075E", outmsg);
 fprintf(report, "%s %hi\n", outmsg, rc);
} /* endif */

/* Attempt to rollback any work already done */
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER NOT FOUND CONTINUE;

EXEC SQL ROLLBACK;
if (sqlca.sqlcode == 0){ /* If rollback */
 DSN8MDG(module, "053I", outmsg); /* completed, display*/
 fprintf(report, "%s\n", outmsg); /* confirmation msg */
} else { /* otherwise, display*/
 DSN8MDG(module, "061E", outmsg); /* error message */
 fprintf(report, "%s %i\n", outmsg, sqlca.sqlcode);
} /* endif */
fclose(report);
exit(0);
} /* end of Sql_err */

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8BE3

Este módulo utiliza la clase emp\_db2 para listar o actualizar números de teléfono de empleados de una base de datos de Db2.

```

/*
/* Module name = DSN8BD3
/*
/* Descriptive name = DB2 SAMPLE APPLICATION
/* PHONE APPLICATION
/* BATCH
/* C++ LANGUAGE
/*
/* LICENSED MATERIALS - PROPERTY OF IBM
/* 5625-DB2
/* (C) COPYRIGHT 1982, 2003 IBM CORP. ALL RIGHTS RESERVED.
/*
/* STATUS = VERSION 8
/*
/* Function = This module uses the class emp_db2 to list or update
/* employee phone numbers from a DB2 database
/*
/* Module type = C++ program
/* Processor = DB2 precompiler, C++ compiler
/* Module size = see link edit
/* Attributes = not reentrant or reusable
/*
/* Entry point = DSN8BD3
/* Purpose = see function
/* Linkage = invoked from DSN command processor subcommand RUN
/*
/* Input = symbolic label/name = CARDIN
/* description = INPUT REQUEST FILE
/*
/* Output = symbolic label/name = REPORT
/* description = PRINTED REPORT AND RESULTS
/*
/* Exit-normal = return code 0 normal completion
/*
/* Exit-error =
/*
/* Return code = none
/*
/* Abend codes = none
/*
/* Error-messages =
/* DSN8000I - REQUEST IS: ...
/* DSN8068E - INVALID REQUEST, SHOULD BE 'L' OR 'U'
/* RETURN CODE IS:
/*
/* External references =
/* Routines/services = none
/*
/* Data-areas = none
/*
/* Control-blocks = none
/*
/* Tables = none
/*
/* Change-activity =
/* 02/05/96 Katja KFD0024 C++ sample (D9031)
/* Created based on C sample
/*

#include <string.h>

/* Include emp_db2 C++ class definition
/* (includes other global declarations)

#include "DSN8BEH"

/* Input record structure

struct {
 char action[2]; /* L for list or U for update */
 char lname[16]; /* last name or pattern- L mode */
 char fname[13]; /* first name or pattern-L mode */

```

```

char eno[7]; /* employee number- U mode */
char newno[5]; /* new phone number- U mode */
} ioarea;

char slname[16]; /* unmodified last name pattern */
class emp_db2 proc1; /* DB2 employee object */

/***/
/* Function to process the current request */
/***/
void Do_req(FILE *outfile)
{
 char *blankloc; /* string translation pointer */

/***/
/* Report headings */
/***/
struct
{
 char hdr111[30];
 char hdr112[22];
 char hdr113[30];
} hdr1 = {
 "-----",
 " TELEPHONE DIRECTORY ",
 "-----"};
#define rpthdr1 hdr1.hdr111,hdr1.hdr112,hdr1.hdr113

struct
{
 char hdr211[10];
 char hdr212[11];
 char hdr213[8];
 char hdr214[6];
 char hdr215[9];
 char hdr216[5];
 char hdr217[5];
 char hdr221[7];
 char hdr222[7];
 char hdr223[5];
 char hdr224[5];
 char hdr225[5];
} hdr2 = {
 "LAST NAME",
 "FIRST NAME",
 "INITIAL",
 "PHONE",
 "EMPLOYEE",
 "WORK",
 "WORK",
 "NUMBER",
 "NUMBER",
 "DEPT",
 "DEPT",
 "NAME"};
#define rpthdr2 hdr2.hdr211,hdr2.hdr212,hdr2.hdr213,hdr2.hdr214,\
 hdr2.hdr215,hdr2.hdr216,hdr2.hdr217,hdr2.hdr221,\
 hdr2.hdr222,hdr2.hdr223,hdr2.hdr224,hdr2.hdr225

/***/
/* Report formats */
/***/
static char fmt1[] = "\n %s\n %s\n %s\n";
static char fmt2[] =
 " %9s%17s%10s%6s%10s%5s%5s\n%43s%8s%7s%5s%5s\n";

/***/
/* Start processing input record */
/***/
strcpy(slname, ioarea.lname); /* save untranslated last name */
while (blankloc = strpbirk(ioarea.lname, " "))
 blankloc = '%'; / translate blanks into % */
while (blankloc = strpbirk(ioarea.fname, " "))
 blankloc = '%'; / translate blanks into % */

/* Determine request type */
switch (ioarea.action[0])
{
 /* Process LIST request */
 case 'L':
 /* Print the report headings */

```

```

fprintf(outfile, fmt1, rpthdr1);
fprintf(outfile, fmt2, rpthdr2);

if (!strcmp(slname,"*")) /*)
 /* List all employees */
 proc1.Listall(outfile);
else
{
 if (strpbrk(slname, "%"))
 /* List generic employees */
 proc1.Listsome(outfile,ioarea.lname);
 else
 /* List specific employee */
 proc1.Listone(outfile,slname,ioarea.fname);
} /* else - list selected employees */
break; /* end 'L' request */

/* Update an employee phone number */
case 'U':
 proc1.Empupdate(outfile,ioarea.newno,ioarea.eno);
 break;

/* Invalid request type */
default:
 DSN8MDG(module, "068E", outmsg); /* Display error msg */
 fprintf(outfile, "%s\n", outmsg);
} /* endswitch */
return;
} /* end Do_req */

/***/
/* Function to read a request from an open file */
/***/
int Read_req(FILE *infile)
{
 static char fmt4[] = " %1c%15c%12c%6c%4c%43c"; /* input format */
char trail[43]; /* unused part of input record */
char *newlloc; /* addr of newline char in field */

strcpy(ioarea.action, " ");
strcpy(ioarea.lname, " ");
strcpy(ioarea.fname, " ");
strcpy(ioarea.eno, " ");
strcpy(ioarea.newno, " ");
/* Read the next request */
if (fscanf(infile, fmt4,
 ioarea.action,
 ioarea.lname,
 ioarea.fname,
 ioarea.eno,
 ioarea.newno,
 trail) == 6)
{
 if ((newlloc = strpbrk(ioarea.lname, "\n")) != NULL)
 newlloc = ' '; / change to blank for now */
 if ((newlloc = strpbrk(ioarea.fname, "\n")) != NULL)
 newlloc = ' '; / change to blank for now */
 ioarea.eno[6] = '\0';
 ioarea.newno[4] = '\0';
 return 0;
}
else
 return 1;
} /* end Read_req */

/***/
/* Function to echo a request */
/***/
void Echo_req(FILE *outfile)
{
 /* Local declarations */
 struct /* report header */
 {
 char hdr011[31];
 char hdr012[33];
 } hdr0 = {
 " REQUEST LAST NAME ",
 "FIRST NAME EMPNO NEW XT.NO"};
#define rpthdr0 hdr0.hdr011

static char fmt5[] = /* output format */
"\n\n %s\n %s%s\n ---7s---15s---12s---7s---9s---\n";

```

```

/* End local declarations */

/* Display the request */
DSN8MDG(module, "000I", outmsg);
fprintf(outfile, fmt5,
 outmsg,
 hdr0.hdr011,
 hdr0.hdr012,
 ioarea.action,
 ioarea.lname,
 ioarea.fname,
 ioarea.eno,
 ioarea.newno);

return;
} /* end Echo_req */

/**
/* main program routine
/**
extern main()
{
 int retcode;
 FILE *cardin; /* Input control cards */
 FILE *report; /* Output phone report */

 /* Open the input and output files */
 cardin = fopen
 ("DD:CARDIN", "r", recfm=fb, lrecl=80, blksize=80");
 report = fopen("DD:REPORT", "w");

 /* While more input, process */
 while (!feof(cardin))
 {
 /* Read the next request */
 retcode = Read_req(cardin);
 if (retcode == 0)
 {
 /* Display the request */
 Echo_req(report);
 Do_req(report);
 }
 } /* endwhile */
 fclose(report);
} /* end main */

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8BP3

ESTE MÓDULO LISTA LOS NÚMEROS DE TELÉFONO DE EMPLEADOS Y LOS ACTUALIZA SI LO DESEA.

```

DSN8BP3: PROC REORDER OPTIONS(MAIN);
/**
* *
* MODULE NAME = DSN8BP3 *
* *
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION *
* PHONE APPLICATION *
* BATCH *
* PL/I *
* *
* LICENSED MATERIALS - PROPERTY OF IBM *
* 5695-DB2 *
* (C) COPYRIGHT 1982, 1995 IBM CORP. ALL RIGHTS RESERVED. *
* *
* STATUS = VERSION 4 *
* *
* FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND *
* UPDATES THEM IF DESIRED. *
* *
* NOTES = NONE *
* *
* MODULE TYPE = PL/I PROC OPTIONS(MAIN) *
* PROCESSOR = DB2 PRECOMPILER, PL/I OPTIMIZER *
* MODULE SIZE = SEE LINK EDIT *
* *

```

```

* ATTRIBUTES = REENTRANT
*
* ENTRY POINT = DSN8BP3
* PURPOSE = SEE FUNCTION
* LINKAGE = INVOKED FROM DSN RUN
* INPUT =
*
* SYMBOLIC LABEL/NAME = CARDIN
* DESCRIPTION = INPUT REQUEST FILE
*
* SYMBOLIC LABEL/NAME = VPHONE
* DESCRIPTION = VIEW OF TELEPHONE INFORMATION
*
* OUTPUT =
*
* SYMBOLIC LABEL/NAME = REPORT
* DESCRIPTION = REPORT OF EMPLOYEE PHONE NUMBERS
*
* SYMBOLIC LABEL/NAME = VEMPLP
* DESCRIPTION = VIEW OF EMPLOYEE INFORMATION
*
*
* EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION
*
* EXIT-ERROR =
*
* RETURN CODE = NONE
*
* ABEND CODES = NONE
*
* ERROR-MESSAGES =
* DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED
* DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE
* DSN8008I - NO EMPLOYEE FOUND IN TABLE
* DSN8053I - ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED
* DSN8060E - SQL ERROR, RETURN CODE IS:
* DSN8061E - ROLLBACK FAILED, RETURN CODE IS:
* DSN8068E - INVALID REQUEST, SHOULD BE 'L' OR 'U'
* DSN8075E - MESSAGE FORMAT ROUTINE ERROR,
* RETURN CODE IS :
*
* EXTERNAL REFERENCES =
* ROUTINES/SERVICES =
* DSN8MPG - ERROR MESSAGE ROUTINE
*
* DATA-AREAS = NONE
*
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA
*
* TABLES = NONE
*
* CHANGE-ACTIVITY = NONE
*
*
* *PSEUDOCODE*
*
* PROCEDURE
* GET FIRST INPUT
* DO WHILE MORE INPUT
* CREATE REPORT HEADING
* CASE (ACTION)
*
* SUBCASE ('L')
* IF LASTNAME IS '*' THEN
* LIST ALL EMPLOYEES
* ELSE
* IF LASTNAME CONTAINS '%' THEN
* LIST EMPLOYEES GENERIC
* ELSE
* LIST EMPLOYEES SPECIFIC
* ENDIF
* ENDIF
* ENDIF
*
* SUBCASE ('U')
* UPDATE PHONENUMBER FOR EMPLOYEE
* WRITE CONFIRMATION MESSAGE
* OTHERWISE
* INVALID REQUEST
* ENDSUB
*
* ENDCASE
* GET NEXT INPUT

```

```

* END *
* *
* IF SQL ERROR OCCURS THEN *
* ROLLBACK *
* END. *
* *
-----/ *
1*****/* INPUT/OUTPUT FILES */
*****/ *

DCL CARDIN FILE STREAM INPUT; /* INPUT CONTROL CARDS */
DCL REPORT FILE STREAM OUTPUT PRINT; /* OUTPUT PHONE REPORT */

*****/* ENDFILE HANDLING */
*****/ *

ON ENDFILE (CARDIN) EOF = '1'B;

*****/* STRUCTURE FOR INPUT */
*****/ *

DCL 1 IOAREA,
 2 ACTION CHAR(1), /* ACTION */
 2 LNAME CHAR(15), /* LAST NAME */
 2 FNAME CHAR(12), /* FIRST NAME */
 2 ENO CHAR(6), /* EMPLOYEE NUMBER */
 2 NEWNO CHAR(4); /* PHONE NUMBER */

*****/* WORK VARIABLES */
*****/ *

DCL LNAMEWK CHAR(15) VAR; /* WORK VERSION OF LAST NAME */
DCL FNAMEWK CHAR(12) VAR; /* WORK VERSION OF FIRST NAME */

*****/* REPORT HEADER STRUCTURE */
*****/ *

DCL 1 REPHDR1 STATIC,
 2 HDR111 CHAR(29) INIT ((29)'-'),
 2 HDR112 CHAR(21) INIT (' TELEPHONE DIRECTORY '),
 2 HDR113 CHAR(28) INIT ((28)'-');

DCL 1 REPHDR2 STATIC,
 2 HDR211 CHAR(9) INIT ('LAST NAME'),
 2 HDR212 CHAR(10) INIT ('FIRST NAME'),
 2 HDR213 CHAR(7) INIT ('INITIAL'),
 2 HDR214 CHAR(5) INIT ('PHONE'),
 2 HDR215 CHAR(8) INIT ('EMPLOYEE'),
 2 HDR216 CHAR(4) INIT ('WORK'),
 2 HDR217 CHAR(4) INIT ('WORK'),
 2 HDR221 CHAR(6) INIT ('NUMBER'),
 2 HDR222 CHAR(6) INIT ('NUMBER'),
 2 HDR223 CHAR(4) INIT ('DEPT'),
 2 HDR224 CHAR(4) INIT ('DEPT'),
 2 HDR225 CHAR(4) INIT ('NAME');

*****/* REPORT FORMATS */
*****/ *

L1: FORMAT (A(29),A(28));
L2: FORMAT (SKIP(2),A(9),X(7),A(10),X(3),A(7),X(1),A(5),X(2),A(8),
 X(1),A(4),X(1),A(4),SKIP,X(37),A(6),X(1),A(6),X(3),
 A(4),X(1),A(4),X(1),A(4));
L3: FORMAT (SKIP,A(15),X(1),A(12),X(4),A(1),X(4),A(4),X(3),A(6),X(3),
 A(3),X(2),A(36));
L4: FORMAT (COL(1),A(1),A(15),A(12),A(6),A(4));

*****/* FIELDS SENT TO MESSAGE ROUTINE*/
*****/ *

DCL OUTMSG CHAR(69);
DCL MODULE CHAR(07) INIT('DSN8BP3');

DCL DSN8MPG EXTERNAL ENTRY;

*****/ *

```

```

/* GENERAL DECLares */
/*****/

DCL (ADDR,
 DIM,
 PLIRETV,
 TRANSLATE,
 INDEX) BUILTIN;
DCL EOF BIT(1) INIT ('0'B);
DCL I BIN FIXED(15);
DCL ZERO BIN FIXED(15) STATIC INIT(0);
DCL ONE BIN FIXED(15) STATIC INIT(1);
DCL NOTFOUND BIN FIXED(15) STATIC INIT(100);

1/*****
/* SQL DECLARATION FOR VIEW VPHONE */
/*****

EXEC SQL DECLARE VPHONE TABLE
 (LASTNAME VARCHAR(15) NOT NULL,
 FIRSTNAME VARCHAR(12) NOT NULL,
 MIDDLEINITIAL CHAR(1) NOT NULL,
 PHONENUMBER CHAR(4),
 EMPLOYEENUMBER CHAR(6) NOT NULL,
 DEPTNUMBER CHAR(3) NOT NULL,
 DEPTNAME VARCHAR(36) NOT NULL);

/*****
/* SQL COMMUNICATION AREA */
/*****

EXEC SQL INCLUDE SQLCA;

/*****
/* STRUCTURE FOR PPHONE RECORD */
/*****

DCL 1 PPHONE,
 2 LASTNAME CHAR(15) VAR,
 2 FIRSTNAME CHAR(12) VAR,
 2 MIDDLEINITIAL CHAR(1),
 2 PHONENUMBER CHAR(4),
 2 EMPLOYEENUMBER CHAR(6),
 2 DEPTNUMBER CHAR(3),
 2 DEPTNAME CHAR(36) VAR;

/*****
/* SQL DECLARATION FOR VIEW VEMPLP */
/*****

EXEC SQL DECLARE VEMPLP TABLE
 (EMPLOYEENUMBER CHAR(6) NOT NULL,
 PHONENUMBER CHAR(4));
;

/*****
/* STRUCTURE FOR PEMPLP RECORD */
/*****

DCL 1 PEMPLP,
 2 EMPLOYEENUMBER CHAR(6),
 2 PHONENUMBER CHAR(4);

/*****
/* SQL CURSORS */
/*****

/* CURSOR LISTS ALL EMPLOYEE NAMES */

EXEC SQL DECLARE TELE1 CURSOR FOR
 SELECT *
 FROM VPHONE;

/* CURSOR LISTS ALL EMPLOYEE NAMES WITH A PATTERN (%) OR _ */
/* IN LAST NAME OR A BLANK LAST NAME. */

EXEC SQL DECLARE TELE2 CURSOR FOR
 SELECT *
 FROM VPHONE
 WHERE LASTNAME LIKE :LNAMEWK
 AND FIRSTNAME LIKE :FNAMEWK;

/* CURSOR LISTS ALL EMPLOYEES WITH A SPECIFIC LAST NAME */

```

```

EXEC SQL DECLARE TELE3 CURSOR FOR
 SELECT *
 FROM VPHONE
 WHERE LASTNAME = :LNAMEWK
 AND FIRSTNAME LIKE :FNAMEWK;
/*****
/* SQL RETURN CODE HANDLING */
*****/

EXEC SQL WHENEVER SQLERROR GOTO DBERROR;
EXEC SQL WHENEVER SQLWARNING GOTO DBERROR;
EXEC SQL WHENEVER NOT FOUND CONTINUE;
1/*****
/* MAIN PROGRAM ROUTINE */
*****/
 GET FILE (CARDIN) EDIT (IOAREA) (R(L4)); /* READ FIRST REQUEST */
 /* PROCESS INPUT REQUESTS */
 DO WHILE (^EOF); /* CONTINUE WHILE MORE TO DO */
 /* PUT REPORT HEADINGS */
/*****
/* CREATE REPORT HEADING */
/* SELECT ACTION */
*****/
 PUT FILE (REPORT) PAGE EDIT (REPHDR1) (R(L1));
 PUT FILE (REPORT) EDIT (REPHDR2) (R(L2));
 IF INDEX(LNAME, ' ') > 0 THEN
 LNAMEWK = SUBSTR(LNAME,1,INDEX(LNAME,' ')-1);
 ELSE
 LNAMEWK = LNAME;
 IF INDEX(FNAME, ' ') > 0 THEN
 FNAMEWK = SUBSTR(FNAME,1,INDEX(FNAME,' ')-1);
 ELSE
 FNAMEWK = FNAME;
 /* GET WORKING VERSIONS OF */
 /* LAST AND FIRST NAMES WITH */
 /* NO TRAILING BLANKS */
 IF LNAME = ' ' THEN LNAMEWK='%; /* BLANK NAMES IN INPUT MEAN */
 IF FNAME = ' ' THEN FNAMEWK='%; /* SEARCH FOR ALL NAMES */

 SELECT (ACTION); /* DETERMINE INPUT REQUEST */

/*****
/* LIST ALL EMPLOYEES */
*****/
 WHEN ('L') DO; /* LIST EMPLOYEES */
 IF LNAME = '*' THEN /* LIST ALL EMPLOYEES */
 DO;
 EXEC SQL OPEN TELE1; /* OPEN CURSOR FOR SEARCH */
 EXEC SQL FETCH TELE1 INTO :PPHONE; /* GET FIRST RECORD */

 IF SQLCODE = NOTFOUND THEN /* NO RECORDS FOUND */
 DO; /* GET ERROR MESSAGE */
 CALL DSN8MPG (MODULE, '008I', OUTMSG);
 PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A);
 END;

 /* GET AND PRINT ALL RECORDS */
 DO WHILE (SQLCODE = ZERO);
 PUT FILE (REPORT) EDIT (PPHONE) (R(L3));
 EXEC SQL FETCH TELE1 INTO :PPHONE; /* GET NEXT RECORD */
 END; /* END DO WHILE */

 EXEC SQL CLOSE TELE1; /* CLOSE CURSOR FOR SEARCH */
 END; /* END DO IF */

/*****
/* LIST GENERIC EMPLOYEES */
*****/
 ELSE /* SELECT EMPLOYEES BY NAME */
 DO; /* SEARCH ON PART OF NAME? */
 IF INDEX(LNAMEWK, '%') > ZERO THEN
 DO; /* YES: SEARCH ON PART OF */
 /* LAST NAME */
 EXEC SQL OPEN TELE2; /* OPEN CURSOR FOR SEARCH */
 EXEC SQL FETCH TELE2 INTO :PPHONE; /* GET 1ST RECORD */

 IF SQLCODE = NOTFOUND THEN /* NO RECORDS FOUND */
 DO; /* GET ERROR MESSAGE */
 CALL DSN8MPG (MODULE, '008I', OUTMSG);
 PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A);
 END;

```

```

 /* GET AND PRINT ALL RECORDS */
DO WHILE (SQLCODE = ZERO);
 PUT FILE (REPORT) EDIT (PPHONE) (R(L3));
 EXEC SQL FETCH TELE2 INTO :PPHONE; /*GET NEXT RECORD*/
END; /* END DO WHILE */
EXEC SQL CLOSE TELE2; /* CLOSE CURSOR FOR SEARCH */
END; /* END DO IF */

/***/
/* LIST SPECIFIC EMPLOYEES */
/***/
ELSE /* NO - SEARCH ON LAST NAME */
DO; /* & OPTIONAL FIRST NAME */
 /* SEE IF FIRST NAME ENTERED */
 /* IF NOT SET UP FOR ALL NAMES*/
EXEC SQL OPEN TELE3; /* OPEN CURSOR FOR SEARCH */
EXEC SQL FETCH TELE3 INTO :PPHONE; /* GET 1ST RECORD */

IF SQLCODE = NOTFOUND THEN /* NO RECORDS FOUND*/
DO; /* GET ERROR MESSAGE*/
 CALL DSN8MPG (MODULE, '008I', OUTMSG);
 PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A);
END;

 /* GET AND PRINT ALL RECORDS */
DO WHILE (SQLCODE = ZERO);
 PUT FILE (REPORT) EDIT (PPHONE) (R(L3));
 EXEC SQL FETCH TELE3 INTO :PPHONE; /*GET NEXT RECORD*/
END; /* END DO WHILE */
EXEC SQL CLOSE TELE3; /* CLOSE CURSOR FOR SEARCH*/
END; /* END DO ELSE */
END; /* END DO IF */
/*END WHEN */

/***/
/* UPDATES PHONE NUMBERS FOR EMPLOYEES */
/***/
WHEN ('U') DO; /* TELEPHONE UPDATE */
 EXEC SQL UPDATE VEMPLP
 SET PHONENUMBER = :NEWNO /* CHANGE PHONE NO.*/
 WHERE EMPLOYEENUMBER = :ENO;

IF SQLCODE = ZERO THEN /* WAS UPDATE OK? */
DO;
 CALL DSN8MPG (MODULE, '004I', OUTMSG); /* YES */
 PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A); /* YES */
END; /*EMPLOYEE FOUND*/
/*UPDATE SUCCESSFUL*/

ELSE
DO;
 CALL DSN8MPG (MODULE, '007E', OUTMSG); /*UPDATE FAILED*/
 PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A);
END; /* END DO ELSE*/

END; /* END WHEN */

OTHERWISE /* INVALID REQUEST */
DO;
 CALL DSN8MPG (MODULE, '068E', OUTMSG);
 PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A);
END; /* END OTHERWISE */
/* END SELECT*/

GET FILE (CARDIN) EDIT (IOAREA) (R(L4)); /* READ NEXT REQUEST */
END; /* END EOF */
GOTO PGMEND; /* BYPASS SQL ERRORHANDLING */

/***/
/* SQL ERROR CODE HANDLING */
/***/

DCL
 DSNTIAR ENTRY OPTIONS(ASM,INTER,RETCODE);
DCL
 DATA_LEN FIXED BIN(31) INIT(120);
DCL
 DATA_DIM FIXED BIN(31) INIT(10);
DCL
 1 ERROR_MESSAGE AUTOMATIC,
 3 ERROR_LEN FIXED BIN(15) UNAL INIT((DATA_LEN*DATA_DIM)),

```

```

3 ERROR_TEXT(DATA_DIM) CHAR(DATA_LEN);

/***/
/* SQL ERROR OCCURRED - GET ERROR MESSAGE*/
/***/
DBERROR:
 /* SQL ERROR */
 /* PRINT ERROR MESSAGE*/
CALL DSN8MPG (MODULE, '060E', OUTMSG);
PUT FILE (REPORT) EDIT (OUTMSG,SQLCODE) (SKIP(2),A,F(10));
CALL DSNTIAR(SQLCA , ERROR_MESSAGE , DATA_LEN);

IF PLIRETV = ZERO THEN /*ZERO RETURN CODE FROM DSNTIAR*/
 DO I=ONE TO DIM(ERROR_TEXT,ONE);
 PUT FILE (REPORT) EDIT (ERROR_TEXT(I)) (SKIP,A) ;
 END;

ELSE
 DO;
 CALL DSN8MPG (MODULE, '075E', OUTMSG);
 PUT FILE (REPORT) EDIT /*NON-ZERO RETURN CODE FROM DSNTIAR*/
 /*PRINT ERROR MESSAGE */*
 (OUTMSG, PLIRETV) (SKIP(2), A, F(10)) ;
 END;

/***/
/* SQL RETURN CODE HANDLING WHEN PROCESSING CANNOT PROCEED*/
/***/

EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER NOT FOUND CONTINUE;

EXEC SQL ROLLBACK; /* PERFORM ROLLBACK */

IF SQLCODE = ZERO THEN
 DO; /* ROLLBACK SUCCESSFUL,*/
 /* ALL UPDATES REMOVED */
 CALL DSN8MPG (MODULE, '053I', OUTMSG);
 PUT FILE (REPORT) EDIT (OUTMSG) (SKIP(2),A);
 END;

ELSE
 DO; /* ROLLBACK FAILED,*/
 /* RETURN CODE IS: */
 CALL DSN8MPG (MODULE, '061E', OUTMSG);
 PUT FILE (REPORT) EDIT (OUTMSG,SQLCODE) (SKIP(2),A,F(10));
 END;

PGMEND: /* PROGRAM END */
END;

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8BF3

ESTE MÓDULO LISTA LOS NÚMEROS DE TELÉFONO DE EMPLEADOS Y LOS ACTUALIZA SI LO DESEA.

```

PROGRAM DSN8B3

* MODULE NAME = DSN8BF3, PROGRAM DSN8B3
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
* PHONE APPLICATION
* BATCH
* FORTRAN
* LICENSED MATERIALS - PROPERTY OF IBM
* 5695-DB2
* (C) COPYRIGHT 1982, 1995 IBM CORP. ALL RIGHTS RESERVED.
* STATUS = VERSION 4
* FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND
* UPDATES THEM IF DESIRED.
*
```

```

*
* NOTES = NONE
*
*
* MODULE TYPE = FORTRAN PROGRAM
* PROCESSOR = DB2 PRECOMPLIER, VS FORTRAN
* MODULE SIZE = SEE LINK EDIT
* ATTRIBUTES = NOT REENTRANT OR REUSABLE
*
* ENTRY POINT = DSN8BF3
* PURPOSE = SEE FUNCTION
* LINKAGE = INVOKED FROM DSN RUN
* INPUT =
*
* SYMBOLIC LABEL/NAME = FT05F001
* DESCRIPTION = INPUT REQUEST FILE
*
* SYMBOLIC LABEL/NAME = VPHONE
* DESCRIPTION = VIEW OF TELEPHONE INFORMATION
*
* OUTPUT =
*
* SYMBOLIC LABEL/NAME = FT06F001
* DESCRIPTION = PRINTED REPORT AND RESULTS
*
* SYMBOLIC LABEL/NAME = VEMPLP
* DESCRIPTION = VIEW OF EMPLOYEE INFORMATION
*
*
* EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION
*
* EXIT-ERROR =
*
* RETURN CODE = NONE
*
* ABEND CODES = NONE
*
* ERROR-MESSAGES =
* DSN8000I - REQUEST IS: ...
* DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED
* DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE
* DSN8008I - NO EMPLOYEE FOUND IN TABLE
* DSN8051I - PROGRAM ENDED
* DSN8053I - ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED
* DSN8060E - SQL ERROR, RETURN CODE IS:
* DSN8061E - ROLLBACK FAILED, RETURN CODE IS:
* DSN8068E - INVALID REQUEST, SHOULD BE 'L' OR 'U'
* DSN8075E - MESSAGE FORMAT ERROR,
* RETURN CODE IS:
*
* EXTERNAL REFERENCES =
* ROUTINES/SERVICES =
* DSNTIR - TRANSLATE SQLCA INTO MESSAGES
*
* DATA-AREAS = NONE
*
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA
*
* TABLES = NONE
*
* CHANGE-ACTIVITY = NONE
*
*
* *PSEUDOCODE*
*
* PROCEDURE
* DO WHILE MORE INPUT
* GET INPUT
* CREATE REPORT HEADING
* CASE (ACTION)
*
* SUBCASE ('L')
* IF LASTNAME IS '*' THEN
* LIST ALL EMPLOYEES
* ELSE
* IF LASTNAME CONTAINS '%' THEN
* LIST EMPLOYEES GENERIC
* ELSE
* LIST EMPLOYEES SPECIFIC
* ENDIF
* ENDIF
* ENDSUB
*
```

```

* SUBCASE ('U')
* UPDATE PHONENUMBER FOR EMPLOYEE
* WRITE CONFIRMATION MESSAGE
*
* OTHERWISE
* INVALID REQUEST
* ENDSUB
*
* ENDCASE
* GET NEXT INPUT
* END
*
* IF SQL ERROR OCCURS THEN
* ROLLBACK
* END.
*
*****SQL DECLARATION FOR VIEW VPHONE ****
* SQL DECLARATION FOR VIEW VPHONE *
*****SQL DECLARATION FOR VIEW VPHONE *

EXEC SQL DECLARE VPHONE TABLE
C (LASTNAME VARCHAR(15) NOT NULL,
C FIRSTNAME VARCHAR(12) NOT NULL,
C MIDDLEINITIAL CHAR(1) NOT NULL,
C PHONENUMBER CHAR(4)
C EMPLOYEENUMBER CHAR(6) NOT NULL,
C DEPTNUMBER CHAR(3) NOT NULL,
C DEPTNAME VARCHAR(36) NOT NULL)

*****SQL DECLARATION FOR VIEW VEMPLP ****
* SQL DECLARATION FOR VIEW VEMPLP *
*****SQL DECLARATION FOR VIEW VEMPLP *

EXEC SQL DECLARE VEMPLP TABLE
C (EMPLOYEENUMBER CHAR(6) NOT NULL,
C PHONENUMBER CHAR(4))

*****PPHONE FIELDS ****
* PPHONE FIELDS *
*****PPHONE FIELDS *

CHARACTER * 15 LASTNM
CHARACTER * 12 FIRSTN
CHARACTER * 1 MIDINI
CHARACTER * 4 PHONEN
CHARACTER * 6 EMPNO
CHARACTER * 3 DEPTNO
CHARACTER * 36 DEPTNM

*****INPUT FIELDS ****
* INPUT FIELDS *
*****INPUT FIELDS *

CHARACTER * 1 ACTION
CHARACTER * 15 LNAME
CHARACTER * 12 FNAME
CHARACTER * 6 ENO
CHARACTER * 4 NEWNO
CHARACTER * 15 LNAMEW
CHARACTER * 12 FNAMEW

*****SQL CURSORS ****
* SQL CURSORS *
*****SQL CURSORS *

EXEC SQL DECLARE TELE1 CURSOR FOR
C SELECT *
C FROM VPHONE

EXEC SQL DECLARE TELE2 CURSOR FOR
C SELECT *
C FROM VPHONE
C WHERE LASTNAME LIKE :LNAMEW
C AND FIRSTNAME LIKE :FNAMEW

EXEC SQL DECLARE TELE3 CURSOR FOR
C SELECT *
C FROM VPHONE
C WHERE LASTNAME = :LNAME
C AND FIRSTNAME LIKE :FNAMEW

```

```

* SQL RETURN CODES: OK/NOTFOUND *

INTEGER OK/0/, NOTFND/100/

* REPORT FORMATS AND INPUT

100 FORMAT ('0',A29,A21,A28)
200 FORMAT ('0',A9,7X,A10,3X,A7,1X,A5,2X,A8,
 C 1X,A4,1X,A4,/,38X,A6,1X,A6,3X,
 C A4,1X,A4,1X,A4,/)
300 FORMAT (' ',A15,1X,A12,4X,A1,4X,A4,3X,A6,3X,
 C A3,2X,A36)
400 FORMAT (A1,A15,A12,A6,A4)
500 FORMAT ('0', A)
600 FORMAT ('0', A, I8)
700 FORMAT ('1',A,/,
 C 5X,'REQUEST',2X,'LAST NAME',8X,'FIRST NAME',4X,
 C 'EMPNO',3X,'NEW XT.NO',/,
 C 3X,'--',A1,6X,'--',A15,'--',A12,'--',A6,'--',A4,'--')
800 FORMAT ('1')

* MESSAGES

CHARACTER * 30 DSN800
CHARACTER * 48 DSN804
CHARACTER * 60 DSN868
CHARACTER * 59 DSN807
CHARACTER * 45 DSN860
CHARACTER * 59 DSN853
CHARACTER * 51 DSN861
CHARACTER * 45 DSN808
CHARACTER * 64 DSN875
CHARACTER * 32 DSN851

* VARIABLES USED WITH DSNTIR

INTEGER ERRLEN /960/
CHARACTER*120 ERRTXT(8)

* MISCELLANEOUS VARIABLES

INTEGER I, ICODE
CHARACTER * 15 PERC15

* SQL COMMUNICATION AREA

EXEC SQL INCLUDE SQLCA

* DATA STATEMENTS

DATA PERC15
C/'%%%%%%%%%%%%%' /

DATA DSN800
C/'DSN8000I: DSN8BF3-REQUEST IS:'/
DATA DSN804
C/'DSN8004I: DSN8BF3-EMPLOYEE SUCCESSFULLY UPDATED'/
DATA DSN868
C/'DSN8068E: DSN8BF3-INVALID REQUEST, SHOULD BE ''L'' OR ''U'''/
DATA DSN807
C/'DSN8007E: DSN8BF3-EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE'/
DATA DSN860
C/'DSN8060E: DSN8BF3-SQL ERROR, RETURN CODE IS:'/
DATA DSN853
C/'DSN8053I: DSN8BF3-ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED'/
DATA DSN861
C/'DSN8061E: DSN8BF3-ROLLBACK FAILED, SQLCODE IS:'/

```

```

DATA DSN808
C/'DSN8008I: DSN8BF3-NO EMPLOYEE FOUND IN TABLE'/
DATA DSN875
C/'DSN8075E: DSN8BF3-MESSAGE FORMAT ROUTINE ERROR, RETURN CODE IS:
C'/
DATA DSN851
C/'DSN8051I: DSN8BF3-PROGRAM ENDED'/

* SQL RETURN CODE HANDLING

EXEC SQL WHENEVER SQLERROR GOTO 4000
EXEC SQL WHENEVER SQLWARNING GOTO 4000

* START OF PROGRAM

* CONTINUE WHILE MORE INPUT

1000 CONTINUE

* GET NEXT INPUT

READ (UNIT=05,FMT=400,END=3000) ACTION, LNAME, FNAME, ENO, NEWNO
WRITE (UNIT=06,FMT=700) DSN800, ACTION, LNAME, FNAME, ENO, NEWNO
WRITE (UNIT=06,FMT=800)

* CREATE REPORT HEADING
* SELECT ACTION

* **CREATE REPORT HEADING
WRITE (UNIT=06,FMT=100) '-----',
C 'TELEPHONE DIRECTORY ',
C '-----',
C
WRITE (UNIT=06,FMT=200) 'LAST NAME', 'FIRST NAME', 'INITIAL',
C 'PHONE', 'EMPLOYEE', 'WORK', 'WORK', 'NUMBER',
C 'NUMBER', 'DEPT', 'DEPT', 'NAME'

* **SELECT ACTION
*
* **LIST EMPLOYEES
*
IF (ACTION .EQ. 'L') THEN
GOTO 1010
*
ELSE IF (ACTION .EQ. 'U') THEN
GOTO 1700
*
ELSE
GOTO 1800
END IF

1010 CONTINUE

* ACTION - LIST

IF (LNAME .NE. '*') GOTO 1300

* LIST ALL EMPLOYEES

* **OPEN CURSOR
EXEC SQL OPEN TELE1
NBRETR = 0

1100 CONTINUE

* **GET EMPLOYEE
*
EXEC SQL FETCH TELE1 INTO
C :LASTNM,:FIRSTN,:MIDINI,
C :PHONEN,:EMPNO,:DEPTNO,:DEPTNM

```

```

 IF (SQLCOD .EQ. NOTFND) GO TO 1200

*
* **LIST ALL EMPLOYEES
NBRETR = NBRETR + 1
 WRITE (UNIT=06,FMT=300)
C LASTNM,FIRSTN,MIDINI,
C PHONEN, EMPNO, DEPTNO, DEPTNM
 GO TO 1100

*
* **NO EMPLOYEE FOUND
* **PRINT ERROR MESSAGE
1200 CONTINUE
 IF (NBRETR .EQ. 0) WRITE (UNIT=06,FMT=500) DSN808

*
* **CLOSE CURSOR
 EXEC SQL CLOSE TELE1
 GO TO 1000

* ELSE DETERMINE IF LASTNAME *
* OR FIRSTNAME IS GIVEN *

1300 CONTINUE
 IPOS=INDEX(LNAME, '%')

* REPLACE FIRST BLANK AND FOLLOWING *
* CHARACTERS IN LASTNAME WORK (LNAMEW) *
* WITH CHARACTER % FOR LIKE PREDICATE. *

 IBLANK=INDEX(LNAME, ' ')
 IF (IBLANK .GT. 1) THEN
 LNAMEW = LNAME(1:IBLANK-1)//PERC15(1:15-IBLANK+1)
 ELSE IF (IBLANK .EQ. 1) THEN
 LNAMEW=PERC15
 IPOS = 1
 ELSE
 END IF

* REPLACE FIRST BLANK AND FOLLOWING *
* CHARACTERS IN FIRSTNAME WORK (FNAMEW) *
* WITH CHARACTER % FOR LIKE PREDICATE. *

 IBLANK=INDEX(FNAME, ' ')
 IF (IBLANK .GT. 1) THEN
 FNAMEW = FNAME(1:IBLANK-1)//PERC15(1:12-IBLANK+1)
 ELSE IF (IBLANK .EQ. 1) THEN
 FNAMEW=PERC15(1:12)
 ELSE
 END IF

 IF (IPOS .LE. 0) GOTO 1600

* LIST GENERIC EMPLOYEES *

* **OPEN CURSOR
 EXEC SQL OPEN TELE2
 NBRETR = 0

1400 CONTINUE
* **GET EMPLOYEES
 EXEC SQL FETCH TELE2 INTO
 :LASTNM,:FIRSTN,:MIDINI,
 :PHONEN,:EMPNO,:DEPTNO,:DEPTNM

 IF (SQLCOD .EQ. NOTFND) GO TO 1500

*
* **LIST GENERIC EMPLOYEES
NBRETR = NBRETR + 1
 WRITE (UNIT=06,FMT=300)
C LASTNM,FIRSTN,MIDINI,
C PHONEN, EMPNO, DEPTNO, DEPTNM
 GOTO 1400

```

```

*
* **EMPLOYEE NOT FOUND
* **PRINT ERROR MESSAGE
1500 CONTINUE
 IF (NBRETR .EQ. 0) WRITE (UNIT=06,FMT=500) DSN808
*
* **CLOSE CURSOR
 EXEC SQL CLOSE TELE2
 GOTO 1000

* LIST SPECIFIC EMPLOYEES

1600 CONTINUE

*
* **OPEN CURSOR
 EXEC SQL OPEN TELE3
 NBRETR = 0

1620 CONTINUE
*
* **GET EMPLOYEES
 EXEC SQL FETCH TELE3 INTO
 C :LASTNM,:FIRSTN,:MIDINI,
 C :PHONEN,:EMPNO,:DEPTNO,:DEPTNM

 IF (SQLCOD .EQ. NOTFND) GO TO 1640

*
* **LIST SPECIFIC EMPLOYEES
 NBRETR = NBRETR + 1
 WRITE (UNIT=06,FMT=300)
 C LASTNM,FIRSTN,MIDINI,
 C PHONEN, EMPNO, DEPTNO, DEPTNM
 GO TO 1620

*
* **EMPLOYEE NOT FOUND
* **PRINT ERROR MESSAGE
1640 CONTINUE
 IF (NBRETR .EQ. 0) WRITE (UNIT=06,FMT=500) DSN808
*
* **CLOSE CURSOR
 EXEC SQL CLOSE TELE3
 GO TO 1000

* UPDATE PHONE NUMBERS FOR EMPLOYEES

1700 CONTINUE
*
* **PERFORM UPDATE
 EXEC SQL UPDATE VEMPLP
 C SET PHONENUMBER = :NEWNO
 C WHERE EMPLOYEENUMBER = :ENO

 IF (SQLCOD .EQ. OK) THEN
*
* **UPDATE SUCCESSFUL
* **EMPLOYEE FOUND
* **PRINT CONFIRMATION
* **MESSAGE
 WRITE (UNIT=06,FMT=500) DSN804
 ELSE
*
* **UPDATE FAILED
* **EMPLOYEE NOT FOUND
* **PRINT ERROR MESSAGE
 WRITE (UNIT=06,FMT=500) DSN807
 END IF
 GO TO 1000

*
* ** INVALID REQUEST
* ** PRINT ERROR MESSAGE

1800 CONTINUE
 WRITE (UNIT=06,FMT=500) DSN868
 GO TO 1000

* END OF LOOP
* FOR MORE INPUT

```

```

*
* ** THIS LABEL IS
* ** BRANCHED TO FOR
* ** END OF DATA
3000 CONTINUE
 WRITE (UNIT=06,FMT=800)
 WRITE (UNIT=06,FMT=500) DSN851
 RETURN

* IF SQL ERROR OCCURRED - GET ERROR MESSAGE*

 EXEC SQL WHENEVER SQLERROR CONTINUE
 EXEC SQL WHENEVER SQLWARNING CONTINUE
 EXEC SQL WHENEVER NOT FOUND CONTINUE

4000 CONTINUE
*
* **SQL ERROR
* **PRINT ERROR MESSAGE
 WRITE (UNIT=06,FMT=600) DSN860, SQLCOD
 CALL DSNTIR (ERRLEN, ERRTXT, ICODE)

 IF (ICODE .EQ. OK) THEN

 DO 4100 I=1, 10
 WRITE (UNIT=06,FMT=500) ERRTXT(I)
4100 CONTINUE
 ELSE
*
* **ERROR DETECTED BY
* **MESSAGE FORMAT
* **ROUTINE
* **PRINT ERROR MESSAGE

 WRITE (UNIT=06,FMT=600) DSN875, ICODE

 END IF
*
* **PERFORM ROLLBACK
 EXEC SQL ROLLBACK
 IF (SQLCOD .EQ. OK) THEN

*
* **ROLLBACK SUCCESSFUL
* **PRINT CONFIRMATION
* **MESSAGE

 WRITE (UNIT=06,FMT=500) DSN853
 ELSE
*
* **ROLLBACK FAILED
* **PRINT ERROR MESSAGE

 WRITE (UNIT=06,FMT=600) DSN861, SQLCOD
 END IF
 RETURN
 END

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8HC3

ESTE MÓDULO MUESTRA EL DEPARTAMENTO DE Db2 Y LAS TABLAS DE EMPLEADO Y LAS ACTUALIZA SI LO DESEA.

```

IDENTIFICATION DIVISION.

*----- PROGRAM-ID. DSN8HC3. 00000100
----- 00000200
----- 00000300
----- 00000400
----------* 00000500
----------* 00000600
----------* 00000700
----------* 00000800
----------* 00000900
----------* 00001000
----------* 00001100
----------* 00001200
----------* 00001300
----------* 00001400
----------* 00001500
----------* 00001600

*----- MODULE NAME = DSN8HC3

*----- DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
----------* ORGANIZATION APPLICATION
----------* ISPF
----------* COBOL
----- LICENSED MATERIALS - PROPERTY OF IBM
----- 5615-DB2
----- (C) COPYRIGHT 1982, 2013 IBM CORP. ALL RIGHTS RESERVED.

```

```

* STATUS = VERSION 11 * 00001700
* * 00001800
* * 00001900
* * 00002000
* FUNCTION = THIS MODULE DISPLAYS THE DB2 DEPARTMENT AND * 00002100
* EMPLOYEE TABLES AND UPDATES THEM IF DESIRED. * 00002200
* * 00002300
* NOTES = * 00002400
* DEPENDENCIES = REQUIRED ISPF PANELS: * 00002500
* DSN8SSH * 00002600
* DSN8SSH1 * 00002700
* DSN8SSH2 * 00002800
* DSN8SSH3 * 00002900
* DSN8SSH4 * 00003000
* DSN8SSH5 * 00003100
* RESTRICTIONS = NONE * 00003200
* * 00003300
* MODULE TYPE = VS COBOL II PROGRAM * 00003400
* PROCESSOR = DB2 PRECOMPILER, VS COBOL II * 00003500
* MODULE SIZE = SEE LINKEDIT * 00003600
* ATTRIBUTES = NOT REENTRANT OR REUSABLE * 00003700
* * 00003800
* ENTRY POINT = DSN8HC3 * 00003900
* PURPOSE = SEE FUNCTION * 00004000
* LINKAGE = INVOKED FROM ISPF * 00004100
* * 00004200
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION: * 00004300
* INPUT-MESSAGE: * 00004400
* * 00004500
* SYMBOLIC LABEL/NAME = DSN8SSH * 00004600
* DESCRIPTION = MAIN MENU * 00004700
* * 00004800
* SYMBOLIC LABEL/NAME = DSN8SSH2 * 00004900
* DESCRIPTION = DEPARTMENT PANEL * 00005000
* * 00005100
* SYMBOLIC LABEL/NAME = DSN8SSH3 * 00005200
* DESCRIPTION = SELECT FROM LIST PANEL * 00005300
* * 00005400
* SYMBOLIC LABEL/NAME = DSN8SSH4 * 00005500
* DESCRIPTION = SELECT FROM LIST PANEL * 00005600
* * 00005700
* SYMBOLIC LABEL/NAME = DSN8SSH5 * 00005800
* DESCRIPTION = EMPLOYEE PANEL * 00005900
* * 00006000
* SYMBOLIC LABEL/NAME = VHDEPT * 00006100
* DESCRIPTION = VIEW OF DEPARTMENT DATA * 00006200
* * 00006300
* SYMBOLIC LABEL/NAME = VEMP * 00006400
* DESCRIPTION = VIEW OF EMPLOYEE DATA * 00006500
* * 00006600
* OUTPUT = PARAMETERS EXPLICITLY RETURNED: * 00006700
* OUTPUT-MESSAGE: * 00006800
* * 00006900
* SYMBOLIC LABEL/NAME = DSN8SSH * 00007000
* DESCRIPTION = MAIN MENU PANEL * 00007100
* * 00007200
* SYMBOLIC LABEL/NAME = DSN8SSH1 * 00007300
* DESCRIPTION = DEPARTMENT STRUCTURE PANEL * 00007400
* * 00007500
* SYMBOLIC LABEL/NAME = DSN8SSH2 * 00007600
* DESCRIPTION = DEPARTMENT PANEL * 00007700
* * 00007800
* SYMBOLIC LABEL/NAME = DSN8SSH3 * 00007900
* DESCRIPTION = SELECTION LIST PANEL * 00008000
* * 00008100
* SYMBOLIC LABEL/NAME = DSN8SSH4 * 00008200
* DESCRIPTION = SELECTION LIST PANEL * 00008300
* * 00008400
* SYMBOLIC LABEL/NAME = DSN8SSH5 * 00008500
* DESCRIPTION = EMPLOYEE PANEL * 00008600
* * 00008700
* EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION * 00008800
* * 00008900
* EXIT-ERROR = * 00009000
* RETURN CODE = NONE * 00009100
* * 00009200
* ABEND CODES = NONE * 00009300
* * 00009400
* * 00009500
* ERROR-MESSAGES = DSN8001I - EMPLOYEE NOT FOUND * 00009600
* * 00009700
* * 00009800

```

```

* DSN8002I - EMPLOYEE SUCCESSFULLY ADDED * 00009900
* DSN8003I - EMPLOYEE SUCCESSFULLY ERASED * 00010000
* DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED * 00010100
* DSN8005E - EMPLOYEE EXISTS ALREADY, ADD NOT DONE * 00010200
* DSN8006E - EMPLOYEE DOES NOT EXIST, ERASE NOT DONE * 00010300
* DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE* 00010400
* DSN8011I - DEPARTMENT NOT FOUND * 00010500
* DSN8012I - DEPARTMENT SUCCESSFULLY ADDED * 00010600
* DSN8013I - DEPARTMENT SUCCESSFULLY ERASED * 00010700
* DSN8014I - DEPARTMENT SUCCESSFULLY UPDATED * 00010800
* DSN8015E - DEPARTMENT EXISTS ALREADY, ADD NOT DONE * 00010900
* DSN8016E - DEPARTMENT DOES NOT EXIST, ERASE NOT * 00011000
* DONE * 00011100
* DSN8017E - DEPARTMENT DOES NOT EXIST, UPDATE NOT * 00011200
* DONE * 00011300
* DSN8060E - SQL ERROR, RETURN CODE IS: * 00011400
* DSN8074E - DATA IS TOO LONG FOR SEARCH CRITERIA * 00011500
* DSN8079E - CONNECTION NOT ESTABLISHED * 00011600
* DSN8200E - INVALID DEPARTMENT NUMBER, EMPLOYEE NOT * 00011700
* ADDED * 00011800
* DSN8203E - INVALID WORK DEPT, EMPLOYEE NOT UPDATED * 00011900
* DSN8210E - INVALID MGRNO, DEPARTMENT NOT ADDED * 00012000
* DSN8213E - INVALID ADMIN DEPT ID, DEPARTMENT NOT * 00012100
* ADDED * 00012200
* DSN8214E - INVALID MANAGER ID, DEPARTMENT NOT * 00012300
* UPDATED * 00012400
* DSN8215E - INVALID ADMIN DEPT ID, DEPARTMENT NOT * 00012500
* UPDATED * 00012600
* DSN8216E - DEPT NOT AT SPECIFIED LOCATION, EMPLOYEE* 00012700
* NOT ADDED * 00012800
* DSN8217E - DEPT NOT AT SPECIFIED LOCATION, EMP NOT * 00012900
* UPDATED * 00013000
* * 00013100
* EXTERNAL REFERENCES = * 00013200
* ROUTINES/SERVICES = * 00013300
* DSN8MCG - ERROR MESSAGE ROUTINE * 00013400
* ISLINK - ISPF SERVICES ROUTINE * 00013500
* * 00013600
* DATA-AREAS = * 00013700
* NONE * 00013800
* * 00013900
* CONTROL-BLOCKS = * 00014000
* SQLCA - SQL COMMUNICATION AREA * 00014100
* * 00014200
* TABLES = NONE * 00014300
* * 00014400
* * 00014500
* CHANGE-ACTIVITY = NONE * 00014600
* * 00014700
* * 00014800
* * 00014900
* *PSEUDOCODE*
* SET UP RETURN CODE HANDLING 0000-PROGRAM-START* 00015000
* SET PREVIOUS LOCATION TO LOCAL * 00015100
* DO UNTIL NO MORE TERMINAL INPUT * 00015200
* GET PANEL INPUT 1000-MAIN-LOOP * 00015300
* IF CURRENT AND PREVIOUS LOCATION DIFFER THEN * 00015400
* IF REMOTE LOCATION THEN * 00015500
* CONNECT TO REMOTE LOCATION * 00015600
* ELSE RESET TO LOCAL LOCATION * 00015700
* DETERMINE PROCESSING REQUEST * 00015800
* IF ACTION FIELD ADD: * 00015900
* IF OBJECT FIELD IS DE: * 00016000
* ADD RECORD TO VHDEPT TABLES 2000-ADDDEPT * 00016100
* AT ALL LOCATIONS * 00016200
* ELSE IF OBJECT FIELD IS EM: * 00016300
* ADD RECORD TO VEMP TABLE 3000-ADDEMP * 00016400
* ELSE: * 00016500
* ELSE: * 00016600
* IF OBJECT FIELD DE OR DS: 5000-DEPARTMENT * 00016700
* IF "LIST GENERIC": 5100-GENDEPT * 00016800
* CHOOSE CURSOR BASED ON 5110-GETDEPTTAB * 00016900
* SEARCH CRITERIA AND DATA * 00017000
* CREATE ISPF TABLE * 00017100
* DO UNTIL NO MORE RECORDS: * 00017200
* FETCH RECORD * 00017300
* STORE RECORD IN TABLE * 00017400
* DISPLAY DEPARTMENT LIST 5121-GETDEPT * 00017500
* ON SCREEN * 00017600
* STORE SELECTED DEPT ID IN * 00017700
* HOST VARIABLE * 00017800
* ELSE: * 00017900
* IF OBJFLD IS DE: 5200-DISPLAYDEPT * 00018000

```

```

* FETCH SELECTED DEPT * 00018100
* DISPLAY DEPT ON SCREEN 5210-DISDEPTACT * 00018200
* IF ACTION IS ERASE: 5220-ERASEDEPT * 00018300
* DELETE DEPARTMENT AT 5221-DELDEPTS * 00018400
* ALL LOCATIONS * 00018500
* PERFORM CASCADE DELETE 5223-DELDEPEND * 00018600
* OF DEPENDENT DEPTS * 00018700
* AT ALL LOCATIONS * 00018800
* ELSE IF ACTION IS UPDATE: 5230-UPDATEDEPT * 00018900
* UPDATE DEPARTMENT AT * 00019000
* ALL LOCATIONS * 00019100
* ELSE: * 00019200
* ELSE (OBJFLD IS DS): 5300-STRUCTURE * 00019300
* FETCH SELECTED DEPT * 00019400
* DISPLAY SELECTED DEPT * 00019500
* CREATE ISPF TABLE 5310-DISSTR * 00019600
* DO UNTIL NO MORE RECORDS: 5312-GETSTRTAB * 00019700
* FETCH DEPT REPORTING TO * 00019800
* SELECTED DEPT * 00019900
* STORE RECORD IN TABLE * 00020000
* DISPLAY DEPT LIST ON SCREEN * 00020100
* ELSE (OBJFLD IS EM): 6000-EMPLOYEE * 00020200
* IF "LIST GENERIC": 6100-GENEMP * 00020300
* CHOOSE CURSOR BASED ON 6110-GETEMPTAB * 00020400
* SEARCH CRITERIA AND DATA * 00020500
* CREATE ISPF TABLE * 00020600
* DO UNTIL NO MORE RECORDS: * 00020700
* FETCH RECORD * 00020800
* STORE RECORD IN TABLE * 00020900
* DISPLAY DEPARTMENT LIST 6121-GETEMP * 00021000
* ON SCREEN * 00021100
* STORE SELECTED DEPT ID IN * 00021200
* HOST VARIABLE * 00021300
* ELSE: * 00021400
* FETCH SELECTED EMPLOYEE 6200-DISPLAYEMP * 00021500
* DISPLAY EMPLOYEE ON SCREEN * 00021600
* IF ACTION IS ERASE: 6220-ERASEEMP * 00021700
* DELETE EMPLOYEE FROM VEMP * 00021800
* ELSE IF ACTION IS UPDATE: 6230-UPDATEEMP * 00021900
* UPDATE EMPLOYEE IN VEMP * 00022000
* END-DO UNTIL NO MORE TERMINAL INPUT * 00022100
* RELEASE ALL CONNECTIONS * 00022200
----- 00022300
* ENVIRONMENT DIVISION. 00022400
----- 00022500
* INPUT-OUTPUT SECTION. 00022600
* FILE-CONTROL. 00022700
* SELECT MSGOUT ASSIGN TO UT-S-SYSPRINT. 00022800
* 00022900
* DATA DIVISION. 00023000
----- 00023100
* FILE SECTION. 00023200
* 00023300
* FD MSGOUT 00023400
* RECORD CONTAINS 71 CHARACTERS 00023500
* LABEL RECORDS ARE OMITTED. 00023600
* 01 MSGREC PIC X(71). 00023700
* 00023800
* WORKING-STORAGE SECTION. 00023900
----- 00024000
* 77 COIBM PIC X(54) VALUE IS 00024100
* 'COPYRIGHT = 5665-DB2 (C) COPYRIGHT IBM CORP 1982, 1990'. 00024200
* 77 MODULE PIC X(07) VALUE 'DSN8HC3'. 00024300
* 77 MSGS-VAR PIC X(08) VALUE 'DSN8MSG'. 00024400
* 77 MSGCODE PIC X(06). 00024500
* 77 SEL-EXIT PIC X(01). 00024600
* 77 GEND-EXIT PIC X(01). 00024700
* 77 GENE-EXIT PIC X(01). 00024800
* 77 SPECIAL-EXIT PIC X(01). 00024900
* 77 ROWS-CHANGED PIC 9(04). 00025000
* 77 NUMROWS PIC 9(08). 00025100
* 77 PERCENT-COUNTER PIC S9(04) COMP. 00025200
* 77 LENGTH-COUNTER PIC S9(04) COMP. 00025300
* 77 W-BLANK PIC X(01) VALUE ' '. 00025400
----- 00025500
* * ISPF DIALOG VARIABLE NAMES * 00025600
----- 00025700
* EXEC SQL INCLUDE SQLCA END-EXEC. 00025800
* 01 LIST-PANEL-VARS. 00025900
* 03 CH-VAR PIC X(08) VALUE 'ZTDSELS '. 00026000
* 03 QROWS PIC X(08) VALUE 'QROWS '. 00026100
* 00026200
*

```

*	ACTION PANEL VARIABLES		00026300
*			00026400
03	ACT-VAR	PIC X(08) VALUE 'A' .	00026500
03	OBJ-VAR	PIC X(08) VALUE 'OB' .	00026600
03	SEA-VAR	PIC X(08) VALUE 'SE' .	00026700
03	LOC-VAR	PIC X(08) VALUE 'LOCATION' .	00026800
03	DAT-VAR	PIC X(08) VALUE 'NAMEID' .	00026900
*			00027000
*	DEPARTMENT STRUCTURE VARIABLES		00027100
*			00027200
03	DN1M-VAR	PIC X(08) VALUE 'MDEPIDP' .	00027300
03	DNAME1M-VAR	PIC X(08) VALUE 'MDEPNAMP' .	00027400
03	DMGR1M-VAR	PIC X(08) VALUE 'MMGRIDP' .	00027500
03	EFN1M-VAR	PIC X(08) VALUE 'MMGNAMP' .	00027600
03	EMI1M-VAR	PIC X(08) VALUE 'MMGMP' .	00027700
03	ELN1M-VAR	PIC X(08) VALUE 'MMGLNMP' .	00027800
03	DN1-VAR	PIC X(08) VALUE 'DEPIDP' .	00027900
03	DNAME1-VAR	PIC X(08) VALUE 'DEPNAMP' .	00028000
03	DMGR1-VAR	PIC X(08) VALUE 'MGRIDP' .	00028100
03	EFN1-VAR	PIC X(08) VALUE 'MGNAMP' .	00028200
03	EMI1-VAR	PIC X(08) VALUE 'MGMP' .	00028300
03	ELN1-VAR	PIC X(08) VALUE 'MGLNMP' .	00028400
*			00028500
*	DISPLAY PANEL VARIABLES		00028600
*			00028700
03	ACTL-VAR	PIC X(08) VALUE 'PACTION' .	00028800
03	DN2-VAR	PIC X(08) VALUE 'DEPID2' .	00028900
03	DNAME2-VAR	PIC X(08) VALUE 'DEPNAM2' .	00029000
03	DMGR2-VAR	PIC X(08) VALUE 'MGRID2' .	00029100
03	DADM-VAR	PIC X(08) VALUE 'MDEPID2' .	00029200
03	DLOC-VAR	PIC X(08) VALUE 'DEPLOC2' .	00029300
03	EN2-VAR	PIC X(08) VALUE 'EMPID2' .	00029400
03	EFN2-VAR	PIC X(08) VALUE 'EMPNAM2' .	00029500
03	EMI2-VAR	PIC X(08) VALUE 'EMPMI2' .	00029600
03	ELN2-VAR	PIC X(08) VALUE 'MLNM2' .	00029700
03	EWD-VAR	PIC X(08) VALUE 'DEPIDB2' .	00029800
*			00029900
*	SELECT DEPARTMENT VARIABLES		00030000
*			00030100
03	SD-VAR	PIC X(08) VALUE 'SELECT' .	00030200
03	DN3-VAR	PIC X(08) VALUE 'DID' .	00030300
03	DNAME3-VAR	PIC X(08) VALUE 'DEPNGEN' .	00030400
03	DMGR3-VAR	PIC X(08) VALUE 'MID' .	00030500
03	MGRN-VAR	PIC X(08) VALUE 'MNGEN' .	00030600
*			00030700
*	SELECT EMPLOYEE VARIABLES		00030800
*			00030900
03	SEM-VAR	PIC X(08) VALUE 'SELEC4' .	00031000
03	EN4-VAR	PIC X(08) VALUE 'EMPID4' .	00031100
03	EMPN-VAR	PIC X(08) VALUE 'EMPNM4' .	00031200
03	DN4-VAR	PIC X(08) VALUE 'DEPID4' .	00031300
03	DNAME4-VAR	PIC X(08) VALUE 'DPNAME4' .	00031400
*			00031500
*	TABLE VARIABLES		00031600
*			00031700
03	DEPT-TABLE	PIC X(08) VALUE 'DSN8DTAB' .	00031800
03	DS-TABLE	PIC X(08) VALUE 'DSN8STAB' .	00031900
03	EMP-TABLE	PIC X(08) VALUE 'DSN8ESEL' .	00032000
*			00032100
*	VARIABLE LISTS		00032200
*			00032300
03	ACTION-VARS	PIC X(27) VALUE IS	00032400
'( A OB SE LOCATION NAMEID )'.			00032500
03	IDEN-VAR	PIC X(19) VALUE IS	00032600
'( PACTION )'.			00032700
03	ADD-DEPT-VARS	PIC X(40) VALUE IS	00032800
'( DEPID2 DEPNAM2 MGRID2 MDEPID2 DEPLOC2)'.			00032900
03	DEPT-VARS	PIC X(77) VALUE IS	00033000
'( DEPID2 DEPNAM2 MGRID2 MDEPID2 DEPLOC2 EMPID2 EMPNAM2 EMPMI00033100			
-	'2 MLNM2 DEPIDB2 )'.		00033200
03	ADD-EMP-VARS	PIC X(39) VALUE IS	00033300
'( EMPID2 EMPNAM2 EMPMI2 MLNM2 DEPIDB2 )'.			00033400
03	SEL-EMP-VARS	PIC X(47) VALUE IS	00033500
'( ZTDSELS SELEC4 EMPID4 EMPNM4 DEPID4 DPNAME4 )'.			00033600
03	SEL-DEPT-VARS	PIC X(40) VALUE IS	00033700
'( ZTDSELS SELECT DID DEPNGEN MID MNGEN )'.			00033800
03	HEAD-DEPT-VARS	PIC X(51) VALUE IS	00033900
'( MDEPIDP MDEPNAMP MMGRIDP MMGNAMP MMGMP MMGLNMP )'.			00034000
03	DS-VARS	PIC X(45) VALUE IS	00034100
'( DEPIDP DEPNAMP MGRIDP MGNAMP MGMIP MGLNMP )'.			00034200
01	PANEL-VARIABLE-LENGTHS.		00034300
03	CH-VAR-STG	PIC 9(06) COMP VALUE 04.	00034400

03	QROWS-STG	PIC 9(06)	COMP VALUE 08.	00034500
*				00034600
*	ACTION PANEL VARIABLES			00034700
*				00034800
03	AC-VAR-STG	PIC 9(06)	COMP VALUE 01.	00034900
03	OB-VAR-STG	PIC 9(06)	COMP VALUE 02.	00035000
03	SE-VAR-STG	PIC 9(06)	COMP VALUE 02.	00035100
03	LO-VAR-STG	PIC 9(06)	COMP VALUE 16.	00035200
03	DT-VAR-STG	PIC 9(06)	COMP VALUE 36.	00035300
*				00035400
*	DEPARTMENT STRUCTURE VARIABLES			00035500
*				00035600
03	DN1M-VAR-STG	PIC 9(06)	COMP VALUE 03.	00035700
03	DNAME1M-VAR-STG	PIC 9(06)	COMP VALUE 36.	00035800
03	DMGR1M-VAR-STG	PIC 9(06)	COMP VALUE 06.	00035900
03	EFN1M-VAR-STG	PIC 9(06)	COMP VALUE 12.	00036000
03	EMI1M-VAR-STG	PIC 9(06)	COMP VALUE 01.	00036100
03	ELN1M-VAR-STG	PIC 9(06)	COMP VALUE 15.	00036200
03	DN1-VAR-STG	PIC 9(06)	COMP VALUE 03.	00036300
03	DNAME1-VAR-STG	PIC 9(06)	COMP VALUE 36.	00036400
03	DMGR1-VAR-STG	PIC 9(06)	COMP VALUE 06.	00036500
03	EFN1-VAR-STG	PIC 9(06)	COMP VALUE 12.	00036600
03	EMI1-VAR-STG	PIC 9(06)	COMP VALUE 01.	00036700
03	ELN1-VAR-STG	PIC 9(06)	COMP VALUE 15.	00036800
*				00036900
*	DISPLAY PANEL VARIABLES			00037000
*				00037100
03	ACL-VAR-STG	PIC 9(06)	COMP VALUE 07.	00037200
03	OCL-VAR-STG	PIC 9(06)	COMP VALUE 10.	00037300
03	DN2-VAR-STG	PIC 9(06)	COMP VALUE 03.	00037400
03	DNAME2-VAR-STG	PIC 9(06)	COMP VALUE 36.	00037500
03	DMGR2-VAR-STG	PIC 9(06)	COMP VALUE 06.	00037600
03	DADM-VAR-STG	PIC 9(06)	COMP VALUE 03.	00037700
03	DLOC-VAR-STG	PIC 9(06)	COMP VALUE 16.	00037800
03	EN2-VAR-STG	PIC 9(06)	COMP VALUE 06.	00037900
03	EFN2-VAR-STG	PIC 9(06)	COMP VALUE 12.	00038000
03	EMI2-VAR-STG	PIC 9(06)	COMP VALUE 01.	00038100
03	ELN2-VAR-STG	PIC 9(06)	COMP VALUE 15.	00038200
03	EWD-VAR-STG	PIC 9(06)	COMP VALUE 03.	00038300
*				00038400
*	SELECT DEPARTMENT VARIABLES			00038500
*				00038600
03	SD-VAR-STG	PIC 9(06)	COMP VALUE 01.	00038700
03	DN3-VAR-STG	PIC 9(06)	COMP VALUE 03.	00038800
03	DNAME3-VAR-STG	PIC 9(06)	COMP VALUE 36.	00038900
03	DMGR3-VAR-STG	PIC 9(06)	COMP VALUE 06.	00039000
03	MGRN-VAR-STG	PIC 9(06)	COMP VALUE 18.	00039100
*				00039200
*	SELECT EMPLOYEE VARIABLES			00039300
*				00039400
03	SEM-VAR-STG	PIC 9(06)	COMP VALUE 01.	00039500
03	EN4-VAR-STG	PIC 9(06)	COMP VALUE 06.	00039600
03	EMPN-VAR-STG	PIC 9(06)	COMP VALUE 17.	00039700
03	DN4-VAR-STG	PIC 9(06)	COMP VALUE 03.	00039800
03	DNAME4-VAR-STG	PIC 9(06)	COMP VALUE 36.	00039900
*				00040000
*	03 MSGS-VAR-STG	PIC 9(06)	COMP VALUE 79.	00040100
*-----*				00040200
* ISPF DIALOG SERVICES DECLARATIONS				* 00040300
*-----*				* 00040400
01	I-VDEFINE	PIC X(08)	VALUE 'VDEFINE '.	00040500
01	I-VGET	PIC X(08)	VALUE 'VGET '.	00040600
01	I-VPUT	PIC X(08)	VALUE 'VPUT '.	00040700
01	I-DISPLAY	PIC X(08)	VALUE 'DISPLAY '.	00040800
01	I-TBDISPL	PIC X(08)	VALUE 'TBDISPL '.	00040900
01	I-TBTOP	PIC X(08)	VALUE 'TBTOP '.	00041000
01	I-TBCREATE	PIC X(08)	VALUE 'TBCREATE '.	00041100
01	I-TBCLOSE	PIC X(08)	VALUE 'TBCLOSE '.	00041200
01	I-TBADD	PIC X(08)	VALUE 'TBADD '.	00041300
01	I-TBGET	PIC X(08)	VALUE 'TBGET '.	00041400
01	I-TBQUERY	PIC X(08)	VALUE 'TBQUERY '.	00041500
*-----*				* 00041600
* ISPF CALL MODIFIERS				* 00041700
*-----*				* 00041800
01	I-NOWRITE	PIC X(08)	VALUE 'NOWRITE '.	00041900
01	I-REPLACE	PIC X(08)	VALUE 'REPLACE '.	00042000
01	I-CHAR	PIC X(08)	VALUE 'CHAR '.	00042100
*-----*				* 00042200
* ISPF PANEL NAMES				* 00042300
*-----*				* 00042400
01	SEL-PANEL	PIC X(08)	VALUE 'DSN8SSH '.	00042500
01	STR-PANEL	PIC X(08)	VALUE 'DSN8SSSH1'.	00042600

```

01 DEPT-PANEL PIC X(08) VALUE 'DSN8SSH2'. 00042700
01 GEND-PANEL PIC X(08) VALUE 'DSN8SSH3'. 00042800
01 GENE-PANEL PIC X(08) VALUE 'DSN8SSH4'. 00042900
01 EMP-PANEL PIC X(08) VALUE 'DSN8SSH5'. 00043000
----------*-----*-----*-----*-----*-----*-----*
* LOCAL-VARIABLES *-----*-----*-----*-----*-----*-----*-----*-----*
----------*-----*-----*-----*-----*-----*-----*
01 LOCAL-VARIABLES. 00043400
03 DATAW PIC X(36). 00043500
03 GENDATA PIC X(36). 00043600
03 SEL-DEPT PIC X(01). 00043700
03 SEL-EMP PIC X(01). 00043800
03 MGR-NAME PIC X(18). 00043900
03 EMP-NAME PIC X(17). 00044000
03 TOKEN PIC X(70). 00044100
03 TEMPLOC PIC X(16). 00044200
03 PREVLOC PIC X(16). 00044300
03 TEMPDEPT PIC X(03). 00044400
03 CURDEPT PIC X(03). 00044500
03 DELDEPT PIC X(03). 00044600
03 STACKTOP PIC S9(04). 00044700
03 DEPTPTR PIC S9(04). 00044800
03 LISTPTR PIC S9(04). 00044900
03 LOCPTR PIC S9(04). 00045000
03 LOCTOP PIC S9(04). 00045100
03 CONVSQL PIC S9(15) COMP-3. 00045200
03 OUTMSG PIC X(69). 00045300
03 TMSG REDEFINES OUTMSG. 00045400
05 TMSGTXT PIC X(46). 00045500
05 FILLER PIC X(23). 00045600
03 MSGS PIC X(79) VALUE SPACES. 00045700
03 MSGS-DETAIL REDEFINES MSGS. 00045800
05 OUT-MESSAGE PIC X(46). 00045900
05 SQL-CODE PIC +(04). 00046000
05 FILLER PIC X(29). 00046100
 00046200
01 CONV PIC S9(15) COMP-3. 00046300
01 DEPTS-TABLE. 00046400
03 DEPTS OCCURS 1000 TIMES. 00046500
05 DEPTS-ITEM PIC X(03). 00046600
01 DEPTLIST-TABLE. 00046700
03 DEPTLIST OCCURS 1000 TIMES. 00046800
05 DEPTLIST-ITEM PIC X(03). 00046900
01 LOCLIST-TABLE. 00047000
03 LOCLIST OCCURS 1000 TIMES. 00047100
05 LOCLIST-ITEM PIC X(16). 00047200
----------*-----*-----*-----*-----*-----*-----*
* ACTION PANEL - IO AREA *-----*-----*-----*-----*-----*-----*-----*-----*
----------*-----*-----*-----*-----*-----*-----*
01 PGM-PANEL-VARS. 00047600
03 ACTION PIC X(01). 00047700
03 OBJFLD PIC X(02). 00047800
03 SEARCH-CRIT PIC X(02). 00047900
03 LOCATION PIC X(16). 00048000
03 NAMEID PIC X(36). 00048100
03 ACTION-LIST PIC X(07). 00048200
----------*-----*-----*-----*-----*-----*-----*
* EMPLOYEE RECORD - IO AREA *-----*-----*-----*-----*-----*-----*-----*-----*
----------*-----*-----*-----*-----*-----*-----*
01 EMP-RECORD. 00048600
02 EMP-NUMB PIC X(06). 00048700
02 EMP-FIRST-NAME PIC X(12). 00048800
02 EMP-MID-INIT PIC X(01). 00048900
02 EMP-LAST-NAME PIC X(15). 00049000
02 EMP-WORK-DEPT PIC X(03). 00049100
01 EMP-INDICATOR-TABLE. 00049200
02 WORK-DEPT-IND PIC S9(4) COMP. 00049300
----------*-----*-----*-----*-----*-----*-----*
* EMPLOYEE RECORD FOR DEPT STRUCTURE - IO AREA *-----*-----*-----*-----*-----*-----*-----*-----*
----------*-----*-----*-----*-----*-----*-----*
01 EMP1-RECORD. 00049600
02 EMP1-NUMB PIC X(06). 00049700
02 EMP1-FIRST-NAME PIC X(12). 00049800
02 EMP1-MID-INIT PIC X(01). 00049900
02 EMP1-LAST-NAME PIC X(15). 00050000
02 EMP1-WORK-DEPT PIC X(03). 00050100
01 EMP1-INDICATOR-TABLE. 00050200
02 WORK1-DEPT-IND PIC S9(4) COMP. 00050300
 00050400
----------*-----*-----*-----*-----*-----*-----*
* DEPARTMENT RECORD - IO AREA *-----*-----*-----*-----*-----*-----*-----*-----*
----------*-----*-----*-----*-----*-----*-----*
01 DEPT-RECORD. 00050500
 00050600
 00050700
 00050800

```

```

02 DEPT-NUMB PIC X(03). 00050900
02 DEPT-NAME PIC X(36). 00051000
02 DEPT-MGR PIC X(06). 00051100
02 DEPT-ADMR PIC X(03). 00051200
02 DEPT-LOC PIC X(16). 00051300
01 DEPT-INDICATOR-TABLE.
02 DEPT-MGR-IND PIC S9(4) COMP. 00051400
----- 00051600
* DEPARTMENT RECORD FOR DEPT STRUCTURE - IO AREA * 00051700
----- 00051800
01 DEPT1-RECORD.
02 DEPT1-NUMB PIC X(03). 00051900
02 DEPT1-NAME PIC X(36). 00052000
02 DEPT1-MGR PIC X(06). 00052100
02 DEPT1-ADMR PIC X(03). 00052200
02 DEPT1-LOC PIC X(16). 00052300
01 DEPT1-INDICATOR-TABLE.
02 DEPT1-MGR-IND PIC S9(4) COMP. 00052400
----- 00052500
* SQLCA OUTPUT * 00052600
----- 00052700
01 SQLCA-LINE0.
02 FILLER PIC X(45) VALUE 00052800
'DSN8060E DSN8HC3 SQL ERROR, RETURN CODE IS: '.
02 SQLCODE-MSG PIC +(16). 00052900
02 FILLER PIC X(11) VALUE SPACES. 00053000
00053100
01 SQLCA-LINE1.
02 FILLER PIC X(05) VALUE SPACES. 00053200
02 SQLCAID-NAME PIC X(13) VALUE 'SQLCAID = '.
02 SQLCAID-VALUE PIC X(08). 00053300
00053400
02 FILLER PIC X(14) VALUE SPACES. 00053500
02 SQLCABC-NAME PIC X(13) VALUE 'SQLABC = '.
02 SQLCABC-VALUE PIC Z(15). 00053600
00053700
02 FILLER PIC X(03) VALUE SPACES. 00053800
00053900
02 SQLCODE-NAME PIC X(13) VALUE 'SQLCODE = '.
02 SQLCODE-VALUE PIC +(16). 00054000
00054100
02 FILLER PIC X(07) VALUE SPACES. 00054200
02 SQLERRML-NAME PIC X(13) VALUE 'SQLERRML = '.
02 SQLERRML-VALUE PIC Z(15). 00054300
00054400
02 FILLER PIC X(03) VALUE SPACES. 00054500
00054600
01 SQLCA-LINE2.
02 FILLER PIC X(05) VALUE SPACES. 00054700
02 SQLCODE-NAME PIC X(13) VALUE 'SQLCODE = '.
02 SQLCODE-VALUE PIC +(16). 00054800
00054900
02 FILLER PIC X(07) VALUE SPACES. 00055000
02 SQLERRML-NAME PIC X(13) VALUE 'SQLERRML = '.
02 SQLERRML-VALUE PIC Z(15). 00055100
00055200
02 FILLER PIC X(03) VALUE SPACES. 00055300
00055400
00055500
01 SQLCA-LINE3.
02 FILLER PIC X(05) VALUE SPACES. 00055600
02 SQLERRMC-NAME PIC X(13) VALUE 'SQLERRMC = '.
02 FILLER PIC X(53) VALUE SPACES. 00055700
00055800
00055900
00056000
01 SQLCA-LINE4.
02 FILLER PIC X(01) VALUE SPACES. 00056100
02 SQLERRMC-VALUE PIC X(70). 00056200
00056300
00056400
01 SQLCA-LINE5.
02 FILLER PIC X(05) VALUE SPACES. 00056500
02 SQLERRP-NAME PIC X(13) VALUE 'SQLERRP = '.
02 SQLERRP-VALUE PIC X(08). 00056600
00056700
02 FILLER PIC X(14) VALUE SPACES. 00056800
02 SQLERRD1-NAME PIC X(13) VALUE 'SQLERRD(1) = '.
02 SQLERRD1-VALUE PIC Z(14)9. 00056900
00057000
02 FILLER PIC X(03) VALUE SPACES. 00057100
00057200
00057300
01 SQLCA-LINE6.
02 FILLER PIC X(05) VALUE SPACES. 00057400
02 SQLERRD2-NAME PIC X(13) VALUE 'SQLERRD(2) = '.
02 SQLERRD2-VALUE PIC Z(14)9. 00057500
00057600
02 FILLER PIC X(07) VALUE SPACES. 00057700
02 SQLERRD3-NAME PIC X(13) VALUE 'SQLERRD(3) = '.
02 SQLERRD3-VALUE PIC Z(14)9. 00057800
00057900
02 FILLER PIC X(03) VALUE SPACES. 00058000
00058100
00058200
01 SQLCA-LINE7.
02 FILLER PIC X(05) VALUE SPACES. 00058300
02 SQLERRD4-NAME PIC X(13) VALUE 'SQLERRD(4) = '.
02 SQLERRD4-VALUE PIC Z(14)9. 00058400
00058500
02 FILLER PIC X(07) VALUE SPACES. 00058600
02 SQLERRD5-NAME PIC X(13) VALUE 'SQLERRD(5) = '.
02 SQLERRD5-VALUE PIC Z(14)9. 00058700
00058800
02 FILLER PIC X(03) VALUE SPACES. 00058900
00059000

```

```

01 SQLCA-LINE8. 00059100
 02 FILLER PIC X(05) VALUE SPACES. 00059200
 02 SQLERRD6-NAME PIC X(13) VALUE 'SQLERRD(6)' = '. 00059300
 02 SQLERRD6-VALUE PIC Z(14)9. 00059400
 02 FILLER PIC X(07) VALUE SPACES. 00059500
 02 SQLWARN0-NAME PIC X(13) VALUE 'SQLWARN0' = '. 00059600
 02 SQLWARN0-VALUE PIC X. 00059700
 02 FILLER PIC X(17) VALUE SPACES. 00059800
 00059900
01 SQLCA-LINE9. 00060000
 02 FILLER PIC X(05) VALUE SPACES. 00060100
 02 SQLWARN1-NAME PIC X(13) VALUE 'SQLWARN1' = '. 00060200
 02 SQLWARN1-VALUE PIC X. 00060300
 02 FILLER PIC X(21) VALUE SPACES. 00060400
 02 SQLWARN2-NAME PIC X(13) VALUE 'SQLWARN2' = '. 00060500
 02 SQLWARN2-VALUE PIC X. 00060600
 02 FILLER PIC X(17) VALUE SPACES. 00060700
 00060800
01 SQLCA-LINE10. 00060900
 02 FILLER PIC X(05) VALUE SPACES. 00061000
 02 SQLWARN3-NAME PIC X(13) VALUE 'SQLWARN3' = '. 00061100
 02 SQLWARN3-VALUE PIC X. 00061200
 02 FILLER PIC X(21) VALUE SPACES. 00061300
 02 SQLWARN4-NAME PIC X(13) VALUE 'SQLWARN4' = '. 00061400
 02 SQLWARN4-VALUE PIC X. 00061500
 02 FILLER PIC X(17) VALUE SPACES. 00061600
 00061700
01 SQLCA-LINE11. 00061800
 02 FILLER PIC X(05) VALUE SPACES. 00061900
 02 SQLWARN5-NAME PIC X(13) VALUE 'SQLWARN5' = '. 00062000
 02 SQLWARN5-VALUE PIC X. 00062100
 02 FILLER PIC X(21) VALUE SPACES. 00062200
 02 SQLWARN6-NAME PIC X(13) VALUE 'SQLWARN6' = '. 00062300
 02 SQLWARN6-VALUE PIC X. 00062400
 02 FILLER PIC X(17) VALUE SPACES. 00062500
 00062600
01 SQLCA-LINE12. 00062700
 02 FILLER PIC X(05) VALUE SPACES. 00062800
 02 SQLWARN7-NAME PIC X(13) VALUE 'SQLWARN7' = '. 00062900
 02 SQLWARN7-VALUE PIC X. 00063000
 02 FILLER PIC X(21) VALUE SPACES. 00063100
 02 SQLWARN8-NAME PIC X(13) VALUE 'SQLWARN8' = '. 00063200
 02 SQLWARN8-VALUE PIC X. 00063300
 02 FILLER PIC X(17) VALUE SPACES. 00063400
 00063500
01 SQLCA-LINE13. 00063600
 02 FILLER PIC X(05) VALUE SPACES. 00063700
 02 SQLWARN9-NAME PIC X(13) VALUE 'SQLWARN9' = '. 00063800
 02 SQLWARN9-VALUE PIC X. 00063900
 02 FILLER PIC X(21) VALUE SPACES. 00064000
 02 SQLWARNA-NAME PIC X(13) VALUE 'SQLWARNA' = '. 00064100
 02 SQLWARNA-VALUE PIC X. 00064200
 02 FILLER PIC X(17) VALUE SPACES. 00064300
 00064400
01 SQLCA-LINE14. 00064500
 02 FILLER PIC X(05) VALUE SPACES. 00064600
 02 SQLSTATE-NAME PIC X(13) VALUE 'SQLSTATE' = '. 00064700
 02 SQLSTATE-VALUE PIC X(05). 00064800
 02 FILLER PIC X(48) VALUE SPACES. 00064900
 00065000
***** * 00065100
* LINKAGE SECTION * 00065200
***** * 00065300
 00065400
LINKAGE SECTION. 00065500
 00065600
----- 00065700
* SQL DECLARATION FOR VIEW VHDEPT * 00065800
----- 00065900
EXEC SQL DECLARE VHDEPT TABLE 00066000
 (DEPTNO CHAR(3) NOT NULL, 00066100
 DEPTNAME VARCHAR(36) NOT NULL, 00066200
 MGRNO CHAR(6) NOT NULL, 00066300
 ADMRDEPT CHAR(3) NOT NULL, 00066400
 LOCATION CHAR(16)) END-EXEC. 00066500
----- 00066600
* SQL DECLARATION FOR VIEW VEMP * 00066700
----- 00066800
EXEC SQL DECLARE VEMP TABLE 00066900
 (EMPNO CHAR(6) NOT NULL, 00067000
 FIRSTNME VARCHAR(12) NOT NULL, 00067100
 MIDINIT CHAR(1) NOT NULL, 00067200

```

```

 LASTNAME VARCHAR(15) NOT NULL,
 WORKDEPT CHAR(3)) END-EXEC. 00067300
 00067400
----- * 00067500
* SQL CURSORS * 00067600
----------*-----*-----*-----*-----*-----*-----*-----*
* EXEC SQL DECLARE CURDEPTLOC CURSOR FOR 00067800
 SELECT LOCATION 00067900
 FROM VHDEPT 00068000
 WHERE DEPTNO = :EMP-WORK-DEPT 00068100
 AND LOCATION = CURRENT SERVER 00068200
 END-EXEC. 00068300
 00068400
* EXEC SQL DECLARE DEPTLOC CURSOR FOR 00068500
 SELECT LOCATION 00068600
 FROM VHDEPT 00068700
 WHERE DEPTNO = :EMP-WORK-DEPT 00068800
 END-EXEC. 00068900
 00069000
* EXEC SQL DECLARE LOCS CURSOR FOR 00069100
 SELECT DISTINCT LOCATION 00069200
 FROM VHDEPT 00069300
 WHERE LOCATION <> :LOCATION 00069400
 AND LOCATION <> :LOCATION 00069500
 AND LOCATION <> CURRENT SERVER 00069600
 END-EXEC. 00069700
 00069800
* EXEC SQL DECLARE SUBDEPTS CURSOR FOR 00069900
 SELECT DEPTNO
 FROM VHDEPT
 WHERE ADMRDEPT = :CURDEPT
 AND DEPTNO <> :CURDEPT
 END-EXEC. 00070000
 00070100
* EXEC SQL DECLARE DEPT1 CURSOR FOR 00070200
 SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION,
 EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
 FROM VHDEPT, VEMP
 WHERE DEPTNO = :DATAW
 AND MGRNO = EMPNO
 UNION
 SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION,
 EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
 FROM VHDEPT
 WHERE DEPTNO = :DATAW
 AND MGRNO IS NULL
 END-EXEC. 00070300
 00070400
 00070500
 00070600
* EXEC SQL DECLARE ALLDEPT1 CURSOR FOR 00070700
 SELECT DEPTNO, DEPTNAME, MGRNO,
 SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME
 FROM VHDEPT, VEMP
 WHERE MGRNO = EMPNO
 AND DEPTNO LIKE :GENDATA
 UNION
 SELECT DEPTNO, DEPTNAME, MGRNO,
 SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME
 FROM VHDEPT
 WHERE MGRNO IS NULL
 AND DEPTNO LIKE :GENDATA
 ORDER BY 1
 END-EXEC. 00070800
 00070900
 00071000
 00071100
 00071200
 00071300
 00071400
 00071500
 00071600
 00071700
 00071800
 00071900
 00072000
 00072100
 00072200
 00072300
 00072400
 00072500
 00072600
 00072700
 00072800
 00072900
 00073000
 00073100
 00073200
 00073300
 00073400
 00073500
 00073600
 00073700
 00073800
 00073900
 00074000
 00074100
 00074200
 00074300
 00074400
 00074500
 00074600
 00074700
 00074800
 00074900
 00075000
 00075100
 00075200
 00075300
 00075400

```

```

 UNION 00075500
 SELECT DEPTNO, DEPTNAME, MGRNO, '' 00075600
 FROM VHDEPT 00075700
 WHERE MGRNO IS NULL 00075800
 AND MGRNO LIKE :GENDATA 00075900
 ORDER BY 1 00076000
 END-EXEC. 00076100
*
 EXEC SQL DECLARE ALLDEPT4 CURSOR FOR 00076200
 SELECT DEPTNO, DEPTNAME, MGRNO,
 SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME 00076300
 FROM VHDEPT, VEMP
 WHERE MGRNO = EMPNO
 AND LASTNAME LIKE :GENDATA 00076400
 ORDER BY 1 00076500
 END-EXEC. 00076600
*
 EXEC SQL DECLARE ALLDEPT5 CURSOR FOR 00076700
 SELECT DEPTNO, DEPTNAME, MGRNO,
 SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME 00076800
 FROM VHDEPT, VEMP
 WHERE MGRNO = EMPNO
 AND DEPTNAME = :GENDATA 00076900
 UNION
 SELECT DEPTNO, DEPTNAME, MGRNO, '' 00077000
 FROM VHDEPT
 WHERE MGRNO IS NULL 00077100
 AND DEPTNAME = :GENDATA 00077200
 ORDER BY 1 00077300
 END-EXEC. 00077400
*
 EXEC SQL DECLARE ALLDEPT6 CURSOR FOR 00077500
 SELECT DEPTNO, DEPTNAME, MGRNO,
 SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME 00077600
 FROM VHDEPT, VEMP
 WHERE MGRNO = EMPNO
 AND MGRNO = :GENDATA 00077700
 UNION
 SELECT DEPTNO, DEPTNAME, MGRNO, '' 00077800
 FROM VHDEPT
 WHERE MGRNO IS NULL 00077900
 AND MGRNO = :GENDATA 00078000
 ORDER BY 1 00078100
 END-EXEC. 00078200
*
 EXEC SQL DECLARE ALLDEPT7 CURSOR FOR 00078300
 SELECT DEPTNO, DEPTNAME, MGRNO,
 SUBSTR(FIRSTNME, 1, 1) || MIDINIT || ' ' || LASTNAME 00078400
 FROM VHDEPT, VEMP
 WHERE MGRNO = EMPNO
 AND LASTNAME = :GENDATA 00078500
 ORDER BY 1 00078600
 END-EXEC. 00078700
*
 EXEC SQL DECLARE EMP1 CURSOR FOR 00078800
 SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION,
 EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT 00078900
 FROM VHDEPT, VEMP
 WHERE EMPNO = :DATAW
 AND WORKDEPT = DEPTNO 00079000
 UNION
 SELECT ' ', ' ', ' ', ' ', ' '
 EMPNO, FIRSTNME, MIDINIT, LASTNAME, '' 00079100
 FROM VEMP
 WHERE EMPNO = :DATAW
 AND WORKDEPT IS NULL 00079200
 END-EXEC. 00079300
*
 EXEC SQL DECLARE ALLEMP1 CURSOR FOR 00079400
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00079500
 WORKDEPT, DEPTNAME 00079600
 FROM VHDEPT, VEMP
 WHERE DEPTNO = WORKDEPT
 AND EMPNO LIKE :GENDATA 00079700
 UNION
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00079800
 WORKDEPT, DEPTNAME 00079900
 FROM VEMP
 WHERE WORKDEPT IS NULL
 AND EMPNO LIKE :GENDATA 00080000
 ORDER BY 1 00080100
 END-EXEC. 00080200
*
 EXEC SQL DECLARE ALLEMP1 CURSOR FOR 00080300
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00080400
 WORKDEPT, DEPTNAME 00080500
 FROM VHDEPT, VEMP
 WHERE DEPTNO = WORKDEPT
 AND EMPNO LIKE :GENDATA 00080600
 UNION
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00080700
 WORKDEPT, DEPTNAME 00080800
 FROM VEMP
 WHERE WORKDEPT IS NULL
 AND EMPNO LIKE :GENDATA 00080900
 ORDER BY 1 00081000
 END-EXEC. 00081100
*
 EXEC SQL DECLARE ALLEMP1 CURSOR FOR 00081200
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00081300
 WORKDEPT, DEPTNAME 00081400
 FROM VHDEPT, VEMP
 WHERE DEPTNO = WORKDEPT
 AND EMPNO LIKE :GENDATA 00081500
 UNION
 SELECT ' ', ' ', ' ', ' ', ' '
 EMPNO, FIRSTNME, MIDINIT, LASTNAME, '' 00081600
 FROM VEMP
 WHERE EMPNO = :DATAW
 AND WORKDEPT IS NULL 00081700
*
 EXEC SQL DECLARE ALLEMP1 CURSOR FOR 00081800
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00081900
 WORKDEPT, DEPTNAME 00082000
 FROM VHDEPT, VEMP
 WHERE DEPTNO = WORKDEPT
 AND EMPNO LIKE :GENDATA 00082100
 UNION
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00082200
 WORKDEPT, DEPTNAME 00082300
 FROM VEMP
 WHERE WORKDEPT IS NULL
 AND EMPNO LIKE :GENDATA 00082400
 ORDER BY 1 00082500
 END-EXEC. 00082600
*
 EXEC SQL DECLARE ALLEMP1 CURSOR FOR 00082700
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00082800
 WORKDEPT, DEPTNAME 00082900
 FROM VHDEPT, VEMP
 WHERE DEPTNO = WORKDEPT
 AND EMPNO LIKE :GENDATA 00083000
 UNION
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00083100
 WORKDEPT, DEPTNAME 00083200
 FROM VEMP
 WHERE WORKDEPT IS NULL
 AND EMPNO LIKE :GENDATA 00083300
 ORDER BY 1 00083400
 END-EXEC. 00083500
*
 EXEC SQL DECLARE ALLEMP1 CURSOR FOR 00083600
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00083700
 WORKDEPT, DEPTNAME 00083800
 FROM VHDEPT, VEMP
 WHERE DEPTNO = WORKDEPT
 AND EMPNO LIKE :GENDATA 00083900
 UNION
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00084000
 WORKDEPT, DEPTNAME 00084100
 FROM VEMP
 WHERE WORKDEPT IS NULL
 AND EMPNO LIKE :GENDATA 00084200
 ORDER BY 1 00084300
 END-EXEC. 00084400

```

```

* 00083700
* EXEC SQL DECLARE ALLEMP2 CURSOR FOR 00083800
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00083900
 WORKDEPT, DEPTNAME 00084000
 FROM VHDEPT, VEMP 00084100
 WHERE DEPTNO = WORKDEPT 00084200
 AND LASTNAME LIKE :GENDATA 00084300
 UNION 00084400
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00084500
 WORKDEPT, ' ' 00084600
 FROM VEMP 00084700
 WHERE WORKDEPT IS NULL 00084800
 AND LASTNAME LIKE :GENDATA 00084900
 ORDER BY 1 00085000
END-EXEC. 00085100
00085200
* 00085300
* EXEC SQL DECLARE ALLEMP3 CURSOR FOR 00085400
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00085500
 WORKDEPT, DEPTNAME 00085600
 FROM VHDEPT, VEMP 00085700
 WHERE DEPTNO = WORKDEPT 00085800
 AND LASTNAME = :GENDATA 00085900
 UNION 00086000
 SELECT EMPNO, SUBSTR(FIRSTNME, 1, 1) || ' ' || LASTNAME, 00086100
 WORKDEPT, ' ' 00086200
 FROM VEMP 00086300
 WHERE WORKDEPT IS NULL 00086400
 AND LASTNAME = :GENDATA 00086500
 ORDER BY 1 00086600
END-EXEC. 00086700
00086800
* 00086900
* EXEC SQL DECLARE DEPTSTR CURSOR FOR 00087000
 SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION, 00087100
 FIRSTNME, MIDINIT, LASTNAME 00087200
 FROM VHDEPT, VEMP 00087300
 WHERE ADMRDEPT = :DATAW 00087400
 AND MGRNO = EMPNO 00087500
 UNION 00087600
 SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION, 00087700
 ' ', ' ', ' ', ' ', ' ' 00087800
 FROM VHDEPT 00087900
 WHERE ADMRDEPT = :DATAW 00088000
 AND MGRNO IS NULL 00088100
 ORDER BY 1 00088200
END-EXEC. 00088300
00088400
* 00088500
* EJECT * 00088600
PROCEDURE DIVISION. * 00088700

* SQL RETURN CODE HANDLING * 00088800

 EXEC SQL WHENEVER SQLERROR GOTO L8000-P3-DBERROR END-EXEC. 00088800
 EXEC SQL WHENEVER SQLWARNING GOTO L8000-P3-DBERROR END-EXEC. 00088900
 EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC. 00089000
* 00089100

* DEFINE COBOL - SPF VARIABLES * 00089200

----- 00089300
----- 00089400
 0000-PROGRAM-START. 00089500
 CALL 'ISPLINK' USING I-VDEFINE, CH-VAR, ROWS-CHANGED, 00089600
 I-CHAR, CH-VAR-STG. 00089700
 CALL 'ISPLINK' USING I-VDEFINE, QROWS, NUMROWS, 00089800
 I-CHAR, QROWS-STG. 00089900
* 00090000

* ACTION PANEL 00090100

----- 00090200
 CALL 'ISPLINK' USING I-VDEFINE, ACT-VAR, ACTION, 00090300
 I-CHAR, AC-VAR-STG. 00090400
 CALL 'ISPLINK' USING I-VDEFINE, OBJ-VAR, OBJJFLD, 00090500
 I-CHAR, OB-VAR-STG. 00090600
 CALL 'ISPLINK' USING I-VDEFINE, SEA-VAR, SEARCH-CRIT, 00090700
 I-CHAR, SE-VAR-STG. 00090800
 CALL 'ISPLINK' USING I-VDEFINE, LOC-VAR, LOCATION, 00090900
 I-CHAR, LO-VAR-STG. 00091000
 CALL 'ISPLINK' USING I-VDEFINE, DAT-VAR, NAMEID, 00091100
 I-CHAR, DT-VAR-STG. 00091200
* 00091300

* DEPARTMENT STRUCTURE PANEL 00091400

----- 00091500
 CALL 'ISPLINK' USING I-VDEFINE, DN1M-VAR, DEPT1-NUMB, 00091600
 I-CHAR, DN1M-VAR-STG. 00091700
 CALL 'ISPLINK' USING I-VDEFINE, DNAME1M-VAR, DEPT1-NAME, 00091800

```

I-CHAR, DNAME1M-VAR-STG.	00091900
CALL 'ISPLINK' USING I-VDEFINE, DMGR1M-VAR, DEPT1-MGR, I-CHAR, DMGR1M-VAR-STG.	00092000
CALL 'ISPLINK' USING I-VDEFINE, EFN1M-VAR, EMP1-FIRST-NAME, I-CHAR, EFN1M-VAR-STG.	00092100
CALL 'ISPLINK' USING I-VDEFINE, EMI1M-VAR, EMP1-MID-INIT, I-CHAR, EMI1M-VAR-STG.	00092200
CALL 'ISPLINK' USING I-VDEFINE, ELN1M-VAR, EMP1-LAST-NAME, I-CHAR, ELN1M-VAR-STG.	00092300
CALL 'ISPLINK' USING I-VDEFINE, DN1-VAR, DEPT-NUMB, I-CHAR, DN1-VAR-STG.	00092400
CALL 'ISPLINK' USING I-VDEFINE, DNAME1-VAR, DEPT-NAME, I-CHAR, DNAME1-VAR-STG.	00092500
CALL 'ISPLINK' USING I-VDEFINE, DMGR1-VAR, DEPT-MGR, I-CHAR, DMGR1-VAR-STG.	00092600
CALL 'ISPLINK' USING I-VDEFINE, EFN1-VAR, EMP-FIRST-NAME, I-CHAR, EFN1-VAR-STG.	00092700
CALL 'ISPLINK' USING I-VDEFINE, EMI1-VAR, EMP-MID-INIT, I-CHAR, EMI1-VAR-STG.	00092800
CALL 'ISPLINK' USING I-VDEFINE, ELN1-VAR, EMP-LAST-NAME, I-CHAR, ELN1-VAR-STG.	00092900
*	00093000
* DISPLAY PANEL	00093100
*	00093200
CALL 'ISPLINK' USING I-VDEFINE, ACTL-VAR, ACTION-LIST, I-CHAR, ACL-VAR-STG.	00093300
CALL 'ISPLINK' USING I-VDEFINE, DN2-VAR, DEPT-NUMB, I-CHAR, DN2-VAR-STG.	00093400
CALL 'ISPLINK' USING I-VDEFINE, DNAME2-VAR, DEPT-NAME, I-CHAR, DNAME2-VAR-STG.	00093500
CALL 'ISPLINK' USING I-VDEFINE, DMGR2-VAR, DEPT-MGR, I-CHAR, DMGR2-VAR-STG.	00093600
CALL 'ISPLINK' USING I-VDEFINE, DADM-VAR, DEPT-ADMR, I-CHAR, DADM-VAR-STG.	00093700
CALL 'ISPLINK' USING I-VDEFINE, DLOC-VAR, DEPT-LOC, I-CHAR, DLOC-VAR-STG.	00093800
CALL 'ISPLINK' USING I-VDEFINE, EN2-VAR, EMP-NUMB, I-CHAR, EN2-VAR-STG.	00093900
CALL 'ISPLINK' USING I-VDEFINE, EFN2-VAR, EMP-FIRST-NAME, I-CHAR, EFN2-VAR-STG.	00094000
CALL 'ISPLINK' USING I-VDEFINE, EMI2-VAR, EMP-MID-INIT, I-CHAR, EMI2-VAR-STG.	00094100
CALL 'ISPLINK' USING I-VDEFINE, ELN2-VAR, EMP-LAST-NAME, I-CHAR, ELN2-VAR-STG.	00094200
CALL 'ISPLINK' USING I-VDEFINE, EWD-VAR, EMP-WORK-DEPT, I-CHAR, EWD-VAR-STG.	00094300
*	00094400
* SELECT DEPARTMENT PANEL	00094500
*	00094600
CALL 'ISPLINK' USING I-VDEFINE, SD-VAR, SEL-DEPT, I-CHAR, SD-VAR-STG.	00094700
CALL 'ISPLINK' USING I-VDEFINE, DN3-VAR, DEPT-NUMB, I-CHAR, DN3-VAR-STG.	00094800
CALL 'ISPLINK' USING I-VDEFINE, DNAME3-VAR, DEPT-NAME, I-CHAR, DNAME3-VAR-STG.	00094900
CALL 'ISPLINK' USING I-VDEFINE, DMGR3-VAR, DEPT-MGR, I-CHAR, DMGR3-VAR-STG.	00095000
CALL 'ISPLINK' USING I-VDEFINE, MGRN-VAR, MGR-NAME, I-CHAR, MGRN-VAR-STG.	00095100
*	00095200
* SELECT EMPLOYEE PANEL	00095300
*	00095400
CALL 'ISPLINK' USING I-VDEFINE, SEM-VAR, SEL-EMP, I-CHAR, SEM-VAR-STG.	00095500
CALL 'ISPLINK' USING I-VDEFINE, EN4-VAR, EMP-NUMB, I-CHAR, EN4-VAR-STG.	00095600
CALL 'ISPLINK' USING I-VDEFINE, EMPN-VAR, EMP-NAME, I-CHAR, EMPN-VAR-STG.	00095700
CALL 'ISPLINK' USING I-VDEFINE, DN4-VAR, EMP-WORK-DEPT, I-CHAR, DN4-VAR-STG.	00095800
CALL 'ISPLINK' USING I-VDEFINE, DNAME4-VAR, DEPT-NAME, I-CHAR, DNAME4-VAR-STG.	00095900
*	00096000
CALL 'ISPLINK' USING I-VDEFINE, MSGS-VAR, MSGS, I-CHAR, MSGS-VAR-STG.	00096100
*	00096200
MOVE 'N' TO SEL-EXIT.	00096300
MOVE SPACES TO PREVLOC.	00096400
PERFORM 1000-MAIN-LOOP THRU 1000-MAIN-LOOP-EXIT	00096500
*	00096600
*-----*	00096700
* MAIN PROGRAM	00096800
*-----*	00096900
MOVE 'N' TO SEL-EXIT.	00097000
MOVE SPACES TO PREVLOC.	00097100
PERFORM 1000-MAIN-LOOP THRU 1000-MAIN-LOOP-EXIT	00097200
*	00097300
CALL 'ISPLINK' USING I-VDEFINE, ACTL-VAR, ACTION-LIST, I-CHAR, ACTL-VAR-STG.	00097400
CALL 'ISPLINK' USING I-VDEFINE, DN5-VAR, DEPT-NUMB, I-CHAR, DN5-VAR-STG.	00097500
CALL 'ISPLINK' USING I-VDEFINE, DNAME5-VAR, DEPT-NAME, I-CHAR, DNAME5-VAR-STG.	00097600
CALL 'ISPLINK' USING I-VDEFINE, DMGR5-VAR, DEPT-MGR, I-CHAR, DMGR5-VAR-STG.	00097700
*	00097800
*-----*	00097900
*-----*	00098000
CALL 'ISPLINK' USING I-VDEFINE, SD-VAR, SEL-DEPT, I-CHAR, SD-VAR-STG.	00098100
CALL 'ISPLINK' USING I-VDEFINE, DN6-VAR, DEPT-NUMB, I-CHAR, DN6-VAR-STG.	00098200
CALL 'ISPLINK' USING I-VDEFINE, DNAME6-VAR, DEPT-NAME, I-CHAR, DNAME6-VAR-STG.	00098300
CALL 'ISPLINK' USING I-VDEFINE, DMGR6-VAR, DEPT-MGR, I-CHAR, DMGR6-VAR-STG.	00098400
*	00098500
*-----*	00098600
*-----*	00098700
CALL 'ISPLINK' USING I-VDEFINE, ACTL-VAR, ACTION-LIST, I-CHAR, ACTL-VAR-STG.	00098800
CALL 'ISPLINK' USING I-VDEFINE, DN7-VAR, DEPT-NUMB, I-CHAR, DN7-VAR-STG.	00098900
CALL 'ISPLINK' USING I-VDEFINE, DNAME7-VAR, DEPT-NAME, I-CHAR, DNAME7-VAR-STG.	00099000
*	00099100
CALL 'ISPLINK' USING I-VDEFINE, ACTL-VAR, ACTION-LIST, I-CHAR, ACTL-VAR-STG.	00099200
*	00099300
*	00099400
*-----*	00099500
*-----*	00099600
*-----*	00099700
MOVE 'N' TO SEL-EXIT.	00099800
MOVE SPACES TO PREVLOC.	00099900
PERFORM 1000-MAIN-LOOP THRU 1000-MAIN-LOOP-EXIT	00100000

```

 UNTIL SEL-EXIT = 'Y'.
MOVE O TO RETURN-CODE. 00100100
MOVE SPACES TO MSGS. 00100200
CALL 'ISPLINK' USING I-VPUT, MSGS-VAR. 00100300
GOBACK. 00100400
00100500
00100600
00100700
00100800
00100900
00101000
00101100
00101200
00101300
00101400
00101500
00101600
00101700
00101800
00101900
00102000
00102100
00102200
00102300
00102400
00102500
00102600
00102700
00102800
00102900
00103000
00103100
00103200
00103300
00103400
* 00103500
* 00103600
00103700
00103800
00103900
00104000
00104100
00104200
00104300
00104400
00104500
00104600
00104700
* 00104800
* 00104900
00105000
00105100
00105200
00105300
00105400
00105500
00105600
00105700
00105800
00105900
00106000
00106100
00106200
00106300
00106400
00106500
00106600
* 00106700
* 00106800
00106900
00107000
00107100
00107200
00107300
00107400
00107500
00107600
00107700
00107800
00107900
00108000
00108100
00108200

```

```

 PERFORM 2300-GETEMPREC THRU 2300-GETEMPREC-EXIT 00108300
 CALL 'ISPLINK' USING I-VPUT, DEPT-VARS 00108400
 CALL 'ISPLINK' USING I-DISPLAY, DEPT-PANEL. 00108500
2000-ADDDDEPT-EXIT. 00108600
 EXIT. 00108700
*
----- 00108800
* DISPLAY INPUT DATA ON PANEL * 00109000
----- 00109100
2100-DISDEPTDATA. 00109200
 MOVE SPACES TO DEPT-RECORD. 00109300
 MOVE SPACES TO EMP-RECORD. 00109400
 IF SEARCH-CRIT = 'DI' THEN 00109500
 MOVE DATAW TO DEPT-NUMB. 00109600
 ELSE
 IF SEARCH-CRIT = 'DN' THEN 00109700
 MOVE DATAW TO DEPT-NAME. 00109800
 ELSE
 IF SEARCH-CRIT = 'MI' THEN 00110000
 MOVE DATAW TO DEPT-MGR. 00110100
00110200
2100-DISDEPTDATA-EXIT. 00110300
 EXIT. 00110400
*
----- 00110500
----- 00110600
* CHECK RETURN CODE FROM INSERT. IF OK, ADD TO OTHER LOCATIONS.* 00110700
----- 00110800
2200-ADDDDEPTCODES. 00110900
 IF SQLERRP = SPACES THEN 00111000
 MOVE '079E' TO MSGCODE. 00111100
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00111200
 MOVE OUTMSG TO MSGS. 00111300
 ELSE
 IF SQLCODE = -803 THEN 00111400
 MOVE '015E' TO MSGCODE. 00111500
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00111600
 MOVE OUTMSG TO MSGS. 00111700
 ELSE
 IF SQLCODE = -530 THEN 00111800
 UNSTRING SQLERRMC
 DELIMITED BY HIGH-VALUE
 INTO TOKEN. 00111900
 IF TOKEN = 'RDD' THEN 00112000
 MOVE '213E' TO MSGCODE. 00112100
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00112200
 MOVE OUTMSG TO MSGS. 00112300
 ELSE
 IF TOKEN = 'RDE' THEN 00112400
 MOVE '210E' TO MSGCODE. 00112500
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00112600
 MOVE OUTMSG TO MSGS. 00112700
 ELSE
 GO TO L8000-P3-DBERROR. 00112800
 ELSE
 IF SQLCODE NOT EQUAL TO 0 THEN 00112900
 GO TO L8000-P3-DBERROR. 00113000
 ELSE
 EXEC SQL OPEN LOCS END-EXEC. 00113100
 MOVE 0 TO LOCPTR. 00113200
 PERFORM 2210-BUILDLOCTABLE THRU
 2210-BUILDLOCTABLE-EXIT. 00113300
 UNTIL SQLCODE NOT EQUAL TO 0 00113400
 EXEC SQL CLOSE LOCS END-EXEC. 00113500
 MOVE LOCPTR TO LOCTOP. 00113600
 MOVE 0 TO LOCPTR. 00113700
 PERFORM 2220-ADDLOCS THRU 2220-ADDLOCS-EXIT
 UNTIL LOCPTR = LOCTOP. 00113800
 MOVE '012I' TO MSGCODE. 00113900
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00114000
 MOVE OUTMSG TO MSGS. 00114100
 MOVE DEPT-LOC TO LOCATION. 00114200
 PERFORM 1100-CONNECT THRU
 1100-CONNECT-EXIT. 00114300
 1100-CONNECT-EXIT. 00114400
 EXEC SQL CLOSE LOCS END-EXEC. 00114500
 MOVE LOCPTR TO LOCTOP. 00114600
 MOVE 0 TO LOCPTR. 00114700
 PERFORM 2220-ADDLOCS THRU 2220-ADDLOCS-EXIT
 UNTIL LOCPTR = LOCTOP. 00114800
 MOVE '012I' TO MSGCODE. 00114900
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00115000
 MOVE OUTMSG TO MSGS. 00115100
 MOVE DEPT-LOC TO LOCATION. 00115200
 PERFORM 1100-CONNECT THRU
 1100-CONNECT-EXIT. 00115300
 1100-CONNECT-EXIT. 00115400
 EXEC SQL CLOSE LOCS END-EXEC. 00115500
 MOVE LOCPTR TO LOCTOP. 00115600
 MOVE 0 TO LOCPTR. 00115700
 PERFORM 2220-ADDLOCS THRU 2220-ADDLOCS-EXIT
 UNTIL LOCPTR = LOCTOP. 00115800
 2220-ADDLOCS-EXIT. 00115900
*
----- 00116000
* BUILD TABLE OF UNIQUE LOCATIONS IN VHDEPT. * 00116100
----- 00116200
2210-BUILDLOCTABLE. 00116300
 EXEC SQL FETCH LOCS INTO :TEMPLOC END-EXEC. 00116400

```

```

 IF SQLCODE = 0 THEN 00116500
 ADD 1 TO LOCPTR 00116600
 MOVE TEMPLOC TO LOCLIST (LOCPTR). 00116700
 2210-BUILDLOCTABLE-EXIT. 00116800
 EXIT.
 *
 ----- 00117100
 * ADD NEW DEPARTMENT TO VHDEPT VIEWS AT ALL LOCATIONS * 00117200
 ----- 00117300
 2220-ADDLOCS.
 IF LOCPTR < LOCTOP THEN 00117400
 ADD 1 TO LOCPTR 00117500
 MOVE LOCLIST (LOCPTR) TO TEMPLOC 00117600
 EXEC SQL CONNECT TO :TEMPLOC END-EXEC 00117700
 EXEC SQL INSERT INTO VHDEPT 00117800
 VALUES (:DEPT-NUMB, :DEPT-NAME, :DEPT-MGR,
 :DEPT-ADM, :DEPT-LOC) 00117900
 END-EXEC.
 2220-ADDLOCS-EXIT. 00118000
 EXIT.
 *
 ----- 00118500
 * RETRIEVE MANAGER INFO FOR NEW DEPARTMENT * 00118700
 ----- 00118800
 2300-GETEMPREC.
 CALL 'ISPLINK' USING I-VGET, ADD-DEPT-VARS. 00118900
 EXEC SQL SELECT *
 INTO :EMP-NUMB, :EMP-FIRST-NAME,
 :EMP-MID-INIT, :EMP-LAST-NAME,
 :EMP-WORK-DEPT:WORK-DEPT-IND
 FROM VEMP
 WHERE EMPNO = :DEPT-MGR
 END-EXEC.
 IF SQLCODE = 100 THEN 00119000
 MOVE SPACES TO EMP-RECORD. 00119100
 2300-GETEMPREC-EXIT. 00119200
 EXIT.
 *
 ----- 00119300
 * ADD AN EMPLOYEE * 00119400
 ----- 00119500
 3000-ADDEMPCODES.
 CALL 'ISPLINK' USING I-VPUT, IDEN-VAR. 00119600
 PERFORM 3100-DISEMPDATA THRU 3100-DISEMPDATA-EXIT. 00119700
 CALL 'ISPLINK' USING I-VPUT, ADD-EMP-VARS. 00119800
 CALL 'ISPLINK' USING I-DISPLAY, EMP-PANEL. 00119900
 IF RETURN-CODE NOT EQUAL TO 8 THEN
 EXEC SQL OPEN CURDEPTLOC END-EXEC 00120000
 PERFORM 3320-SETCURLOC THRU 3320-SETCURLOC-EXIT. 00120100
 EXEC SQL CLOSE CURDEPTLOC END-EXEC 00120200
 EXEC SQL OPEN DEPTLOC END-EXEC 00120300
 EXEC SQL FETCH DEPTLOC INTO :DEPT-LOC END-EXEC 00120400
 EXEC SQL CLOSE DEPTLOC END-EXEC 00120500
 IF DEPT-LOC NOT EQUAL TO LOCATION THEN
 MOVE '216E' TO MSGCODE 00120600
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00120700
 MOVE OUTMSG TO MSGS 00120800
 ELSE
 EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC 00120900
 MOVE SPACES TO SQLERRP 00121000
 EXEC SQL INSERT INTO VEMP
 VALUES (:EMP-NUMB, :EMP-FIRST-NAME,
 :EMP-MID-INIT, :EMP-LAST-NAME,
 :EMP-WORK-DEPT) 00121100
 END-EXEC.
 PERFORM 3200-ADDEMPCODES THRU 3200-ADDEMPCODES-EXIT 00121200
 EXEC SQL WHENEVER SQLERROR GOTO L8000-P3-DBERROR 00121300
 END-EXEC
 PERFORM 3300-GETDEPTREC THRU 3300-GETDEPTREC-EXIT 00121400
 CALL 'ISPLINK' USING I-VPUT, DEPT-VARS 00121500
 CALL 'ISPLINK' USING I-DISPLAY, EMP-PANEL. 00121600
 3000-ADDEMPCODES-EXIT. 00121700
 EXIT.
 *
 ----- 00121800
 * DISPLAY INPUT DATA ON PANEL * 00121900
 ----- 00122000
 3100-DISEMPDATA.
 MOVE SPACES TO DEPT-RECORD. 00122100
 MOVE SPACES TO EMP-RECORD. 00122200
 IF SEARCH-CRIT = 'EI' THEN
 MOVE DATAW TO EMP-NUMB 00122300

```

```

 ELSE
 IF SEARCH-CRIT = 'EN' THEN
 MOVE DATAW TO EMP-LAST-NAME.
 3100-DISEMPDATA-EXIT.
 EXIT.
*
----- 00125300
* CHECK RETURN CODE FROM INSERT * 00125400
----- 00125500
3200-ADDEMPCODES.
 IF SQLERRP = SPACES THEN
 MOVE '079E' TO MSGCODE
 ELSE
 IF SQLCODE = -803 THEN
 MOVE '005E' TO MSGCODE
 ELSE
 IF SQLCODE = -530 THEN
 MOVE '200E' TO MSGCODE
 ELSE
 IF SQLCODE = 0 THEN
 MOVE '002I' TO MSGCODE
 ELSE
 GO TO L8000-P3-DBERROR.
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG.
 MOVE OUTMSG TO MSGS.
 3200-ADDEMPCODES-EXIT.
 EXIT.
*
----- 00127400
----- 00127500
* RETRIEVE DEPARTMENT INFO FOR NEW EMPLOYEE * 00127600
----- 00127700
3300-GETDEPTREC.
 CALL 'ISPLINK' USING I-VGET, ADD-EMP-VARS.
 EXEC SQL SELECT *
 INTO :DEPT-NUMB, :DEPT-NAME,
 :DEPT-MGR:DEPT-MGR-IND,
 :DEPT-ADM, :DEPT-LOC
 FROM VHDEPT
 WHERE DEPTNO = :EMP-WORK-DEPT
 END-EXEC.
 IF SQLCODE = 100 THEN
 MOVE SPACES TO DEPT-RECORD
 ELSE
 PERFORM 3310-CHECKDEPTIND THRU 3310-CHECKDEPTIND-EXIT.
 3300-GETDEPTREC-EXIT.
 EXIT.
*
----- 00129300
----- 00129400
* IF MGRNO NULL, MOVE BLANKS INTO FIELD * 00129500
----- 00129600
3310-CHECKDEPTIND.
 IF DEPT-MGR-IND < 0 THEN
 MOVE SPACES TO DEPT-MGR.
 3310-CHECKDEPTIND-EXIT.
 EXIT.
*
----- 00130200
----- 00130300
* SET LOCATION TO CURRENT SERVER * 00130400
----- 00130500
3320-SETCURLOC.
 IF LOCATION EQUAL TO SPACES THEN
 EXEC SQL FETCH CURDEPTLOC
 INTO :LOCATION
 END-EXEC.
 3320-SETCURLOC-EXIT.
 EXIT.
*
----- 00131300
----- 00131400
* MOVE APPROPRIATE ACTION INTO ACTION-LIST * 00131500
----- 00131600
4000-ACTION.
 IF ACTION = 'E' THEN
 MOVE 'ERASE' TO ACTION-LIST
 ELSE
 IF ACTION = 'U' THEN
 MOVE 'UPDATE' TO ACTION-LIST
 ELSE
 MOVE 'DISPLAY' TO ACTION-LIST.
 MOVE 0 TO PERCENT-COUNTER.
 INSPECT DATAW
 TALLYING PERCENT-COUNTER FOR ALL '%'.
 IF PERCENT-COUNTER > 0 THEN

```

```

 INSPECT DATAW
 REPLACING ALL ' ' BY '%'.
4000-ACTION-EXIT.
 EXIT.
*
----- * PERFORM ACTION ON DEPARTMENT OR DEPARTMENT STRUCTURE * 00133500
----- * 00133600
----- 5000-DEPARTMENT.
 IF NOT (SEARCH-CRIT = 'DI' AND PERCENT-COUNTER = 0) THEN 00133800
 MOVE DATAW TO GENDATA
 PERFORM 5100-GENDEPT THRU 5100-GENDEPT-EXIT
 UNTIL GEND-EXIT = 'Y'
 ELSE
 IF OBJFLD = 'DE' THEN
 PERFORM 5200-DISPLAYDEPT THRU 5200-DISPLAYDEPT-EXIT
 ELSE
 PERFORM 5300-STRUCTURE THRU 5300-STRUCTURE-EXIT.
5000-DEPARTMENT-EXIT.
 EXIT.
*
----- * GENERIC LIST OF DEPARTMENTS * 00135100
----- * 00135200
----- 5100-GENDEPT.
 CALL 'ISPLINK' USING I-TBCREATE, DEPT-TABLE, W-BLANK,
 SEL-DEPT-VARS, I-NOWRITE, I-REPLACE.
 MOVE SPACE TO SEL-DEPT.
 PERFORM 5110-GETDEPTTAB THRU 5110-GETDEPTTAB-EXIT.
 CALL 'ISPLINK' USING I-TBQUERY, DEPT-TABLE, W-BLANK,
 W-BLANK, QROWS.
 IF NUMROWS = 1 AND GENDATA = DATAW THEN
 MOVE 'Y' TO SPECIAL-EXIT
 CALL 'ISPLINK' USING I-TBGET, DEPT-TABLE
 MOVE DEPT-NUMB TO DATAW
 ELSE
 MOVE 'N' TO SPECIAL-EXIT
 IF NUMROWS = 0 THEN
 PERFORM 5120-DEPTMSG THRU 5120-DEPTMSG-EXIT
 MOVE 'Y' TO GEND-EXIT
 ELSE
 CALL 'ISPLINK' USING I-VPUT, ACTL-VAR
 CALL 'ISPLINK' USING I-TBTOP, DEPT-TABLE
 CALL 'ISPLINK' USING I-TBDISPL, DEPT-TABLE,
 GEND-PANEL
 IF RETURN-CODE = 8 THEN
 MOVE 'Y' TO GEND-EXIT
 ELSE
 IF ROWS-CHANGED > 0 THEN
 CALL 'ISPLINK' USING I-TBGET, DEPT-TABLE
 MOVE DEPT-NUMB TO DATAW
 ELSE
 MOVE 'Y' TO GEND-EXIT.
 IF GEND-EXIT = 'N' THEN
 IF OBJFLD = 'DE' THEN
 PERFORM 5200-DISPLAYDEPT THRU 5200-DISPLAYDEPT-EXIT
 ELSE
 PERFORM 5300-STRUCTURE THRU 5300-STRUCTURE-EXIT.
 IF SPECIAL-EXIT = 'Y' THEN
 MOVE 'Y' TO GEND-EXIT.
 CALL 'ISPLINK' USING I-TBCLOSE, DEPT-TABLE.
5100-GENDEPT-EXIT.
 EXIT.
*
----- * CREATE TABLE OF DEPARTMENTS TO FIT SEARCH-CRIT * 00139400
----- * 00139500
----- 5110-GETDEPTTAB.
 IF SEARCH-CRIT = 'DI' THEN
 EXEC SQL OPEN ALLDEPT1 END-EXEC
 MOVE SPACES TO SQLERRP
 PERFORM 5111-ALLDEPT1 THRU 5111-ALLDEPT1-EXIT
 UNTIL SQLCODE NOT EQUAL TO 0 OR GEND-EXIT = 'Y'
 EXEC SQL CLOSE ALLDEPT1 END-EXEC
 ELSE
 IF SEARCH-CRIT = 'DN' AND PERCENT-COUNTER > 0 THEN
 EXEC SQL OPEN ALLDEPT2 END-EXEC
 MOVE SPACES TO SQLERRP
 PERFORM 5112-ALLDEPT2 THRU 5112-ALLDEPT2-EXIT
 UNTIL SQLCODE NOT EQUAL TO 0 OR GEND-EXIT = 'Y'
 EXEC SQL CLOSE ALLDEPT2 END-EXEC
 ELSE

```

```

 IF SEARCH-CRIT = 'DN' THEN 00141100
 EXEC SQL OPEN ALLDEPT5 END-EXEC 00141200
 MOVE SPACES TO SQLERRP 00141300
 PERFORM 5113-ALLDEPT5 THRU 5113-ALLDEPT5-EXIT 00141400
 UNTIL SQLCODE NOT EQUAL TO 0 OR 00141500
 GEND-EXIT = 'Y' 00141600
 EXEC SQL CLOSE ALLDEPT5 END-EXEC 00141700
 ELSE 00141800
 IF SEARCH-CRIT = 'MI' AND 00141900
 PERCENT-COUNTER > 0 THEN 00142000
 EXEC SQL OPEN ALLDEPT3 END-EXEC 00142100
 MOVE SPACES TO SQLERRP 00142200
 PERFORM 5114-ALLDEPT3 THRU 00142300
 5114-ALLDEPT3-EXIT 00142400
 UNTIL SQLCODE NOT EQUAL TO 0 OR 00142500
 GEND-EXIT = 'Y' 00142600
 EXEC SQL CLOSE ALLDEPT3 END-EXEC 00142700
 ELSE 00142800
 IF SEARCH-CRIT = 'MI' THEN 00142900
 EXEC SQL OPEN ALLDEPT6 END-EXEC 00143000
 MOVE SPACES TO SQLERRP 00143100
 PERFORM 5115-ALLDEPT6 THRU 00143200
 5115-ALLDEPT6-EXIT 00143300
 UNTIL SQLCODE NOT EQUAL TO 0 OR 00143400
 GEND-EXIT = 'Y' 00143500
 EXEC SQL CLOSE ALLDEPT6 END-EXEC 00143600
 ELSE 00143700
 IF SEARCH-CRIT = 'MN' AND 00143800
 PERCENT-COUNTER > 0 THEN 00143900
 EXEC SQL OPEN ALLDEPT4 END-EXEC 00144000
 MOVE SPACES TO SQLERRP 00144100
 PERFORM 5116-ALLDEPT4 THRU 00144200
 5116-ALLDEPT4-EXIT 00144300
 UNTIL SQLCODE NOT EQUAL TO 0 OR 00144400
 OR GEND-EXIT = 'Y' 00144500
 EXEC SQL CLOSE ALLDEPT4 END-EXEC 00144600
 ELSE 00144700
 IF SEARCH-CRIT = 'MN' THEN 00144800
 EXEC SQL OPEN ALLDEPT7 END-EXEC 00144900
 MOVE SPACES TO SQLERRP 00145000
 PERFORM 5117-ALLDEPT7 THRU 00145100
 5117-ALLDEPT7-EXIT 00145200
 UNTIL SQLCODE NOT EQUAL 00145300
 TO 0 OR GEND-EXIT = 'Y' 00145400
 EXEC SQL CLOSE ALLDEPT7 END-EXEC 00145500
 END-EXEC. 00145600
 00145700
 00145800
 00145900
 00146000
* 5111-ALLDEPT1. 00146100
 EXEC SQL FETCH ALLDEPT1
 INTO :DEPT-NUMB, :DEPT-NAME,
 :DEPT-MGR:DEPT-MGR-IND,
 :MGR-NAME
 END-EXEC.
 IF SQLERRP = SPACES THEN 00146200
 MOVE '079E' TO MSGCODE 00146300
 MOVE 'Y' TO GEND-EXIT 00146400
 ELSE 00146500
 IF SQLCODE = 0 THEN 00146600
 MOVE '079E' TO MSGCODE 00146700
 MOVE 'Y' TO GEND-EXIT 00146800
 ELSE 00146900
 IF SQLCODE = 0 THEN 00147000
 PERFORM 3310-CHECKDEPTIND THRU 00147100
 3310-CHECKDEPTIND-EXIT 00147200
 CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE. 00147300
5111-ALLDEPT1-EXIT. 00147400
 EXIT. 00147500
 00147600
* 5112-ALLDEPT2. 00147700
 EXEC SQL FETCH ALLDEPT2
 INTO :DEPT-NUMB, :DEPT-NAME,
 :DEPT-MGR:DEPT-MGR-IND,
 :MGR-NAME
 END-EXEC.
 IF SQLERRP = SPACES THEN 00147800
 MOVE '079E' TO MSGCODE 00147900
 MOVE 'Y' TO GEND-EXIT 00148000
 ELSE 00148100
 IF SQLCODE = 0 THEN 00148200
 MOVE '079E' TO MSGCODE 00148300
 MOVE 'Y' TO GEND-EXIT 00148400
 ELSE 00148500
 IF SQLCODE = 0 THEN 00148600
 PERFORM 3310-CHECKDEPTIND THRU 00148700
 3310-CHECKDEPTIND-EXIT 00148800
 CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE. 00148900
5112-ALLDEPT2-EXIT. 00149000
 EXIT. 00149100
 00149200

```

```

*
5113-ALLDEPT5. 00149300
 EXEC SQL FETCH ALLDEPT5 00149400
 INTO :DEPT-NUMB, :DEPT-NAME,
 :DEPT-MGR:DEPT-MGR-IND,
 :MGR-NAME 00149500
 END-EXEC. 00149600
 IF SQLERRP = SPACES THEN 00149700
 MOVE '079E' TO MSGCODE 00149800
 MOVE 'Y' TO GEND-EXIT 00149900
 ELSE 00150000
 IF SQLCODE = 0 THEN 00150100
 PERFORM 3310-CHECKDEPTIND THRU
 3310-CHECKDEPTIND-EXIT
 CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE. 00150200
 5113-ALLDEPT5-EXIT. 00150300
 EXIT. 00150400
*
5114-ALLDEPT3. 00150500
 EXEC SQL FETCH ALLDEPT3 00150600
 INTO :DEPT-NUMB, :DEPT-NAME,
 :DEPT-MGR:DEPT-MGR-IND,
 :MGR-NAME 00150700
 END-EXEC. 00150800
 IF SQLERRP = SPACES THEN 00150900
 MOVE '079E' TO MSGCODE 00151000
 MOVE 'Y' TO GEND-EXIT 00151100
 ELSE 00151200
 IF SQLCODE = 0 THEN 00151300
 PERFORM 3310-CHECKDEPTIND THRU
 3310-CHECKDEPTIND-EXIT
 CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE. 00151400
 5114-ALLDEPT3-EXIT. 00151500
 EXIT. 00151600
*
5115-ALLDEPT6. 00151700
 EXEC SQL FETCH ALLDEPT6 00151800
 INTO :DEPT-NUMB, :DEPT-NAME,
 :DEPT-MGR:DEPT-MGR-IND,
 :MGR-NAME 00151900
 END-EXEC. 00152000
 IF SQLERRP = SPACES THEN 00152100
 MOVE '079E' TO MSGCODE 00152200
 MOVE 'Y' TO GEND-EXIT 00152300
 ELSE 00152400
 IF SQLCODE = 0 THEN 00152500
 PERFORM 3310-CHECKDEPTIND THRU
 3310-CHECKDEPTIND-EXIT
 CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE. 00152600
 5115-ALLDEPT6-EXIT. 00152700
 EXIT. 00152800
*
5116-ALLDEPT4. 00152900
 EXEC SQL FETCH ALLDEPT4 00153000
 INTO :DEPT-NUMB, :DEPT-NAME,
 :DEPT-MGR:DEPT-MGR-IND,
 :MGR-NAME 00153100
 END-EXEC. 00153200
 IF SQLERRP = SPACES THEN 00153300
 MOVE '079E' TO MSGCODE 00153400
 MOVE 'Y' TO GEND-EXIT 00153500
 ELSE 00153600
 IF SQLCODE = 0 THEN 00153700
 PERFORM 3310-CHECKDEPTIND THRU
 3310-CHECKDEPTIND-EXIT
 CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE. 00153800
 5116-ALLDEPT4-EXIT. 00153900
 EXIT. 00154000
*
5117-ALLDEPT7. 00154100
 EXEC SQL FETCH ALLDEPT7 00154200
 INTO :DEPT-NUMB, :DEPT-NAME,
 :DEPT-MGR:DEPT-MGR-IND,
 :MGR-NAME 00154300
 END-EXEC. 00154400
 IF SQLERRP = SPACES THEN 00154500
 MOVE '079E' TO MSGCODE 00154600
 MOVE 'Y' TO GEND-EXIT 00154700
 ELSE 00154800
 IF SQLCODE = 0 THEN 00154900
 PERFORM 3310-CHECKDEPTIND THRU
 3310-CHECKDEPTIND-EXIT
 CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE. 00155000
 5117-ALLDEPT7-EXIT. 00155100
 EXIT. 00155200
*

```

```

 CALL 'ISPLINK' USING I-TBADD, DEPT-TABLE. 00157500
5117-ALLDEPT7-EXIT. 00157600
 EXIT. 00157700
*
----- 00157800
----- PRINT CORRECT 'DEPARTMENT NOT FOUND' MESSAGE * 00158000
----- 00158100
5120-DEPTMSG. 00158200
 IF MSGCODE NOT EQUAL TO '079E' THEN 00158300
 IF ACTION = 'E' THEN 00158400
 MOVE '016E' TO MSGCODE 00158500
 ELSE 00158600
 IF ACTION = 'U' THEN 00158700
 MOVE '017E' TO MSGCODE 00158800
 ELSE 00158900
 MOVE '011I' TO MSGCODE 00159000
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00159100
 MOVE OUTMSG TO MSGS. 00159200
5120-DEPTMSG-EXIT. 00159300
 EXIT. 00159400
*
----- 00159500
----- DISPLAY A DEPARTMENT * 00159600
----- 00159700
----- 00159800
5200-DISPLAYDEPT. 00159900
 MOVE SPACES TO DEPT-RECORD. 00160000
 MOVE SPACES TO EMP-RECORD. 00160100
 EXEC SQL OPEN DEPT1 END-EXEC. 00160200
 MOVE SPACES TO SQLERRP. 00160300
 EXEC SQL FETCH DEPT1 INTO :DEPT-NUMB, :DEPT-NAME, 00160400
 :DEPT-MGR:DEPT-MGR-IND, 00160500
 :DEPT-ADMR, :DEPT-LOC, 00160600
 :EMP-NUMB, :EMP-FIRST-NAME, 00160700
 :EMP-MID-INIT, :EMP-LAST-NAME, 00160800
 :EMP-WORK-DEPT:WORK-DEPT-IND 00160900
 END-EXEC. 00161000
 PERFORM 5210-DISDEPTACT THRU 5210-DISDEPTACT-EXIT. 00161100
5200-DISPLAYDEPT-EXIT. 00161200
 EXIT. 00161300
*
----- 00161400
----- DISPLAY, ERASE, OR UPDATE DEPARTMENT * 00161500
----- 00161600
----- 00161700
5210-DISDEPTACT. 00161800
 IF SQLERRP = SPACES THEN 00161900
 EXEC SQL CLOSE DEPT1 END-EXEC 00162000
 MOVE '079E' TO MSGCODE 00162100
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00162200
 MOVE OUTMSG TO MSGS. 00162300
 ELSE 00162400
 IF SQLCODE = 100 THEN 00162500
 EXEC SQL CLOSE DEPT1 END-EXEC 00162600
 PERFORM 5120-DEPTMSG THRU 5120-DEPTMSG-EXIT 00162700
 ELSE 00162800
 EXEC SQL CLOSE DEPT1 END-EXEC 00162900
 PERFORM 3310-CHECKDEPTIND THRU 00163000
 3310-CHECKDEPTIND-EXIT 00163100
 CALL 'ISPLINK' USING I-DISPLAY, DEPT-PANEL 00163200
 IF RETURN-CODE NOT EQUAL TO 8 THEN 00163300
 IF ACTION = 'E' THEN 00163400
 PERFORM 5220-ERASEDEPT THRU 00163500
 5220-ERASEDEPT-EXIT 00163600
 ELSE 00163700
 IF ACTION = 'U' THEN 00163800
 PERFORM 5230-UPDATEDEPT THRU 00163900
 5230-UPDATEDEPT-EXIT 00164000
 5210-DISDEPTACT-EXIT. 00164100
 EXIT. 00164200
*
----- 00164300
----- ERASE A DEPARTMENT * 00164400
----- 00164500
----- 00164600
5220-ERASEDEPT. 00164700
 MOVE 1 TO DEPTPTR. 00164800
 MOVE 0 TO LISTPTR. 00164900
 MOVE DATAW TO DEPTS (DEPTPTR). 00165000
 PERFORM 5221-DELDEPTS THRU 5221-DELDEPTS-EXIT 00165100
 UNTIL DEPTPTR = 0. 00165200
 MOVE LISTPTR TO STACKTOP. 00165300
 PERFORM 5223-DELDEPEND THRU 5223-DELDEPEND-EXIT 00165400
 UNTIL LISTPTR = 0. 00165500
 EXEC SQL OPEN LOCS END-EXEC. 00165600

```

```

MOVE 0 TO LOCPTR.
PERFORM 2210-BUILDLOCTABLE THRU 2210-BUILDLOCTABLE-EXIT
 UNTIL SQLCODE NOT EQUAL TO 0.
EXEC SQL CLOSE LOCS END-EXEC.
MOVE LOCPTR TO LOCTOP.
MOVE 0 TO LOCPTR.
PERFORM 5224-DELETELOCs THRU 5224-DELETELOCs-EXIT
 UNTIL LOCPTR = LOCTOP.
MOVE '013I' TO MSGCODE.
CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG.
MOVE OUTMSG TO MSGS.
PERFORM 1100-CONNECT THRU 1100-CONNECT-EXIT.
5220-ERASEDEPT-EXIT.
 EXIT.

-----* ERASE DEPARTMENT FROM OTHER LOCATIONS -----*
* 00167200
* 00167300
* 00167400

5221-DELDEPTS.
 ADD 1 TO LISTPTR.
 MOVE DEPTS (DEPTPTR) TO DEPTLIST (LISTPTR).
 MOVE DEPTS (DEPTPTR) TO CURDEPT.
 SUBTRACT 1 FROM DEPTPTR.
 EXEC SQL OPEN SUBDEPTS END-EXEC.
 PERFORM 5222-GETSUBDEPTS THRU 5222-GETSUBDEPTS-EXIT
 UNTIL SQLCODE NOT EQUAL TO 0.
 EXEC SQL CLOSE SUBDEPTS END-EXEC.
5221-DELDEPTS-EXIT.
 EXIT.

-----* BUILD TABLE OF DEPARTMENTS DEPENDENT ON ERASED DEPARTMENTS -----*
* 00168700
* 00168800
* 00168900
* 00169000

5222-GETSUBDEPTS.
 EXEC SQL FETCH SUBDEPTS INTO :TEMPDEPT END-EXEC.
 IF SQLCODE = 0 THEN
 ADD 1 TO DEPTPTR
 MOVE TEMPDEPT TO DEPTS (DEPTPTR).
5222-GETSUBDEPTS-EXIT.
 EXIT.

-----* ENFORCE REFERENTIAL INTEGRITY RULE ON VHDEPT BY CASCADE -----*
* 00169900
* 00170000
* 00170100
* 00170200

5223-DELDEPEND.
 MOVE DEPTLIST (LISTPTR) TO DELDEPT.
 EXEC SQL DELETE FROM VHDEPT
 WHERE DEPTNO = :DELDEPT
 END-EXEC.
 SUBTRACT 1 FROM LISTPTR.
5223-DELDEPEND-EXIT.
 EXIT.

-----* PERFORM CASCADE DELETE AT ALL LOCATIONS -----*
* 00171200
* 00171300
* 00171400

5224-DELETELOCs.
 IF LOCPTR < LOCTOP THEN
 ADD 1 TO LOCPTR
 MOVE LOCLIST (LOCPTR) TO TEMPLOC
 EXEC SQL CONNECT TO :TEMPLOC END-EXEC
 MOVE STACKTOP TO LISTPTR
 PERFORM 5223-DELDEPEND THRU 5223-DELDEPEND-EXIT
 UNTIL LISTPTR = 0.
5224-DELETELOCs-EXIT.
 EXIT.

-----* UPDATE A DEPARTMENT -----*
* 00172600
* 00172700
* 00172800

5230-UPDATEDEPT.
 PERFORM 2300-GETEMPREC THRU 2300-GETEMPREC-EXIT.
 EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
 EXEC SQL UPDATE VHDEPT
 SET DEPTNAME = :DEPT-NAME,
 MGRNO = :DEPT-MGR,
 ADMRDEPT = :DEPT-ADMR,
 LOCATION = :DEPT-LOC
 WHERE DEPTNO = :DATAW
 END-EXEC.

```

```

IF SQLCODE = -530 THEN 00173900
 UNSTRING SQLERRMC 00174000
 DELIMITED BY HIGH-VALUE
 INTO TOKEN 00174100
 IF TOKEN = 'RDD' THEN 00174200
 MOVE '215E' TO MSGCODE 00174300
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00174400
 MOVE OUTMSG TO MSGS 00174500
 ELSE
 IF TOKEN = 'RDE' THEN 00174600
 MOVE '214E' TO MSGCODE 00174700
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00174800
 MOVE OUTMSG TO MSGS 00174900
 ELSE
 GO TO L8000-P3-DBERROR 00175000
 ELSE
 IF SQLCODE NOT EQUAL TO 0 THEN 00175100
 GO TO L8000-P3-DBERROR 00175200
 ELSE
 EXEC SQL WHENEVER SQLERROR GOTO L8000-P3-DBERROR 00175300
 END-EXEC
 EXEC SQL OPEN LOCS END-EXEC 00175400
 MOVE 0 TO LOCPTR 00175500
 PERFORM 2210-BUILDOBJECT THRU
 2210-BUILDOBJECT-EXIT 00175600
 UNTIL SQLCODE NOT EQUAL TO 0
 EXEC SQL CLOSE LOCS END-EXEC 00175700
 MOVE LOCPTR TO LOCTOP 00175800
 MOVE 0 TO LOCPTR 00175900
 PERFORM 5231-UPDATELOCS THRU
 5231-UPDATELOCS-EXIT 00176000
 UNTIL LOCPTR = LOCTOP
 MOVE '014I' TO MSGCODE 00176100
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00176200
 MOVE OUTMSG TO MSGS 00176300
 PERFORM 1100-CONNECT THRU 1100-CONNECT-EXIT 00176400
 EXEC SQL WHENEVER SQLERROR GOTO L8000-P3-DBERROR END-EXEC. 00176500
 CALL 'ISPLINK' USING I-DISPLAY, DEPT-PANEL. 00176600
 5230-UPDATEDEPT-EXIT.
 EXIT.
*
----- 00177000
----- 00177100
----- 00177200
----- 00177300
----- 00177400
----- 00177500
----- 00177600
----- 00177700
----- 00177800
----- 00177900
----- 00178000
----- 00178100
----- 00178200
----- 00178300
5231-UPDATELOCS.
 IF LOCPTR < LOCTOP THEN 00178400
 ADD 1 TO LOCPTR 00178500
 MOVE LOCLIST (LOCPTR) TO TEMPLOC 00178600
 EXEC SQL CONNECT TO :TEMPLOC END-EXEC 00178700
 EXEC SQL UPDATE VHDEPT 00178800
 SET DEPTNAME = :DEPT-NAME, 00178900
 MGRNO = :DEPT-MGR, 00179000
 ADMRDEPT = :DEPT-ADMR, 00179100
 LOCATION = :DEPT-LOC, 00179200
 WHERE DEPTNO = :DEPT-NUMB 00179300
 END-EXEC.
 5231-UPDATELOCS-EXIT.
 EXIT.
*
----- 00179400
----- 00179500
----- 00179600
----- 00179700
----- 00179800
----- 00179900
----- 00180000
5300-STRUCTURE.
 MOVE SPACES TO DEPT-RECORD. 00180100
 MOVE SPACES TO EMP-RECORD. 00180200
 MOVE SPACES TO DEPT1-RECORD. 00180300
 MOVE SPACES TO EMP1-RECORD. 00180400
 EXEC SQL OPEN DEPT1 END-EXEC. 00180500
 MOVE SPACES TO SQLERRP. 00180600
 EXEC SQL FETCH DEPT1 INTO :DEPT1-NUMB, :DEPT1-NAME, 00180700
 :DEPT1-MGR:DEPT1-MGR-IND, 00180800
 :DEPT1-ADMR, :DEPT1-LOC, 00180900
 :EMP-NUMB, 00181000
 :EMP1-FIRST-NAME, 00181100
 :EMP1-MID-INIT, 00181200
 :EMP1-LAST-NAME, 00181300
 :EMP1-WORK-DEPT:WORK1-DEPT-IND 00181400
 END-EXEC.
 PERFORM 5310-DISSTR THRU 5310-DISSTR-EXIT. 00181500
 5300-STRUCTURE-EXIT.
 EXIT.
*

```

```

----- 00182100
* DISPLAY DEPARTMENTS REPORTING TO SELECTED DEPARTMENT * 00182200
----- 00182300
----- 00182400
5310-DISSTR.
 IF SQLERRP = SPACES THEN 00182500
 EXEC SQL CLOSE DEPT1 END-EXEC
 MOVE '079E' TO MSGCODE 00182600
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00182700
 MOVE OUTMSG TO MSGS 00182800
 ELSE
 IF SQLCODE = 100 THEN 00182900
 EXEC SQL CLOSE DEPT1 END-EXEC
 MOVE '011I' TO MSGCODE 00183000
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00183100
 MOVE OUTMSG TO MSGS 00183200
 ELSE
 EXEC SQL CLOSE DEPT1 END-EXEC
 PERFORM 5311-CHECKDEPT1IND THRU 00183300
 5311-CHECKDEPT1IND-EXIT
 CALL 'ISPLINK' USING I-TBCREATE, DS-TABLE, W-BLANK, 00183400
 DS-VARS, I-NOWRITE, I-REPLACE 00183500
 EXEC SQL OPEN DEPTSTR END-EXEC
 PERFORM 5312-GETSTRTAB THRU 5312-GETSTRTAB-EXIT 00183600
 UNTIL SQLCODE NOT EQUAL TO 0
 EXEC SQL CLOSE DEPTSTR END-EXEC
 CALL 'ISPLINK' USING I-TBTOP, DS-TABLE 00183700
 CALL 'ISPLINK' USING I-TBDISPL, DS-TABLE, STR-PANEL 00183800
 CALL 'ISPLINK' USING I-VPUT, HEAD-DEPT-VARS 00183900
 CALL 'ISPLINK' USING I-TBCLOSE, DS-TABLE. 00184000
 5310-DISSTR-EXIT.
 EXIT.
*
----- 00185000
----- 00185100
----- 00185200
----- 00185300
----- 00185400
----- 00185500
5311-CHECKDEPT1IND.
 IF DEPT1-MGR-IND < 0 THEN 00185600
 MOVE SPACES TO DEPT1-MGR. 00185700
 5311-CHECKDEPT1IND-EXIT.
 EXIT.
*
----- 00186000
----- 00186100
----- 00186200
----- 00186300
----- 00186400
5312-GETSTRTAB.
 EXEC SQL FETCH DEPTSTR
 INTO :DEPT-NUMB, :DEPT-NAME, 00186500
 :DEPT-MGR:DEPT-MGR-IND, 00186600
 :DEPT-ADMR, :DEPT-LOC, 00186700
 :EMP-FIRST-NAME, :EMP-MID-INIT, 00186800
 :EMP-LAST-NAME 00186900
 END-EXEC.
 IF SQLCODE = 0 THEN 00187000
 PERFORM 3310-CHECKDEPTIND THRU 3310-CHECKDEPTIND-EXIT 00187100
 CALL 'ISPLINK' USING I-TBADD, DS-TABLE. 00187200
 5312-GETSTRTAB-EXIT.
 EXIT.
*
----- 00187300
----- 00187400
----- 00187500
----- 00187600
----- 00187700
----- 00187800
----- 00187900
----- 00188000
----- 00188100
----- 00188200
----- 00188300
----- 00188400
----- 00188500
----- 00188600
----- 00188700
----- 00188800
----- 00188900
----- 00189000
----- 00189100
----- 00189200
----- 00189300
----- 00189400
6000-EMPLOYEE.
 IF NOT (SEARCH-CRIT = 'EI' AND PERCENT-COUNTER = 0) THEN 00189500
 MOVE DATAW TO GENDATA 00189600
 PERFORM 6100-GENEMP THRU 6100-GENEMP-EXIT 00189700
 UNTIL GENE-EXIT = 'Y'
 ELSE
 PERFORM 6200-DISPLAYEMP THRU 6200-DISPLAYEMP-EXIT. 00189800
 6000-EMPLOYEE-EXIT.
 EXIT.
*
----- 00189900
----- 00190000
----- 00190100
----- 00190200
6100-GENEMP.
 CALL 'ISPLINK' USING I-TBCREATE, EMP-TABLE, W-BLANK, 00190300
 SEL-EMP-VARS, I-NOWRITE, I-REPLACE. 00190400
 MOVE SPACE TO SEL-EMP. 00190500
 PERFORM 6110-GETEMPTAB THRU 6110-GETEMPTAB-EXIT. 00190600
 CALL 'ISPLINK' USING I-TBQUERY, EMP-TABLE, W-BLANK, W-BLANK, 00190700
 QROWS.
 IF NUMROWS = 1 AND DATAW = GENDATA THEN 00190800

```

```

MOVE 'Y' TO SPECIAL-EXIT 00190300
CALL 'ISPLINK' USING I-TBGET, EMP-TABLE 00190400
MOVE EMP-NUMB TO DATAW 00190500
ELSE
MOVE 'N' TO SPECIAL-EXIT 00190600
IF NUMROWS = 0 THEN
 PERFORM 6120-EMPMMSG THRU 6120-EMPMMSG-EXIT 00190700
 MOVE 'Y' TO GENE-EXIT 00190800
ELSE
CALL 'ISPLINK' USING I-VPUT, ACTL-VAR 00191200
CALL 'ISPLINK' USING I-TBTOP, EMP-TABLE 00191300
CALL 'ISPLINK' USING I-TBDISPL, EMP-TABLE,
 GENE-PANEL 00191400
 IF RETURN-CODE = 8 THEN 00191500
 MOVE 'Y' TO GENE-EXIT 00191600
 ELSE
 IF ROWS-CHANGED > 0 THEN 00191700
 CALL 'ISPLINK' USING I-TBGET, EMP-TABLE 00192000
 MOVE EMP-NUMB TO DATAW 00192100
 ELSE
 MOVE 'Y' TO GENE-EXIT. 00192200
 IF GENE-EXIT = 'N' THEN 00192300
 PERFORM 6200-DISPLAYEMP THRU 6200-DISPLAYEMP-EXIT. 00192400
 IF SPECIAL-EXIT = 'Y' THEN 00192500
 MOVE 'Y' TO GENE-EXIT. 00192600
 CALL 'ISPLINK' USING I-TBCLOSE, EMP-TABLE. 00192800
6100-GENEMP-EXIT.
 EXIT. 00192900
*
----- 00193100
----- * 00193200
----- * CREATE TABLE OF EMPLOYEES TO FIT SEARCH-CRIT * 00193300
----- *-----* 00193400
6110-GETEMPTAB. 00193500
 IF SEARCH-CRIT = 'EI' THEN 00193600
 EXEC SQL OPEN ALLEMP1 END-EXEC 00193700
 MOVE SPACES TO SQLERRP 00193800
 PERFORM 6111-ALLEMP1 THRU 6111-ALLEMP1-EXIT 00193900
 UNTIL SQLCODE NOT EQUAL TO 0 OR GENE-EXIT = 'Y' 00194000
 EXEC SQL CLOSE ALLEMP1 END-EXEC 00194100
 ELSE
 IF SEARCH-CRIT = 'EN' AND PERCENT-COUNTER > 0 THEN 00194200
 EXEC SQL OPEN ALLEMP2 END-EXEC 00194300
 MOVE SPACES TO SQLERRP 00194400
 PERFORM 6112-ALLEMP2 THRU 6112-ALLEMP2-EXIT 00194500
 UNTIL SQLCODE NOT EQUAL TO 0 OR GENE-EXIT = 'Y' 00194600
 EXEC SQL CLOSE ALLEMP2 END-EXEC 00194700
 ELSE
 EXEC SQL OPEN ALLEMP3 END-EXEC 00194800
 MOVE SPACES TO SQLERRP 00194900
 PERFORM 6113-ALLEMP3 THRU 6113-ALLEMP3-EXIT 00195000
 UNTIL SQLCODE NOT EQUAL TO 0 OR GENE-EXIT = 'Y' 00195100
 EXEC SQL CLOSE ALLEMP3 END-EXEC 00195200
 6110-GETEMPTAB-EXIT. 00195300
 EXIT. 00195400
*
----- 00195500
----- 00195600
6111-ALLEMP1.
 EXEC SQL FETCH ALLEMP1 00195700
 INTO :EMP-NUMB, :EMP-NAME,
 :EMP-WORK-DEPT:WORK-DEPT-IND,
 :DEPT-NAME
 END-EXEC. 00195800
 IF SQLERRP = SPACES THEN 00195900
 MOVE '079E' TO MSGCODE 00196000
 MOVE 'Y' TO GENE-EXIT 00196100
 ELSE
 IF SQLCODE = 0 THEN 00196200
 PERFORM 6114-CHECKEMPIND THRU 6114-CHECKEMPIND-EXIT 00196300
 CALL 'ISPLINK' USING I-TBADD, EMP-TABLE. 00196400
 6111-ALLEMP1-EXIT. 00196500
 EXIT. 00196600
*
----- 00196700
----- 00196800
6112-ALLEMP2.
 EXEC SQL FETCH ALLEMP2 00196900
 INTO :EMP-NUMB, :EMP-NAME,
 :EMP-WORK-DEPT:WORK-DEPT-IND,
 :DEPT-NAME
 END-EXEC. 00197000
 IF SQLERRP = SPACES THEN 00197100
 MOVE '079E' TO MSGCODE 00197200
 MOVE 'Y' TO GENE-EXIT 00197300
 ELSE
 IF SQLCODE = 0 THEN 00197400
 PERFORM 6114-CHECKEMPIND THRU 6114-CHECKEMPIND-EXIT 00197500
 CALL 'ISPLINK' USING I-TBADD, EMP-TABLE. 00197600
 6112-ALLEMP2-EXIT. 00197700
 EXIT. 00197800
*
----- 00197900
----- 00198000
----- 00198100
----- 00198200
----- 00198300
----- 00198400

```

```

 PERFORM 6114-CHECKKEMPIND THRU 6114-CHECKKEMPIND-EXIT 00198500
 CALL 'ISPLINK' USING I-TBADD, EMP-TABLE. 00198600
6112-ALLEMP2-EXIT. 00198700
 EXIT. 00198800
*
6113-ALLEMP3. 00198900
 EXEC SQL FETCH ALLEMP3 00199000
 INTO :EMP-NUMB, :EMP-NAME,
 :EMP-WORK-DEPT:WORK-DEPT-IND,
 :DEPT-NAME
 END-EXEC. 00199100
 IF SQLERRP = SPACES THEN 00199200
 MOVE '079E' TO MSGCODE 00199300
 MOVE 'Y' TO GENE-EXIT 00199400
 ELSE 00199500
 IF SQLCODE = 0 THEN 00199600
 PERFORM 6114-CHECKKEMPIND THRU 6114-CHECKKEMPIND-EXIT 00200100
 CALL 'ISPLINK' USING I-TBADD, EMP-TABLE. 00200200
6113-ALLEMP3-EXIT. 00200300
 EXIT. 00200400
*
----- 00200500
----- 00200600
* IF WORKDEPT NULL, MOVE BLANKS INTO FIELD * 00200700
----- 00200800
6114-CHECKKEMPIND. 00200900
 IF WORK-DEPT-IND < 0 THEN 00201000
 MOVE SPACES TO EMP-WORK-DEPT. 00201100
6114-CHECKKEMPIND-EXIT. 00201200
 EXIT. 00201300
*
----- 00201400
----- 00201500
* PRINT CORRECT 'EMPLOYEE NOT FOUND' MESSAGE * 00201600
----- 00201700
6120-EMPPMSG. 00201800
 IF MSGCODE NOT EQUAL TO '079E' THEN 00201900
 IF ACTION = 'E' THEN 00202000
 MOVE '006E' TO MSGCODE 00202100
 ELSE 00202200
 IF ACTION = 'U' THEN 00202300
 MOVE '007E' TO MSGCODE 00202400
 ELSE 00202500
 MOVE '001I' TO MSGCODE. 00202600
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00202700
 MOVE OUTMSG TO MSGS. 00202800
6120-EMPPMSG-EXIT. 00202900
 EXIT. 00203000
*
----- 00203100
----- 00203200
* DISPLAY AN EMPLOYEE * 00203300
----- 00203400
6200-DISPLAYEMP. 00203500
 MOVE SPACES TO DEPT-RECORD. 00203600
 MOVE SPACES TO EMP-RECORD. 00203700
 EXEC SQL OPEN EMP1 END-EXEC. 00203800
 MOVE SPACES TO SQLERRP. 00203900
 EXEC SQL FETCH EMP1 INTO :DEPT-NUMB, :DEPT-NAME,
 :DEPT-MGR:DEPT-MGR-IND,
 :DEPT-ADMR, :DEPT-LOC,
 :EMP-NUMB, :EMP-FIRST-NAME,
 :EMP-MID-INIT, :EMP-LAST-NAME,
 :EMP-WORK-DEPT:WORK-DEPT-IND
 END-EXEC. 00204000
 PERFORM 6210-DISEMPACT THRU 6210-DISEMPACT-EXIT. 00204700
6200-DISPLAYEMP-EXIT. 00204800
 EXIT. 00204900
*
----- 00205000
----- 00205100
* DISPLAY, ERASE, OR UPDATE EMPLOYEE * 00205200
----- 00205300
6210-DISEMPACT. 00205400
 IF SQLERRP = SPACES THEN 00205500
 EXEC SQL CLOSE EMP1 END-EXEC 00205600
 MOVE '079E' TO MSGCODE 00205700
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00205800
 MOVE OUTMSG TO MSGS 00205900
 ELSE 00206000
 IF SQLCODE = 100 THEN 00206100
 EXEC SQL CLOSE EMP1 END-EXEC 00206200
 PERFORM 6120-EMPPMSG THRU 6120-EMPPMSG-EXIT 00206300
 ELSE 00206400
 EXEC SQL CLOSE EMP1 END-EXEC 00206500
 PERFORM 3310-CHECKDEPTIND THRU 00206600

```

```

 3310-CHECKDEPTIND-EXIT 00206700
CALL 'ISPLINK' USING I-DISPLAY, EMP-PANEL 00206800
IF RETURN-CODE NOT EQUAL TO 8 THEN 00206900
 IF ACTION = 'E' THEN 00207000
 PERFORM 6220-ERASEEMP THRU 00207100
 6220-ERASEEMP-EXIT 00207200
 ELSE 00207300
 IF ACTION = 'U' THEN 00207400
 PERFORM 6230-UPDATEEMP THRU 00207500
 6230-UPDATEEMP-EXIT. 00207600
6210-DISEMPACT-EXIT. 00207700
 EXIT. 00207800
*
----- 00208000
* ERASE AN EMPLOYEE * 00208100
----- 00208200
6220-ERASEEMP. 00208300
 EXEC SQL DELETE FROM VEMP 00208400
 WHERE EMPNO = :DATAW 00208500
 END-EXEC. 00208600
 IF SQLCODE = 0 THEN 00208700
 MOVE '003I' TO MSGCODE 00208800
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00208900
 MOVE OUTMSG TO MSGS. 00209000
6220-ERASEEMP-EXIT. 00209100
 EXIT. 00209200
*
----- 00209300
* UPDATE AN EMPLOYEE * 00209400
----- 00209500
----- 00209600
6230-UPDATEEMP. 00209700
 PERFORM 3300-GETDEPTREC THRU 3300-GETDEPTREC-EXIT. 00209800
 EXEC SQL OPEN CURDEPTLOC END-EXEC 00209900
 PERFORM 3320-SETCURLOC THRU 3320-SETCURLOC-EXIT. 00210000
 EXEC SQL CLOSE CURDEPTLOC END-EXEC 00210100
 IF DEPT-LOC NOT EQUAL TO LOCATION THEN 00210200
 MOVE '217E' TO MSGCODE 00210300
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00210400
 MOVE OUTMSG TO MSGS. 00210500
 ELSE 00210600
 EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC 00210700
 EXEC SQL UPDATE VEMP 00210800
 SET FIRSTNAME = :EMP-FIRST-NAME, 00210900
 MIDINIT = :EMP-MID-INIT, 00211000
 LASTNAME = :EMP-LAST-NAME, 00211100
 WORKDEPT = :EMP-WORK-DEPT 00211200
 WHERE EMPNO = :DATAW 00211300
 END-EXEC. 00211400
 IF SQLCODE = -530 THEN 00211500
 MOVE '203E' TO MSGCODE 00211600
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00211700
 MOVE OUTMSG TO MSGS. 00211800
 ELSE 00211900
 IF SQLCODE = 0 THEN 00212000
 MOVE '004I' TO MSGCODE 00212100
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00212200
 MOVE OUTMSG TO MSGS. 00212300
 ELSE 00212400
 GO TO L8000-P3-DBERROR. 00212500
 CALL 'ISPLINK' USING I-DISPLAY, EMP-PANEL. 00212600
6230-UPDATEEMP-EXIT. 00212700
 EXIT. 00212800
*
----- 00212900
* DB2 ERROR PROCESSING * 00213000
----- 00213100
----- 00213200
L8000-P3-DBERROR.

 MOVE SQLCAID TO SQLCAID-VALUE. 00213300
 MOVE SQLCABC TO CONV. 00213400
 MOVE CONV TO SQLCABC-VALUE. 00213500
 MOVE SQLCODE TO CONV. 00213600
 MOVE CONV TO SQLCODE-VALUE, SQLCODE-MSG. 00213700
 MOVE SQLERRML TO CONV. 00213800
 MOVE CONV TO SQLERRML-VALUE. 00213900
 MOVE SQLERRMC TO SQLERRMC-VALUE. 00214000
 MOVE SQLERRP TO SQLERRP-VALUE. 00214100
 MOVE SQLERRD (1) TO CONV. 00214200
 MOVE CONV TO SQLERRD1-VALUE. 00214300
 MOVE SQLERRD (2) TO CONV. 00214400
 MOVE CONV TO SQLERRD2-VALUE. 00214500
 MOVE SQLERRD (3) TO CONV. 00214600

```

```

MOVE CONV TO SQLERRD3-VALUE. 00214900
MOVE SQLERRD (4) TO CONV. 00215000
MOVE CONV TO SQLERRD4-VALUE. 00215100
MOVE SQLERRD (5) TO CONV. 00215200
MOVE CONV TO SQLERRD5-VALUE. 00215300
MOVE SQLERRD (6) TO CONV. 00215400
MOVE CONV TO SQLERRD6-VALUE. 00215500
MOVE SQLWARN0 TO SQLWARN0-VALUE. 00215600
MOVE SQLWARN1 TO SQLWARN1-VALUE. 00215700
MOVE SQLWARN2 TO SQLWARN2-VALUE. 00215800
MOVE SQLWARN3 TO SQLWARN3-VALUE. 00215900
MOVE SQLWARN4 TO SQLWARN4-VALUE. 00216000
MOVE SQLWARN5 TO SQLWARN5-VALUE. 00216100
MOVE SQLWARN6 TO SQLWARN6-VALUE. 00216200
MOVE SQLWARN7 TO SQLWARN7-VALUE. 00216300
MOVE SQLWARN8 TO SQLWARN8-VALUE. 00216400
MOVE SQLWARN9 TO SQLWARN9-VALUE. 00216500
MOVE SQLWARNA TO SQLWARNA-VALUE. 00216600
MOVE SQLSTATE TO SQLSTATE-VALUE. 00216700
 00216800
OPEN OUTPUT MSGOUT. 00216900
WRITE MSGREC FROM SQLCA-LINE0. 00217000
WRITE MSGREC FROM SQLCA-LINE1. 00217100
WRITE MSGREC FROM SQLCA-LINE2. 00217200
WRITE MSGREC FROM SQLCA-LINE3. 00217300
WRITE MSGREC FROM SQLCA-LINE4. 00217400
WRITE MSGREC FROM SQLCA-LINE5. 00217500
WRITE MSGREC FROM SQLCA-LINE6. 00217600
WRITE MSGREC FROM SQLCA-LINE7. 00217700
WRITE MSGREC FROM SQLCA-LINE8. 00217800
WRITE MSGREC FROM SQLCA-LINE9. 00217900
WRITE MSGREC FROM SQLCA-LINE10. 00218000
WRITE MSGREC FROM SQLCA-LINE11. 00218100
WRITE MSGREC FROM SQLCA-LINE12. 00218200
WRITE MSGREC FROM SQLCA-LINE13. 00218300
WRITE MSGREC FROM SQLCA-LINE14. 00218400
CLOSE MSGOUT. 00218500
 00218600
GOBACK. 00218700

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8SC3

ESTE MÓDULO LISTA LOS NÚMEROS DE TELÉFONO DE EMPLEADOS Y LOS ACTUALIZA SI LO DESEA.

```

IDENTIFICATION DIVISION. 00000100
*----- 00000200
PROGRAM-ID. DSN8SC3. 00000300
 00000400
*----- * 00000500
* * 00000600
* MODULE NAME = DSN8SC3 * 00000700
* * 00000800
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION * 00000900
* PHONE APPLICATION * 00001000
* ISPF * 00001100
* COBOL * 00001200
* * 00001300
*COPYRIGHT = 5615-DB2 (C) COPYRIGHT 1982, 2013 IBM CORP. * 00001400
*SEE COPYRIGHT INSTRUCTIONS * 00001500
*LICENSED MATERIALS - PROPERTY OF IBM * 00001600
* * 00001700
*STATUS = STATUS = VERSION 11 * 00001800
* * 00001900
* FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND * 00002000
* UPDATES THEM IF DESIRED. * 00002100
* * 00002200
* NOTES = * 00002300
* DEPENDENCIES = TWO ISPF PANELS ARE REQUIRED: * 00002400
* DSN8SSL AND DSN8SSN * 00002500
* RESTRICTIONS = NONE * 00002600
* * 00002700
* MODULE TYPE = VS COBOL II PROGRAM * 00002800
* PROCESSOR = DB2 PRECOMPILER, VS COBOL II * 00002900
* MODULE SIZE = SEE LINKEDIT * 00003000

```

```

* ATTRIBUTES = NOT REENTRANT OR REUSABLE * 00003100
* * 00003200
* ENTRY POINT = DSN8SC3 * 00003300
* PURPOSE = SEE FUNCTION * 00003400
* LINKAGE = INVOKED FROM ISPF * 00003500
* * 00003600
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION: * 00003700
* INPUT-MESSAGE: * 00003800
* * 00003900
* SYMBOLIC LABEL/NAME = DSN8SSL * 00004000
* DESCRIPTION = PHONE MENU 1 (SELECT) * 00004100
* * 00004200
* SYMBOLIC LABEL/NAME = DSN8SSN * 00004300
* DESCRIPTION = PHONE MENU 2 (LIST) * 00004400
* * 00004500
* SYMBOLIC LABEL/NAME = VPHONE * 00004600
* DESCRIPTION = VIEW OF TELEPHONE DATA * 00004700
* * 00004800
* SYMBOLIC LABEL/NAME = VEMPLP * 00004900
* DESCRIPTION = VIEW OF EMPLOYEE DATA * 00005000
* * 00005100
* OUTPUT = PARAMETERS EXPLICITLY RETURNED: * 00005200
* OUTPUT-MESSAGE: * 00005300
* * 00005400
* SYMBOLIC LABEL/NAME = DSN8SSL * 00005500
* DESCRIPTION = PHONE MENU 1 (SELECT) * 00005600
* * 00005700
* SYMBOLIC LABEL/NAME = DSN8SSN * 00005800
* DESCRIPTION = PHONE MENU 2 (LIST) * 00005900
* * 00006000
* EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION * 00006100
* * 00006200
* EXIT-ERROR = * 00006300
* * 00006400
* RETURN CODE = NONE * 00006500
* * 00006600
* ABEND CODES = NONE * 00006700
* * 00006800
* * 00006900
* ERROR-MESSAGES = * 00007000
* DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED * 00007100
* DSN8008I - NO EMPLOYEE FOUND IN TABLE * 00007200
* DSN8060E - SQL ERROR, RETURN CODE IS: * 00007300
* DSN8079E - CONNECTION TO DB2 NOT ESTABLISHED * 00007400
* * 00007500
* EXTERNAL REFERENCES = * 00007600
* ROUTINES/SERVICES = * 00007700
* DSN8MCG - ERROR MESSAGE ROUTINE * 00007800
* ISLINK - ISPF SERVICES ROUTINE * 00007900
* * 00008000
* DATA-AREAS = * 00008100
* NONE * 00008200
* * 00008300
* CONTROL-BLOCKS = * 00008400
* SQLCA - SQL COMMUNICATION AREA * 00008500
* * 00008600
* TABLES = NONE * 00008700
* * 00008800
* * 00008900
* CHANGE-ACTIVITY: * 00009000
* * 00009100
* CHECK SQLERRP FOR NON-BLANKS TO ENSURE CONNECTION V2R3 * 00009200
* HAS BEEN ESTABLISHED. ISSUE 079E IF NOT. * 00009300
* * 00009400
* * 00009500
* * 00009600
* SET UP RETURN CODE HANDLING 0000-PROGRAM-START * 00009700
* DO UNTIL NO MORE TERMINAL INPUT * 00009800
* GET PANEL INPUT 1000-MAIN-LOOP * 00009900
* DETERMINE PROCESSING REQUEST 2000-GET-TYPE * 00010000
* -IF "LIST ALL" (*):
* FETCH FIRST RECORD * 00010100
* CREATE ISPF TABLE * 00010200
* DO UNTIL NO MORE RECORDS: * 00010300
* STORE RECORD IN TABLE 3500-LIST-AND-GET * 00010400
* GET ANOTHER RECORD * 00010500
* -IF "LIST GENERIC" (%):
* FETCH FIRST RECORD * 00010600
* CREATE ISPF TABLE * 00010700
* DO UNTIL NO MORE MATCHING RECORDS: * 00010800
* STORE RECORD IN TABLE 4500-LIST-AND-GET * 00010900
* GET ANOTHER RECORD * 00011000
* * 00011100
* * 00011200

```

```

* -IF "LIST SPECIFIC: 5000-LIST-SPECIFIC* 00011300
* FETCH FIRST RECORD * 00011400
* CREATE ISPF TABLE * 00011500
* DO UNTIL NO MORE MATCHING RECORDS: * 00011600
* STORE RECORD IN TABLE 5500-LIST-AND-GET * 00011700
* GET ANOTHER RECORD * 00011800
* DISPLAY PHONE LIST ON SCREEN 6000-DISPLAY-LIST * 00011900
* IF UPDATE REQUESTED 6500-UPDATE-LOOP * 00012000
* UPDATE PHONE RECORDS 7000-UPDATE * 00012100
----- 00012200
* ENVIRONMENT DIVISION. 00012300
* DATA DIVISION. 00012400
* WORKING-STORAGE SECTION. 00012500
----- 00012600
77 COIBM PIC X(54) VALUE IS 00012700
'COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1987'. 00012800
77 SEL-EXIT PIC X(01). 00012900
77 DIS-EXIT PIC X(01). 00013000
77 DISPLAY-TABLE PIC X(01). 00013100
77 MORE-CHANGES PIC X(01). 00013200
77 ROWS-CHANGED PIC 9(04). 00013300
77 PERCENT-COUNTER PIC S9(4) COMP. 00013400
77 MODULE PIC X(07) VALUE 'DSN8SC3'. 00013500
77 MSGCODE PIC X(04). 00013600
77 W-BLANK PIC X(01) VALUE ' '. 00013700
77 MSGS-VAR PIC X(08) VALUE 'DSN8MSGS'. 00013800
77 FI-VAR PIC X(08) VALUE 'FNAMEI '. 00013900
77 LI-VAR PIC X(08) VALUE 'LNAMEI '. 00014000
----- 00014100
* ISPF DIALOG VARIABLE NAMES * 00014200
----- 00014300
* EXEC SQL INCLUDE SQLCA END-EXEC. 00014400
01 LNAMEW PIC X(15). 00014500
01 FNAMEW PIC X(12). 00014600
01 LIST-PANEL-VARIABLES. 00014700
03 CH-VAR PIC X(08) VALUE 'ZTDSELS '. 00014800
03 FN-VAR PIC X(08) VALUE 'FNAMED '. 00014900
03 MI-VAR PIC X(08) VALUE 'MINITD '. 00015000
03 LN-VAR PIC X(08) VALUE 'LNAMED '. 00015100
03 PN-VAR PIC X(08) VALUE 'PNOD '. 00015200
03 EN-VAR PIC X(08) VALUE 'ENOD '. 00015300
03 WD-VAR PIC X(08) VALUE 'WDEPTD '. 00015400
03 WN-VAR PIC X(08) VALUE 'WNAMED '. 00015500
03 TABLE-NAME PIC X(08) VALUE 'DSN8TABL'. 00015600
03 SEL-VARS PIC X(20) VALUE IS 00015700
' (FNAMEI LNAMEI) '. 00015800
03 DIS-VARS PIC X(56) VALUE IS 00015900
' (ZTDSELS FNAMED MINITD LNAMED PNOD ENOD WDEPTD WNAMED) '. 00016000
03 EMP-VARS PIC X(48) VALUE IS 00016100
' (FNAMED MINITD LNAMED PNOD ENOD WDEPTD WNAMED) '. 00016200
01 PANEL-VARIABLE-LENGTHS. 00016300
03 CH-VAR-STG PIC 9(06) COMP VALUE 04. 00016400
03 FN-VAR-STG PIC 9(06) COMP VALUE 12. 00016500
03 MI-VAR-STG PIC 9(06) COMP VALUE 01. 00016600
03 LN-VAR-STG PIC 9(06) COMP VALUE 15. 00016700
03 PN-VAR-STG PIC 9(06) COMP VALUE 04. 00016800
03 EN-VAR-STG PIC 9(06) COMP VALUE 06. 00016900
03 WD-VAR-STG PIC 9(06) COMP VALUE 03. 00017000
03 WN-VAR-STG PIC 9(06) COMP VALUE 36. 00017100
03 FI-VAR-STG PIC 9(06) COMP VALUE 12. 00017200
03 LI-VAR-STG PIC 9(06) COMP VALUE 15. 00017300
03 MSGS-VAR-STG PIC 9(06) COMP VALUE 79. 00017400
----- 00017500
* ISPF DIALOG SERVICES DECLARATIONS * 00017600
----- 00017700
01 I-VDEFINE PIC X(08) VALUE 'VDEFINE '. 00017800
01 I-VGET PIC X(08) VALUE 'VGET '. 00017900
01 I-VPUT PIC X(08) VALUE 'VPUT '. 00018000
01 I-DISPLAY PIC X(08) VALUE 'DISPLAY '. 00018100
01 I-TBDISPL PIC X(08) VALUE 'TBDISPL '. 00018200
01 I-TBTOP PIC X(08) VALUE 'TBTOP '. 00018300
01 I-TBCREATE PIC X(08) VALUE 'TBCREATE '. 00018400
01 I-TBCLOSE PIC X(08) VALUE 'TBCLOSE '. 00018500
01 I-TBADD PIC X(08) VALUE 'TBADD '. 00018600
01 I-TBPUT PIC X(08) VALUE 'TBPUT '. 00018700
----- 00018800
* ISPF CALL MODIFIERS * 00018900
----- 00019000
01 I-NOWRITE PIC X(08) VALUE 'NOWRITE '. 00019100
01 I-REPLACE PIC X(08) VALUE 'REPLACE '. 00019200
01 I-CHAR PIC X(08) VALUE 'CHAR '. 00019300
----- 00019400

```

```

* ISPF PANEL NAMES * 00019500
----- * 00019600
 01 SEL-PANEL PIC X(08) VALUE 'DSN8SSL '. 00019700
 01 DIS-PANEL PIC X(08) VALUE 'DSN8SSN '. 00019800
----- * 00019900
* LOCAL-VARIABLES * 00020000
----- * 00020100
 01 LOCAL-VARIABLES. 00020200
 03 LNAMEI PIC X(15) VALUE SPACES. 00020300
 03 FNAMEI PIC X(12) VALUE SPACES. 00020400
 03 CONVSQL PIC S9(15) COMP-3. 00020500
 03 OUTMSG PIC X(69). 00020600
 03 TMSG REDEFINES OUTMSG. 00020700
 05 TMSGXTT PIC X(46). 00020800
 05 FILLER PIC X(23). 00020900
 03 MSGS PIC X(79) VALUE SPACES. 00021000
 03 MSGS-DETAIL REDEFINES MSGS. 00021100
 05 OUT-MESSAGE PIC X(46). 00021200
 05 SQL-CODE PIC +(04). 00021300
 05 FILLER PIC X(29). 00021400
----- * 00021500
* EMPLOYEE RECORD - IO AREA * 00021600
----- * 00021700
 01 EMP-RECORD. 00021800
 02 EMPLAST PIC X(15). 00021900
 02 EMP-FIRST-NAME PIC X(12). 00022000
 02 EMP-MIDDLE-INITIAL PIC X(01). 00022100
 02 EMPPHONE PIC X(04). 00022200
 02 EMPNUMB PIC X(06). 00022300
 02 EMP-DEPT-NUMBER PIC X(03). 00022400
 02 EMP-DEPTNAME PIC X(36). 00022500
----- * 00022600
* SQL DECLARATION FOR VIEW PHONE * 00022700
----- * 00022800
 EXEC SQL DECLARE VPHONE TABLE 00022900
 (LASTNAME VARCHAR(15) , 00023000
 FIRSTNAME VARCHAR(12) , 00023100
 MIDDLEINITIAL CHAR(1) , 00023200
 PHONENUMBER CHAR(4) , 00023300
 EMPLOYEENUMBER CHAR(6) , 00023400
 DEPTNUMBER CHAR(3) NOT NULL, 00023500
 DEPTNAME VARCHAR(36) NOT NULL) END-EXEC. 00023600
----- * 00023700
* STRUCTURE FOR PHONE RECORD * 00023800
----- * 00023900
 01 PPHONE. 00024000
 02 LAST-NAME PIC X(15). 00024100
 02 FIRST-NAME PIC X(12). 00024200
 02 MIDDLE-INITIAL PIC X(01). 00024300
 02 PHONE-NUMBER PIC X(04). 00024400
 02 EMPLOYEE-NUMBER PIC X(06). 00024500
 02 DEPT-NUMBER PIC X(03). 00024600
 02 DEPTNAME PIC X(36). 00024700
----- * 00024800
* SQL DECLARATION FOR VIEW VEMPLP * 00024900
----- * 00025000
 EXEC SQL DECLARE VEMPLP TABLE 00025100
 (EMPLOYEENUMBER CHAR(6) 00025200
 PHONENUMBER CHAR(4)) END-EXEC. 00025300
----- * 00025400
* SQL CURSORS * 00025500
----- * 00025600
 EXEC SQL DECLARE TELE1 CURSOR FOR 00025700
 SELECT * 00025800
 FROM VPHONE 00025900
 END-EXEC. 00026000
----- * 00026100
 EXEC SQL DECLARE TELE2 CURSOR FOR 00026200
 SELECT * 00026300
 FROM VPHONE 00026400
 WHERE LASTNAME LIKE :LNAMEW 00026500
 AND FIRSTNAME LIKE :FNAMEW 00026600
 END-EXEC. 00026700
----- * 00026800
 EXEC SQL DECLARE TELE3 CURSOR FOR 00026900
 SELECT * 00027000
 FROM VPHONE 00027100
 WHERE LASTNAME = :LNAMEW 00027200
 AND FIRSTNAME LIKE :FNAMEW 00027300
 END-EXEC. 00027400
----- * 00027500
 EJECT 00027600

```

```

PROCEDURE DIVISION. 00027700
----- 00027800
* SQL RETURN CODE HANDLING * 00027900
----- 00028000
 EXEC SQL WHENEVER SQLERROR GOTO L8000-P3-DBERROR END-EXEC. 00028100
 EXEC SQL WHENEVER SQLWARNING GOTO L8000-P3-DBERROR END-EXEC. 00028200
 EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC. 00028300
*
----- 00028400
----- 00028500
* DEFINE COBOL - SPF VARIABLES * 00028600
----- 00028700
 0000-PROGRAM-START. 00028800
 CALL 'ISPLINK' USING I-VDEFINE, CH-VAR, ROWS-CHANGED, 00028900
 I-CHAR, CH-VAR-STG. 00029000
 CALL 'ISPLINK' USING I-VDEFINE, FN-VAR, EMP-FIRST-NAME, 00029100
 I-CHAR, FN-VAR-STG. 00029200
 CALL 'ISPLINK' USING I-VDEFINE, MI-VAR, EMP-MIDDLE-INITIAL, 00029300
 I-CHAR, MI-VAR-STG. 00029400
 CALL 'ISPLINK' USING I-VDEFINE, LN-VAR, EMPLAST, 00029500
 I-CHAR, LN-VAR-STG. 00029600
 CALL 'ISPLINK' USING I-VDEFINE, PN-VAR, EMPPHONE, 00029700
 I-CHAR, PN-VAR-STG. 00029800
 CALL 'ISPLINK' USING I-VDEFINE, EN-VAR, EMPNUMB, 00029900
 I-CHAR, EN-VAR-STG. 00030000
 CALL 'ISPLINK' USING I-VDEFINE, WD-VAR, EMP-DEPT-NUMBER, 00030100
 I-CHAR, WD-VAR-STG. 00030200
 CALL 'ISPLINK' USING I-VDEFINE, WN-VAR, EMP-DEPTNAME, 00030300
 I-CHAR, WN-VAR-STG. 00030400
 CALL 'ISPLINK' USING I-VDEFINE, FI-VAR, FNAMEI, 00030500
 I-CHAR, FI-VAR-STG. 00030600
 CALL 'ISPLINK' USING I-VDEFINE, LI-VAR, LNAMEI, 00030700
 I-CHAR, LI-VAR-STG. 00030800
 CALL 'ISPLINK' USING I-VDEFINE, MSGS-VAR, MSGS, 00030900
 I-CHAR, MSGS-VAR-STG. 00031000
*
----- 00031100
----- 00031200
* MAIN PROGRAM * 00031300
----- 00031400
 MOVE 'N' TO SEL-EXIT. 00031500
 PERFORM 1000-MAIN-LOOP THRU 1000-MAIN-LOOP-EXIT 00031600
 UNTIL SEL-EXIT = 'Y'.
 MOVE 0 TO RETURN-CODE. 00031800
 GOBACK. 00031900
*
----- 00032000
1000-MAIN-LOOP. 00032100
 CALL 'ISPLINK' USING I-DISPLAY, SEL-PANEL. 00032200
 MOVE SPACES TO MSGS. 00032300
 MOVE SPACES TO OUTMSG. 00032400
 IF RETURN-CODE = 8 00032500
 MOVE 'Y' TO SEL-EXIT. 00032600
 ELSE
 MOVE 'N' TO DISPLAY-TABLE. 00032800
 CALL 'ISPLINK' USING I-VGET, SEL-VARS. 00032900
 MOVE LNAMEI TO LNAMEW. 00033000
 MOVE FNAMEI TO FNAMEW. 00033100
 PERFORM 2000-GET-TYPE THRU 2000-GET-TYPE-EXIT 00033200
 IF DISPLAY-TABLE = 'Y'
 PERFORM 6000-DISPLAY-LIST. 00033300
 THRU 6000-DISPLAY-LIST-EXIT. 00033400
 00033500
 CALL 'ISPLINK' USING I-VPUT MSGS-VAR. 00033600
 1000-MAIN-LOOP-EXIT. 00033700
 EXIT. 00033800
*
----- 00033900
----- 00034000
* DETERMINE PROCESSING REQUEST * 00034100
----- 00034200
----- 00034300
2000-GET-TYPE. 00034400
 IF LNAMEW = '*' 00034500
 PERFORM 3000-LIST-ALL. 00034600
 THRU 3000-LIST-ALL-EXIT. 00034700
 ELSE
 UNSTRING LNAMEW
 DELIMITED BY SPACE
 INTO LNAMEW. 00034800
 UNSTRING FNAMEW
 DELIMITED BY SPACE
 INTO FNAMEW. 00034900
 INSPECT FNAMEW
 REPLACING ALL ' ' BY '%'. 00035000
 00035100
 MOVE 0 TO PERCENT-COUNTER. 00035200
 INSPECT LNAMEW
 TALLYING PERCENT-COUNTER FOR ALL '%' 00035300
 00035400
 00035500
 00035600
 00035700
 00035800

```

```

 IF PERCENT-COUNTER > 0 00035900
 INSPECT LNAMEW 00036000
 REPLACING ALL ' ' BY '%' 00036100
 PERFORM 4000-LIST-GENERIC 00036200
 THRU 4000-LIST-GENERIC-EXIT. 00036300
 ELSE
 PERFORM 5000-LIST-SPECIFIC 00036400
 THRU 5000-LIST-SPECIFIC-EXIT. 00036500
 2000-GET-TYPE-EXIT.
 EXIT.
*
----- 00036900
----- LIST ALL EMPLOYEES * 00037100
----- 00037200
3000-LIST-ALL.
 EXEC SQL OPEN TELE1 END-EXEC. 00037300
 MOVE SPACES TO SQLERRP. 00037400
 EXEC SQL FETCH TELE1 INTO :PPHONE END-EXEC. 00037500
 IF SQLERRP = SPACES 00037600
 MOVE '079E' TO MSGCODE 00037700
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00037800
 MOVE OUTMSG TO MSGS 00037900
 ELSE
 IF SQLCODE = 100 00038000
 MOVE '008I' TO MSGCODE 00038100
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00038200
 MOVE OUTMSG TO MSGS 00038300
 ELSE
 MOVE 'Y' TO DISPLAY-TABLE 00038400
 CALL 'ISPLINK' USING I-TBCREATE, TABLE-NAME,
 W-BLANK, EMP-VARS, I-NOWRITE, I-REPLACE 00038500
 PERFORM 3500-LIST-AND-GET 00038600
 THRU 3500-LIST-AND-GET-EXIT. 00038700
 UNTIL SQLCODE NOT EQUAL 0. 00038800
 EXEC SQL CLOSE TELE1 END-EXEC. 00038900
 3000-LIST-ALL-EXIT.
 EXIT.
*
----- 00039000
----- 3500-LIST-AND-GET. * 00039100
----- 00039200
 MOVE PPHONE TO EMP-RECORD. 00039300
 CALL 'ISPLINK' USING I-TBADD, TABLE-NAME. 00039400
 EXEC SQL FETCH TELE1 INTO :PPHONE END-EXEC. 00039500
 3500-LIST-AND-GET-EXIT.
 EXIT.
*
----- 00039600
----- 4000-LIST-GENERIC. * 00039700
----- 00039800
 MOVE SPACES TO SQLERRP. 00039900
 EXEC SQL FETCH TELE2 INTO :PPHONE END-EXEC. 00040000
 IF SQLERRP = SPACES 00040100
 MOVE '079E' TO MSGCODE 00040200
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00040300
 MOVE OUTMSG TO MSGS 00040400
 ELSE
 IF SQLCODE = 100 00040500
 MOVE '008I' TO MSGCODE 00040600
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00040700
 MOVE OUTMSG TO MSGS 00040800
 ELSE
 MOVE 'Y' TO DISPLAY-TABLE 00040900
 CALL 'ISPLINK' USING I-TBCREATE, TABLE-NAME,
 W-BLANK, EMP-VARS, I-NOWRITE, I-REPLACE 00041000
 PERFORM 4500-LIST-AND-GET 00041100
 THRU 4500-LIST-AND-GET-EXIT. 00041200
 UNTIL SQLCODE NOT EQUAL 0. 00041300
 EXEC SQL CLOSE TELE2 END-EXEC. 00041400
 4000-LIST-GENERIC-EXIT.
 EXIT.
*
----- 00041500
----- 4500-LIST-AND-GET. * 00041600
----- 00041700
 MOVE PPHONE TO EMP-RECORD. 00041800
 CALL 'ISPLINK' USING I-TBADD, TABLE-NAME. 00041900
 EXEC SQL FETCH TELE2 INTO :PPHONE END-EXEC. 00042000
 4500-LIST-AND-GET-EXIT.
 EXIT.
*
----- 00042100
----- SPECIFIC LIST OF EMPLOYEES * 00042200
----- 00042300
5000-LIST-SPECIFIC.

```

```

EXEC SQL OPEN TELE3 END-EXEC. 00044100
MOVE SPACES TO SQLERRP. 00044200
EXEC SQL FETCH TELE3 INTO :PPHONE END-EXEC. 00044300
IF SQLERRP = SPACES 00044400
 MOVE '079E' TO MSGCODE 00044500
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00044600
 MOVE OUTMSG TO MSGS 00044700
ELSE 00044800
 IF SQLCODE = 100 00044900
 MOVE '0081' TO MSGCODE 00045000
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00045100
 MOVE OUTMSG TO MSGS 00045200
 ELSE 00045300
 MOVE 'Y' TO DISPLAY-TABLE 00045400
 CALL 'ISPLINK' USING I-TBCREATE, TABLE-NAME,
 W-BLANK, EMP-VARS, I-NOWRITE, I-REPLACE 00045500
 PERFORM 5500-LIST-AND-GET 00045600
 THRU 5500-LIST-AND-GET-EXIT 00045700
 UNTIL SQLCODE NOT EQUAL 0. 00045800
 EXEC SQL CLOSE TELE3 END-EXEC. 00045900
 5000-LIST-SPECIFIC-EXIT. 00046000
 EXIT. 00046100
* 00046200
* 00046300
* 5500-LIST-AND-GET. 00046400
 MOVE PPHONE TO EMP-RECORD. 00046500
 CALL 'ISPLINK' USING I-TBADD, TABLE-NAME. 00046600
 EXEC SQL FETCH TELE3 INTO :PPHONE END-EXEC. 00046700
 5500-LIST-AND-GET-EXIT. 00046800
 EXIT. 00046900
* 00047000
----- 00047100
* DISPLAY EMPLOYEE PHONE NUMBERS * 00047200
----- 00047300
----- 00047400
6000-DISPLAY-LIST. 00047500
 EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC. 00047600
 EXEC SQL WHENEVER SQLWARNING CONTINUE END-EXEC. 00047700
 CALL 'ISPLINK' USING I-TBTOP, TABLE-NAME. 00047800
 CALL 'ISPLINK' USING I-TBDISPL, TABLE-NAME, DIS-PANEL. 00047900
 IF RETURN-CODE NOT EQUAL 8 00048000
 CALL 'ISPLINK' USING I-VGET, DIS-VARS 00048100
 PERFORM 6500-UPDATE-LOOP THRU 6500-UPDATE-LOOP-EXIT. 00048200
 6000-DISPLAY-LIST-EXIT. 00048300
 EXIT. 00048400
----- 00048500
* DETERMINE IF UPDATE HAS BEEN REQUESTED * 00048600
----- 00048700
----- 00048800
6500-UPDATE-LOOP. 00048900
 IF ROWS-CHANGED > 0 00049000
 MOVE 'Y' TO MORE-CHANGES 00049100
 PERFORM 7000-UPDATE THRU 7000-UPDATE-EXIT 00049200
 UNTIL MORE-CHANGES = 'N'. 00049300
 CALL 'ISPLINK' USING I-TBCLOSE, TABLE-NAME. 00049400
 6500-UPDATE-LOOP-EXIT. 00049500
 EXIT. 00049600
----- 00049700
* UPDATE EMPLOYEE PHONE NUMBERS * 00049800
----- 00049900
----- 00050000
7000-UPDATE. 00050100
 EXEC SQL UPDATE VEMPLP 00050200
 SET PHONENUMBER = :EMPPHONE 00050300
 WHERE EMPLOYEENUMBER = :EMPNUMB END-EXEC. 00050400
 IF SQLCODE NOT EQUAL 0 00050500
 MOVE '060E' TO MSGCODE 00050600
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00050700
 MOVE OUTMSG TO TMSG 00050800
 MOVE TMSGTXT TO OUT-MESSAGE 00050900
 MOVE SQLCODE TO CONVSQL 00051000
 MOVE CONVSQL TO SQL-CODE 00051100
 EXEC SQL ROLLBACK END-EXEC. 00051200
 MOVE 'N' TO MORE-CHANGES 00051300
 ELSE 00051400
 MOVE '004I' TO MSGCODE 00051500
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG 00051600
 MOVE OUTMSG TO MSGS 00051700
 CALL 'ISPLINK' USING I-TBPUT, TABLE-NAME 00051800
 IF ROWS-CHANGED > 1 00051900
 CALL 'ISPLINK' USING I-TBDISPL, TABLE-NAME 00052000
 CALL 'ISPLINK' USING I-VGET, DIS-VARS 00052100
 ELSE MOVE 'N' TO MORE-CHANGES. 00052200
 7000-UPDATE-EXIT. 00052300

```

```

 EXIT. 00052300
 * 00052400
 ----- 00052500
 * DB2 ERROR PROCESSING * 00052600
 ----- 00052700
 L8000-P3-DBERROR. 00052800
 MOVE '060E' TO MSGCODE. 00052900
 CALL 'DSN8MCG' USING MODULE, MSGCODE, OUTMSG. 00053000
 MOVE OUTMSG TO TMSG. 00053100
 MOVE TMSGTXT TO OUT-MESSAGE. 00053200
 MOVE SQLCODE TO CONVSQL. 00053300
 MOVE CONVSQL TO SQL-CODE. 00053400
 CALL 'ISPLINK' USING I-VPUT, MSGS-VAR. 00053500
 GOBACK. 00053600

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8SP3

ESTE MÓDULO LISTA LOS NÚMEROS DE TELÉFONO DE EMPLEADOS Y LOS ACTUALIZA SI LO DESEA.

```

DSN8SP3: PROC OPTIONS (MAIN);
/***/
*
* MODULE NAME = DSN8SP3
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
* PHONE APPLICATION
* ISPF
* PL/I
*
* COPYRIGHT = 5665-DB2 (C) COPYRIGHT IBM CORP 1982, 1991
* SEE COPYRIGHT INSTRUCTIONS
* LICENSED MATERIALS - PROPERTY OF IBM
*
* STATUS = VERSION 2 RELEASE 3, LEVEL 0
*
* FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND
* UPDATES THEM IF DESIRED.
*
* NOTES =
* DEPENDENCIES = TWO ISPF PANELS ARE REQUIRED:
* DSN8SSL AND DSN8SSN
* RESTRICTIONS = NONE
*
* MODULE TYPE = PL/I PROC OPTIONS(MAIN)
* PROCESSOR = DB2 PRECOMPLIER, PL/I OPTIMIZER
* MODULE SIZE = SEE LINKEDIT
* ATTRIBUTES = REENTRANT
*
* ENTRY POINT = DSN8SP3
* PURPOSE = SEE FUNCTION
* LINKAGE = INVOKED FROM ISPF
*
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
* INPUT-MESSAGE:
*
* SYMBOLIC LABEL/NAME = DSN8SSL
* DESCRIPTION = PHONE MENU 1 (SELECT)
*
* SYMBOLIC LABEL/NAME = DSN8SSN
* DESCRIPTION = PHONE MENU 2 (LIST)
*
* SYMBOLIC LABEL/NAME = VPHONE
* DESCRIPTION = VIEW OF TELEPHONE INFORMATION
*
* SYMBOLIC LABEL/NAME = VEMPLP
* DESCRIPTION = VIEW OF EMPLOYEE INFORMATION
*
* OUTPUT = PARAMETERS EXPLICITLY RETURNED:
* OUTPUT-MESSAGE:
*
* SYMBOLIC LABEL/NAME = DSN8SSL
* DESCRIPTION = PHONE MENU 1 (SELECT)
*
* SYMBOLIC LABEL/NAME = DSN8SSN

```

```

* DESCRIPTION = PHONE MENU 2 (LIST) *
*
* EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION *
*
* EXIT-ERROR = *
*
* RETURN CODE = NONE *
*
* ABEND CODES = NONE *
*
*
* ERROR-MESSAGES = *
* DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED *
* DSN8008I - NO EMPLOYEE FOUND IN TABLE *
* DSN8060E - SQL ERROR, RETURN CODE IS: *
* DSN8079E - CONNECTION TO DB2 NOT ESTABLISHED *
*
* EXTERNAL REFERENCES = *
* ROUTINES/SERVICES = *
* DSN8MPG - ERROR MESSAGE ROUTINE *
* ISPLINK - ISPF SERVICES ROUTINE *
*
* DATA-AREAS = *
* NONE *
*
* CONTROL-BLOCKS = *
* SQLCA - SQL COMMUNICATION AREA *
*
* TABLES = NONE *
*
*
* CHANGE-ACTIVITY: *
*
* CHECK SQLERRP FOR NON-BLANKS TO ENSURE CONNECTION V2R3 *
* HAS BEEN ESTABLISHED. ISSUE 079E IF NOT. *
*
* *PSEUDOCODE* *
*
* PROCEDURE *
* DO WHILE NOT EXIT-PRESSED *
* CALL GET-TYPE *
* CALL GET-LIST *
* CALL DISPLAY-LIST *
*
* GET_TYPE: *
* IF LASTNAME IS '*' *
* TYPE = 'ALL' *
* ELSE *
* IF LASTNAME CONTAINS '%' *
* TYPE = 'GENERIC' *
* ELSE *
* TYPE = 'SPECIFIC' *
*
* GET_LIST: *
* CASE (TYPE) *
* SUBCASE ('ALL') *
* GET ALL EMPLOYEES *
* SUBCASE ('GENERIC') *
* GET GENERIC EMPLOYEES *
* SUBCASE ('SPECIFIC') *
* GET SPECIFIC EMPLOYEES *
* ENDSUB *
*
* DISPLAY_LIST: *
* DISPLAY LIST *
* IF NOT EXIT-PRESSED *
* UPDATE PHONE NUMBER(S) *
* WRITE CONFIRMATION MESSAGE *
*
* P3_DBERROR: *
* IF SQL ERROR OCCURS THEN *
* FORMAT ERROR MESSAGE *
* ROLLBACK *
* END *
* END. *
*****/* *****/
/* DECLARATION FOR BUILTIN FUNCTIONS */
*****/*
DCL ADDR BUILTIN;

```

```

DCL INDEX BUILTIN;
DCL PLIRETC BUILTIN;
DCL PLIRETV BUILTIN;
DCL STG BUILTIN;
DCL SUBSTR BUILTIN;
DCL TRANSLATE BUILTIN;

/***
/* MESSAGE ROUTINE DECLARATIONS */
 /**

DCL DSN8MPG EXTERNAL ENTRY;

DCL MODULE CHAR (7) INIT('DSN8SP3'); /* EXECUTING PROGRAM */
DCL OUTMSG CHAR (69); /* MESSAGE TEXT */

1/***
/* ISPF DIALOG VARIABLE NAMES */
 /**

/* SELECTION AND LIST PANEL VARIABLES */
DCL MSGS_VAR CHAR(08) STATIC INIT('DSN8MSGS'); /*PANEL MSG FIELD*/

/* SELECTION PANEL VARIABLES */
DCL FI_VAR CHAR(08) STATIC INIT('FNAMEI '); /*FIRST NAME VAR */
DCL LI_VAR CHAR(08) STATIC INIT('LNAMEI '); /*LAST NAME VAR */

/* LIST PANEL VARIABLES */
DCL CH_VAR CHAR(08) STATIC INIT('ZTDSELS '); /*# ROWS CHANGED */
DCL FN_VAR CHAR(08) STATIC INIT('FNAMED '); /*FIRST NAME VAR */
DCL MI_VAR CHAR(08) STATIC INIT('MINITD '); /*MID INIT VAR */
DCL LN_VAR CHAR(08) STATIC INIT('LNAMED '); /*LAST NAME VAR */
DCL PN_VAR CHAR(08) STATIC INIT('PNOD '); /*PHONE NUM VAR */
DCL EN_VAR CHAR(08) STATIC INIT('ENOD '); /*EMPL NUM VAR */
DCL WD_VAR CHAR(08) STATIC INIT('WDEPTD '); /*WORK DEPT VAR */
DCL WN_VAR CHAR(08) STATIC INIT('WNAMED '); /*DEPT NAME VAR */
DCL TABLE_NAME CHAR(08) STATIC INIT('DSN8TABL'); /*TABLE NAME VAR */
DCL SEL_VARS CHAR(20) STATIC /*SELECTION VARS */
INIT('FNAMEI LNAMEI ')';

DCL DIS_VARS CHAR(56) STATIC /*DISPLAY VARS */
INIT('ZTDSELS FNAMED MINITD LNAMED PNOD ENOD WDEPTD WNAMED ');
DCL EMP_VARS CHAR(48) STATIC /*DISPLAY VARS */
INIT('FNAMEI MINITD LNAMED PNOD ENOD WDEPTD WNAMED ');

/***
/* ISPF DIALOG SERVICES DECLARATIONS */
 /**

/* PROGRAM NAME */
DCL ISPLINK EXTERNAL ENTRY OPTIONS(ASM INTER RETCODE);

/* ISPF DIALOG SERVICE TYPES */
DCL I_VDEFINE CHAR(8) STATIC INIT('VDEFINE ');
DCL I_VGET CHAR(8) STATIC INIT('VGET ');
DCL I_VPUT CHAR(8) STATIC INIT('VPUT ');
DCL I_DISPLAY CHAR(8) STATIC INIT('DISPLAY ');
DCL I_TBDISPL CHAR(8) STATIC INIT('TBDISPL ');
DCL I_TBTOP CHAR(8) STATIC INIT('TBTOP ');
DCL I_TBCREATE CHAR(8) STATIC INIT('TBCREATE');
DCL I_TBCLOSE CHAR(8) STATIC INIT('TBCLOSE ');
DCL I_TBADD CHAR(8) STATIC INIT('TBADD ');
DCL I_TBPUT CHAR(8) STATIC INIT('TBPUT ');

/* ISPF CALL MODIFIERS */
DCL I_NOWRITE CHAR(8) STATIC INIT('NOWRITE');
DCL I_REPLACE CHAR(8) STATIC INIT('REPLACE');
DCL I_CHAR CHAR(8) STATIC INIT('CHAR');

/* PANEL NAMES */
DCL SEL_PANEL CHAR(8) STATIC INIT('DSN8SSL'); /* SELECTION PANEL */
DCL DIS_PANEL CHAR(8) STATIC INIT('DSN8SSN'); /* LIST PANEL */

/* LOCAL VARIABLES FOR ISPF VARIABLES */
DCL LNAMEI CHAR(15) INIT(' '); /* LAST-NAME INPUT */
DCL FNAMEI CHAR(12) INIT(' '); /* FIRST-NAME INPUT */

DCL MSGS CHAR(79) INIT(' '); /* MESSAGE FOR ISPF PANEL */

DCL 1 EMP_RECORD, /* PANEL DISPLAY INFORMATION */
2 LASTNAME CHAR (15),
2 FIRSTNAME CHAR (12),
2 MIDDLEINITIAL CHAR (1),
2 PHONENUMBER CHAR (4),

```

```

2 EMPLOYEENUMBER CHAR (6),
2 DEPTNUMBER CHAR (3),
2 DEPTNAME CHAR (36);

1/***
/* DECLARATION FOR PROGRAM LOGIC */
 /***/
/* CONSTANTS */*
DCL YES BIT(1) STATIC INIT('1'B);
DCL NO BIT(1) STATIC INIT('0'B);
DCL ZERO FIXED BIN(31,0) STATIC INIT(0);

/* FLAGS */*
DCL SEL_EXIT BIT(1); /* EXIT PRESSED? FLAG */
DCL DIS_EXIT BIT(1); /* EXIT PRESSED? FLAG */
DCL DIS_TABLE BIT(1); /* DISPLAY-TABLE? FLAG */
DCL MORE_CHANGES BIT(1); /* MORE CHANGES TO PROCESS? */

/* DATA VARIABLES */*
DCL ROWS_CHANGED PIC'9999';
DCL TYPE CHAR(8); /* TYPE OF LIST */
DCL L NAMES CHAR(15); /* LAST NAME SELECTION VALUE */
DCL F NAMES CHAR(12); /* FIRST NAME SELECTION VALUE */

1/***
/* SQL DECLARATIONS */
 /***/
/*
 SQL COMMUNICATION AREA */
0EXEC SQL INCLUDE SQLCA;
DCL SQL_PIC PIC'-999';

/*
 SQL DECLARATION FOR VIEW PHONE */
EXEC SQL DECLARE VPHONE TABLE
 (LASTNAME VARCHAR(15),
 FIRSTNAME VARCHAR(12),
 MIDDLEINITIAL CHAR(1),
 PHONENUMBER CHAR(4),
 EMPLOYEENUMBER CHAR(6),
 DEPTNUMBER CHAR(3) NOT NULL,
 DEPTNAME VARCHAR(36) NOT NULL);
 /* STUTURE FOR PHONE RECORD */

DCL 1 PPHONE,
 2 LASTNAME CHAR (15) VAR,
 2 FIRSTNAME CHAR (12) VAR,
 2 MIDDLEINITIAL CHAR (1),
 2 PHONENUMBER CHAR (4),
 2 EMPLOYEENUMBER CHAR (6),
 2 DEPTNUMBER CHAR (3),
 2 DEPTNAME CHAR (36) VAR;
 /* SQL DECLARATION FOR VIEW VEMPLP*/

EXEC SQL DECLARE VEMPLP TABLE
 (EMPLOYEENUMBER CHAR(6),
 PHONENUMBER CHAR(4));

1/***
/* CURSOR DECLARATIONS */
 /***/
EXEC SQL DECLARE TELE1 CURSOR FOR
 SELECT *
 FROM VPHONE;

EXEC SQL DECLARE TELE2 CURSOR FOR
 SELECT *
 FROM VPHONE
 WHERE LASTNAME LIKE :LNAMES
 AND FIRSTNAME LIKE :FNAMES;

EXEC SQL DECLARE TELE3 CURSOR FOR
 SELECT *
 FROM VPHONE
 WHERE LASTNAME = :LNAMES
 AND FIRSTNAME LIKE :FNAMES;

1/***
/* SQL RETURN CODE HANDLING */
 /***/
EXEC SQL WHENEVER SQLERROR GOTO P3_DBERROR;
EXEC SQL WHENEVER SQLWARNING GOTO P3_DBERROR;

```

```

EXEC SQL WHENEVER NOT FOUND CONTINUE;

/*****PL/I - ISPF VARIABLES *****/
CALL ISPLINK(I_VDEFINE, CH_VAR, ROWS_CHANGED,
 I_CHAR, STG(ROWS_CHANGED));
CALL ISPLINK(I_VDEFINE, FN_VAR, EMP_RECORD.FIRSTNAME,
 I_CHAR, STG(EMP_RECORD.FIRSTNAME));
CALL ISPLINK(I_VDEFINE, MI_VAR, EMP_RECORD.MIDDLEINITIAL,
 I_CHAR, STG(EMP_RECORD.MIDDLEINITIAL));
CALL ISPLINK(I_VDEFINE, LN_VAR, EMP_RECORD.LASTNAME,
 I_CHAR, STG(EMP_RECORD.LASTNAME));
CALL ISPLINK(I_VDEFINE, PN_VAR, EMP_RECORD.PHONENUMBER,
 I_CHAR, STG(EMP_RECORD.PHONENUMBER));
CALL ISPLINK(I_VDEFINE, EN_VAR, EMP_RECORD.EMPLOYEENUMBER,
 I_CHAR, STG(EMP_RECORD.EMPLOYEENUMBER));
CALL ISPLINK(I_VDEFINE, WD_VAR, EMP_RECORD.DEPTNUMBER,
 I_CHAR, STG(EMP_RECORD.DEPTNUMBER));
CALL ISPLINK(I_VDEFINE, WN_VAR, EMP_RECORD.DEPTNAME,
 I_CHAR, STG(EMP_RECORD.DEPTNAME));
CALL ISPLINK(I_VDEFINE, FI_VAR, FNAMEI,
 I_CHAR, STG(FNAMEI));
CALL ISPLINK(I_VDEFINE, LI_VAR, LNAMEI,
 I_CHAR, STG(LNAMEI));
CALL ISPLINK(I_VDEFINE, MSGS_VAR, MSGS,
 I_CHAR, STG(MSGS));

1/*****
/* MAIN PROGRAM
*****/
SEL_EXIT = '0'B; /* INITIALIZE EXIT BIT */
DO WHILE (^SEL_EXIT); /* DO WHILE NOT EXIT */
 CALL ISPLINK(I_DISPLAY, SEL_PANEL); /* */
 MSGS = ' '; /* RESET THE MSG FIELD */
 OUTMSG = ' '; /* RESET THE MSG FIELD */
 SEL_EXIT = (PLIRETV = 8); /* SEL_EXIT = TRUE IF RC=8 */
/*****
/* EXIT WAS NOT SPECIFIED SO PROCESS THE REQUEST
*/
*****/

IF ^SEL_EXIT THEN /* IF USER PRESSED ENTER */
 DO;
 DIS_TABLE = NO; /* INIT FLAG TO NO */
 CALL ISPLINK(I_VGET, SEL_VARS); /* */
 L NAMES = LNAMEI; /* COPY INPUT TO WORKING VAR */
 F NAMES = FNAMEI; /* COPY INPUT TO WORKING VAR */
 CALL GET_TYPE; /* DETERMINE LIST TYPE */
 CALL GET_LIST; /* GET LIST OF EMPLOYEES */
 IF DIS_TABLE THEN
 CALL DISPLAY_LIST;
 END; /* END IF USER PRESSED ENTER */
 CALL ISPLINK(I_VPUT, MSGS_VAR); /* SET PANEL MESSAGE */
 END; /* END DO WHILE NOT EXIT */
 CALL PLIRETC(ZERO); /* SET EXIT RETURN CODE TO 0 */
 RETURN;

1/*****
/* GET TYPE OF LIST
*/
*****/

GET_TYPE: PROCEDURE;
 IF L NAMES = '*' THEN /* LIST DIRECTORY */
 TYPE = 'ALL';
 ELSE
 IF INDEX(L NAMES, '%') > 0 THEN
 DO; /* GENERIC LIST */
 TYPE = 'GENERIC';
 L NAMES = TRANSLATE(L NAMES, '%', ' ');
 F NAMES = TRANSLATE(F NAMES, '%', ' ');
 END; /* END IF GENERIC */
 ELSE
 DO; /* SPECIFIC NAME */
 TYPE = 'SPECIFIC';
 F NAMES = TRANSLATE(F NAMES, '%', ' ');
 END; /* END IF SPECIFIC */

```

```

END GET_TYPE;

1/***
/* GET LIST OF EMPLOYEES
/**

GET_LIST: PROCEDURE;

 SQLERRP = ' ' ; /* CONNECTION CHECK: INIT SQLERRP */

 SELECT (TYPE); /* OPEN CURSOR & GET FIRST RECORD */
 WHEN ('ALL') /* FOR ALL EMPLOYEES */
 DO;
 EXEC SQL OPEN TELE1;
 EXEC SQL FETCH TELE1
 INTO :PPHONE;
 END;
 WHEN ('GENERIC') /* FOR GENERIC EMPLOYEES */
 DO;
 EXEC SQL OPEN TELE2;
 EXEC SQL FETCH TELE2
 INTO :PPHONE;
 END;
 OTHERWISE /* FOR SPECIFIC EMPLOYEE(S) */
 DO;
 EXEC SQL OPEN TELE3;
 EXEC SQL FETCH TELE3
 INTO :PPHONE;
 END;
 END; /* SELECT */

/***
/* NO EMPLOYEE FULFILLED THE REQUEST
/**

SELECT;
WHEN (SQLERRP = ' ') /* NO CONNECTION TO DB2 */
 DO;
 CALL DSN8MPG (MODULE, '079E', OUTMSG);
 MSGS = OUTMSG; /* SET ISPF ERROR MESSAGE */
 END; /* END NO EMPLOYEE FOUND */
WHEN (SQLCODE = 100)
 DO;
 CALL DSN8MPG (MODULE, '008I', OUTMSG);
 MSGS = OUTMSG; /* SET ISPF ERROR MESSAGE */
 END; /* END NO EMPLOYEE FOUND */
OTHERWISE

/***
/* EMPLOYEES EXIST THAT FULFILL THE REQUEST. DISPLAY THEM.
/**

DO; /* BUILD RESULT TABLE */
 DIS_TABLE = YES;
 CALL ISPLINK(I_TBCREATE, TABLE_NAME, ' ', EMP_VARS, I_NOWRITE,
 I_REPLACE);
 DO WHILE (SQLCODE = 0); /* WHILE MORE ENTRIES */
 EMP_RECORD = PPHONE, BY NAME;
 CALL ISPLINK(I_TBADD, TABLE_NAME); /* ADD TO ISPF TABLE */
 SELECT (TYPE); /* GET NEXT RECORD */
 WHEN ('ALL')
 EXEC SQL FETCH TELE1 INTO :PPHONE;
 WHEN ('GENERIC')
 EXEC SQL FETCH TELE2 INTO :PPHONE;
 OTHERWISE
 EXEC SQL FETCH TELE3 INTO :PPHONE;
 END; /* END SELECT */
 END; /* END WHILE MORE */
 END; /* END EMPLOYEE FOUND */
END; /* END SELECT */

/***
/* CLOSE THE CURSORS
/**

SELECT (TYPE);
WHEN ('ALL')
 EXEC SQL CLOSE TELE1;
WHEN ('GENERIC')
 EXEC SQL CLOSE TELE2;
OTHERWISE
 EXEC SQL CLOSE TELE3;

```

```

 END;
 END GET_LIST;

1/***
/* DISPLAY/UPDATE EMPLOYEE PHONE NUMBERS */
/* */
 /**/

DISPLAY_LIST: PROCEDURE;

EXEC SQL WHENEVER SQLERROR CONTINUE; /* CHANGE ERROR HANDLING */
EXEC SQL WHENEVER SQLWARNING CONTINUE; /* FOR UPDATE */

CALL ISPLINK(I_TBTOP, TABLE_NAME);
CALL ISPLINK(I_TBDISPL, TABLE_NAME, DIS_PANEL);

DIS_EXIT = (PLIRETV = 8); /* WAS EXIT PRESSED? */
IF ^DIS_EXIT THEN /* IF EXIT NOT PRESSED */
 DO;
 CALL ISPLINK(I_VGET, DIS_VARS);
 MORE_CHANGES = (ROWS_CHANGED > 0); /* ANY CHANGES?
 DO WHILE(MORE_CHANGES); /* FIND PHONE NUM UPDATES */
 EXEC SQL UPDATE VEMPLP /* PERFORM UPDATE */
 SET PHONENUMBER = :EMP_RECORD.PHONENUMBER
 WHERE EMPLOYEENUMBER = :EMP_RECORD.EMPLOYEENUMBER;
 IF SQLCODE ^= 0 THEN /* IF UPDATE FAILED */
 DO;
 CALL DSN8MPG(MODULE, '060E', OUTMSG);
 SQL_PIC = SQLCODE;
 MSGS = SUBSTR(OUTMSG,1,46) || SQL_PIC;
 EXEC SQL ROLLBACK;
 MORE_CHANGES = NO;
 END; /* END UPDATE FAILED */
 ELSE /* SUCCESSFUL UPDATE */
 DO;
 CALL DSN8MPG(MODULE, '004I', OUTMSG);
 MSGS = OUTMSG;
 CALL ISPLINK(I_TBPUT, TABLE_NAME);
 IF ROWS_CHANGED > 1 THEN /* MORE CHANGES TO DO */
 DO; /* DISPLAY CHANGES */
 CALL ISPLINK(I_TBDISPL, TABLE_NAME);
 CALL ISPLINK(I_VGET, DIS_VARS);
 END;
 ELSE /* NO MORE CHANGES */
 MORE_CHANGES = NO;
 END; /* END SUCCESSFUL UPDATE */
 END; /* DO WHILE MORE CHANGES */
 CALL ISPLINK(I_TBCLOSE, TABLE_NAME); /* CLOSE ISPF TABLE */
 END; /* END IF ^DIS_EXIT */
END;

END DISPLAY_LIST;

1/***
/* ERROR HANDLING
 /**/

P3_DBERROR:

 CALL DSN8MPG(MODULE, '060E', OUTMSG); /* GET FULL MSG TEXT */
 SQL_PIC = SQLCODE;
 MSGS = SUBSTR(OUTMSG,1,46) || SQL_PIC; /* APPEND SQL_CODE */
 CALL ISPLINK(I_VPUT, MSGS_VAR); /* PUT MSG OUT */
 RETURN; /* EXIT PROGRAM */

END DSN8SP3;

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8EP1

PASE MANDATOS DE DB2 PARA QUE LOS EJECUTE EL PROGRAMA DE PROCEDIMIENTO ALMACENADO DSN8EP2.

DSN8EP1: PROCEDURE(PARMS) OPTIONS(MAIN);	00010000
/*****	00020000
* MODULE NAME = DSN8EP1 (SAMPLE PROGRAM)	* 00030000

```

* DESCRIPTIVE NAME = STORED PROCEDURE REQUESTER PROGRAM *
* LICENSED MATERIALS - PROPERTY OF IBM *
* 5675-DB2 *
* (C) COPYRIGHT 1982, 2000 IBM CORP. ALL RIGHTS RESERVED. *
* STATUS = VERSION 7 *
* FUNCTION = *
* PASS DB2 COMMANDS TO BE EXECUTED BY THE STORED *
* PROCEDURE PROGRAM DSN8EP2. GET INPUT FROM 'SYSIN'. *
* PASS THE COMMAND AND RECEIVE THE COMMAND RESULTS *
* VIA THE PARAMETERS CONTAINED IN THE EXEC SQL CALL *
* STATEMENT. WRITE THE RESULTS TO 'SYSPRINT'. *
* DEPENDENCIES = NONE *
* RESTRICTIONS = *
* 1. BEGIN DB2 COMMANDS WITH A HYPHEN AND END THEM *
* WITH A SEMICOLON. A '*' IN COLUMN ONE OR '--' *
* ANYWHERE ON A LINE (EXCEPT WITHIN A COMMAND) CAN *
* BE USED TO DENOTE COMMENTS. *
* 2. THIS PROGRAM ACCEPTS COMMANDS OF AT MOST 4096 BYTES. *
* PROGRAM SIZES = *
* THE FOLLOWING VARIABLES CAN BE CHANGED TO FIT THE *
* SPECIFIC ENVIRONMENT OF THE USER. *
* VARIABLE VALUE MEANING *
* NAME - - *
* PAGEWIDTH 133 MAXIMUM WIDTH OF A PAGE IN *
* CHARACTERS (INCLUDING THE CONTROL *
* CHARACTER IN COLUMN ONE) *
* MAXPAGWD 125 PRINT LINE WIDTH CONTROLLER = *
* MAXIMUM WIDTH - 1 (FOR CONTROL *
* CHARACTER) - 6 (LENGTH OF THE *
* COLUMN DISPLAY) - 1 (A '-' *
* BETWEEN THE COLUMN NUMBER DISPLAY *
* THE SQL OUTPUT DISPLAY). *
* MAXPAGLN 60 MAXIMUM NUMBER OF LINES ON THE *
* PRINT OUTPUT PAGES 2 THRN N. PAGE *
* 1 WILL HAVE MAXPAGLN + 1 LINES. *
* INPUTL 72 LENGTH OF THE INPUT RECORD *
* INPUT = *
* 1. INPUT STATEMENTS WILL BE TRANSFERRED *
* TO THE STATEMENT BUFFER WITH ONE BLANK BETWEEN *
* WORDS. *
* 2. BLANKS IN DELIMITED STRINGS WILL BE *
* TRANSFERRED INTO THE STATEMENT BUFFER *
* EXACTLY AS THEY APPEAR IN THE INPUT *
* STATEMENT. *
* 3. AN INPUT LINE CONSISTS OF CHARACTERS FROM *
* COLUMNS 1-INPUTL. IF AN INPUT STATEMENT SPANS *
* OVER MULTIPLE LINES, THE LINES ARE CONCATENATED *
* AND BLANKS ARE REMOVED SUCH THAT ONLY ONE *
* BLANK OCCURS BETWEEN WORDS. *
* MODULE TYPE = PROCEDURE *
* PROCESSOR = *
* ADMF PRECOMPILER *
* PL/I MVS/VM (FORMERLY PL/I SAA AD/CYCLE) *
* MODULE SIZE = 2K *
* ATTRIBUTES = RE-ENTERABLE *
* ENTRY POINT = DSN8EP1 *
* PURPOSE = SEE FUNCTION *
* LINKAGE = STANDARD MVS PROGRAM INVOCATION, ONE PARAMETER. *
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION: *
* 00040000 *
* 00050000 *
* 00060000 *
* 00070000 *
* 00080000 *
* 00090000 *
* 00100000 *
* 00110000 *
* 00120000 *
* 00130000 *
* 00140000 *
* 00150000 *
* 00160000 *
* 00170000 *
* 00180000 *
* 00190000 *
* 00200000 *
* 00210000 *
* 00220000 *
* 00230000 *
* 00240000 *
* 00250000 *
* 00260000 *
* 00270000 *
* 00280000 *
* 00290000 *
* 00300000 *
* 00310000 *
* 00320000 *
* 00330000 *
* 00340000 *
* 00350000 *
* 00360000 *
* 00370000 *
* 00380000 *
* 00390000 *
* 00400000 *
* 00410000 *
* 00420000 *
* 00430000 *
* 00440000 *
* 00450000 *
* 00460000 *
* 00470000 *
* 00480000 *
* 00490000 *
* 00500000 *
* 00510000 *
* 00520000 *
* 00530000 *
* 00540000 *
* 00550000 *
* 00560000 *
* 00570000 *
* 00580000 *
* 00590000 *
* 00600000 *
* 00610000 *
* 00620000 *
* 00630000 *
* 00640000 *
* 00650000 *
* 00660000 *
* 00670000 *
* 00680000 *
* 00690000 *
* 00700000 *
* 00710000 *
* 00720000 *
* 00730000 *
* 00740000 *
* 00750000 *
* 00760000 *
* 00770000 *
* 00780000 *
* 00790000 *
* 00800000 *
* 00810000 *
* 00820000 *
* 00830000 *
* 00840000 *
* 00850000

```

```

* SYMBOLIC LABEL/NAME = SYSIN * 00860000
* DESCRIPTION = DDNAME OF SEQUENTIAL DATA SET CONTAINING * 00870000
* DB2 COMMANDS TO BE EXECUTED. * 00880000
* OUTPUT = PARAMETERS EXPLICITLY RETURNED: * 00890000
* SYMBOLIC LABEL/NAME = SYSPRINT * 00900000
* DESCRIPTION = DDNAME OF SEQUENTIAL OUTPUT DATA SET TO * 00910000
* CONTAIN RESULTS OF THE COMMANDS EXECUTED. * 00920000
* * 00930000
* EXIT NORMAL = * 00940000
* * 00950000
* NO ERRORS WERE FOUND IN THE SOURCE AND NO * 00960000
* ERRORS OCCURRED DURING PROCESSING. * 00970000
* * 00980000
* * 00990000
* NORMAL MESSAGES = * 01000000
* * 01010000
* 1. THE FOLLOWING MESSAGE WILL BE GENERATED FOR ALL INPUT * 01020000
* STATEMENTS: * 01030000
* ***INPUT STATEMENT: DB2 COMMAND INPUT STATEMENT * 01040000
* * 01050000
* * 01060000
* * 01070000
* EXIT-ERROR = * 01080000
* * 01090000
* ERRORS WERE FOUND IN THE SOURCE, OR OCCURRED DURING * 01100000
* PROCESSING. * 01110000
* * 01120000
* RETURN CODE = 4 - WARNING-LEVEL ERRORS DETECTED. * 01130000
* SQLWARNING OR IFI WARNING FOUND DURING EXECUTION. * 01140000
* REASON CODE = 0 OR IFI REASON CODE * 01150000
* * 01160000
* RETURN CODE = 8 - ERRORS DETECTED. * 01170000
* SQLERROR OR IFI ERROR FOUND DURING EXECUTION. * 01180000
* REASON CODE = 0 OR IFI REASON CODE * 01190000
* * 01200000
* RETURN CODE = 12 - SEVERE ERRORS DETECTED. * 01210000
* ONE OF THE FOLLOWING ERRORS OCCURRED: * 01220000
* UNABLE TO OPEN FILES. * 01230000
* INTERNAL ERROR, ERROR MESSAGE ROUTINE RETURN CODE. * 01240000
* STATEMENT IS TOO LONG. * 01250000
* SQL OR IFI BUFFER OVERFLOW. * 01260000
* REASON CODE = 0 OR IFI REASON CODE * 01270000
* * 01280000
* ABEND CODES = NONE * 01290000
* * 01300000
* ERROR MESSAGES = * 01310000
* * 01320000
* 1. THE FOLLOWING MESSAGE WILL BE GENERATED WHEN A DB2 * 01330000
* COMMAND DOES NOT BEGIN WITH A HYPHEN "-". * 01340000
* * 01350000
* * 01360000
* * 01370000
* * 01380000
* 2. THE FOLLOWING MESSAGE WILL BE GENERATED WHEN AN INPUT * 01390000
* STATEMENT IS GREATER THAN STMTMAX SIZE: * 01400000
* * 01410000
* * 01420000
* * 01430000
* * 01440000
* * 01450000
* * 01460000
* * 01470000
* * 01480000
* * 01490000
* EXTERNAL REFERENCES = * 01500000
* ROUTINES/SERVICES = NONE * 01510000
* DSNTIAR - SQL COMMUNICATION AREA FORMATTING * 01520000
* DATA-AREAS = NONE * 01530000
* CONTROL-BLOCKS = * 01540000
* SQLCA - SQL COMMUNICATION AREA * 01550000
* * 01560000
* PSEUDOCODE = * 01570000
* * 01580000
* DSN8EP1: PROCEDURE. * 01590000
* DECLARATIONS. * 01600000
* INITIALIZE VARIABLES. * 01610000
* CALL READRTN TO READ IN A DB2 COMMAND STATEMENT. * 01620000
* DO UNTIL END-OF-FILE. * 01630000
* CALL READRTN TO READ A NEW DB2 COMMAND STATEMENT. * 01640000
* END. * 01650000
* * 01660000
* HEX2CHAR: PROCEDURE. * 01670000

```

```

* CONVERT THE RETURN CODE AND REASON CODE THAT ARE RETURNED * 01680000
* FROM THE IFI CALL FROM BINARY TO HEXADECIMAL. * 01690000
* END HEX2CHAR. * 01700000
* * 01710000
* PRINTCA: PROCEDURE. * 01720000
* CALL DSNTIAR TO FORMAT ANY MESSAGES. * 01730000
* IF A RETURN CODE WAS PASSED FROM DSNTIAR, INDICATE IT. * 01740000
* PRINT THE DATA FORMATTED FORMATTED BY DSNTIAR. * 01750000
* SET THE RETURN CODE TO 8. * 01760000
* END PRINTCA. * 01770000
* * 01780000
* READRTN: PROCEDURE. * 01790000
* SET ENDSTR = "NO". * 01800000
* SET REREAD = "NO". * 01810000
* DO WHILE (ENDSTR = NO). * 01820000
* FILL THE STATEMENT BUFFER FROM THE CURRENT INPUT LINE. * 01830000
* AVOID INITIAL BLANKS. * 01840000
* TERMINATE A STATEMENT WHEN A SEMICOLON IS FOUND. * 01850000
* VERIFY THAT COMMAND IS VALID. * 01860000
* DO SQL TO CALL DSN8EP2. * 01870000
* PROCESS THE COMMAND RESULTS. * 01880000
* SET REREAD FLAG. * 01890000
* RETURN TO CALLER. * 01900000
* END COMMAND. * 01910000
* END READRTN. * 01920000
* * 01930000
* RESULTS: PROCEDURE. * 01940000
* PROCESS THE RETURN CODE, REASON CODE, THE NUMBER OF * 01950000
* BYTES IN THE RETURN BUFFER, AND THE RETURN BUFFER * 01960000
* THAT ARE RETURNED FROM THE IFI CALL. * 01970000
* END RESULTS. * 01980000
* * 01990000
* CHANGE ACTIVITY =
* 6/29/95 UPDATED THE REMOTE LOCATION NAME VARIABLES (DB2LOC @03* 02010000
* & PARMs) TO ACCEPT A SIXTEEN CHARACTER NAME @03@ 02020000
* (PN69303) @03 KFF0296* 02030000
* 7/05/95 CHANGED THE OUTPUT STRING LENGTH FROM VARYING @35* 02040000
* TO FIXED 80 BYTE STRINGS (PN72035) @35 KFF0347* 02050000
* 8/28/95 ADDED ROLLBACK WORK STATEMENT TO ENSURE THAT DB2 @42* 02060000
* WORK IS ROLLED BACK IN ERROR SITUATIONS @42@ 02070000
* (PN74842) @42 KFF0580* 02080000
* 04/17/00 INITIALIZE STORAGE TO PREVENT RETURN CODE=04, * 02090000
* REASON CODE=00E60804 FROM IFI PQ36800* 02100000
* 05/22/03 FIX CODE HOLE CLOSED BY VA AND ENTERPRISE PL/I P044916* 02110000
*****PAGE;*/ 02120000
*****PAGE;*/ 02130000
/* VARIABLE DECLARATIONS */ 02140000
*****PAGE;*/ 02150000
*****PAGE;*/ 02160000
*****PAGE;*/ 02170000
*****PAGE;*/ 02180000
/* DECLARE IFI-RELATED VARIABLES */ 02190000
*****PAGE;*/ 02200000
DCL
 IFCA_RET_CODE CHAR(8) INIT(' ') /* RETURN CODE IN HEX */ 02220000
 IFCA_RES_CODE CHAR(8) INIT(' ') /* REASON CODE IN HEX */ 02230000
 INPUTCMD VAR CHAR(4096) INIT(' ') /* DB2 COMMAND */ 02240000
 IFCA_RET_HEX FIXED BIN(31) INIT(0) /* RETURN CODE PARAMETER */ 02250000
 IFCA_RES_HEX FIXED BIN(31) INIT(0) /* REASON CODE PARAMETER */ 02260000
 BUFF_OVERFLOW FIXED BIN(31) INIT(0) /* BUFFER OVERFLOW IND@35*/ 02270000
 REMBYTES FIXED BIN(15) INIT(0) /* BYTES REMAINING @35*/ 02280000
 RETURN_BUFF VAR CHAR(8320) INIT(' ') /* COMMAND RESULT @35*/ 02290000
 RETURN_IND FIXED BIN(15) INIT(0) /* INDICATOR VARIABLE @35*/ 02300000
 /* FOR RETURN_BUFFER */ 02310000
 /* */ 02320000
 /* */ 02330000
*****PAGE;*/ 02340000
/* CHARACTER CONSTANTS */ 02350000
*****PAGE;*/ 02360000
*****PAGE;*/ 02370000
DCL
 ASTERISK CHAR(1) INIT('*') STATIC /* COMMENT INDICATOR */ 02380000
 BLANK CHAR(1) INIT(' ') STATIC /* INITIALIZATION BLANKS */ 02400000
 HYPHEN CHAR(1) INIT('-') STATIC /* HYPHEN */ 02410000
 NULLCHAR CHAR(1) VAR INIT('') STATIC /* NULL CHARACTER */ 02420000
 QUOTE CHAR(1) INIT('}') STATIC /* QUOTATION MARK */ 02430000
 DQUOTE CHAR(1) INIT('\"') STATIC /* DOUBLE QUOTATION MARK */ 02440000
 SEMICOLON CHAR(1) INIT(';') STATIC /* SQL STMT TERMINATOR */ 02450000
 /* */ 02460000
*****PAGE;*/ 02470000
/* PROGRAM INPUT/OUTPUT CONSTANTS */ 02480000
*****PAGE;*/ 02490000

```

```

DCL
 INPUTL FIXED BIN(15) INIT(72) STATIC, /* SYSIN LRECL */ 02500000
 MAXPAGWD FIXED BIN(31) INIT(125) STATIC, /* OUTPUT WIDTH */ 02510000
 MAXPAGLN FIXED BIN(15) INIT(60) STATIC, /* # LINES / PAGE */ 02520000
 OUTLEN FIXED BIN(15) INIT(80) STATIC, /* LENGTH OF AN @35*/ 02530000
 /* OUTPUT LINE */ 02540000
 PAGEWIDTH FIXED BIN(31) INIT(133) STATIC; /* SYSOUT LRECL */ 02550000
 /* AREA LENGTH */ 02560000
 /* */ 02570000
 /* */ 02580000
 /* */ 02590000
/****** */
/* ERROR CODE CONSTANTS */ 02600000
/****** */
/****** */

DCL
 RETWRN FIXED BIN(15) INIT(4) STATIC, /* WARN RET COD @35*/ 02610000
 RETERR FIXED BIN(15) INIT(8) STATIC, /* ERROR RET CODE */ 02620000
 SEVERE FIXED BIN(15) INIT(12) STATIC; /* SEVERE ERROR */ 02630000
 /* RETURN CODE */ 02640000
 /* */ 02650000
 /* */ 02660000
 /* */ 02670000
 /* */ 02680000
 /* */ 02690000
/****** */
/* NUMBER CONSTANTS */ 02700000
/****** */
/****** */

DCL
 ZERO FIXED BIN(15) INIT(0) STATIC, 02710000
 ONE FIXED BIN(15) INIT(1) STATIC, 02720000
 TWO FIXED BIN(15) INIT(2) STATIC, 02730000
 FOUR FIXED BIN(15) INIT(4) STATIC, 02740000
 FIVE FIXED BIN(15) INIT(5) STATIC, 02750000
 EIGHT FIXED BIN(15) INIT(8) STATIC, 02760000
 TEN FIXED BIN(15) INIT(10) STATIC, 02770000
 /* */ 02780000
 /* */ 02790000
 /* */ 02800000
 /* */ 02810000
 /* */ 02820000
/****** */
/* FLAG CONSTANTS */ 02830000
/****** */
/****** */

DCL
 YES BIT(1) INIT('1'B) STATIC, /* BIT FLAG ON */ 02840000
 NO BIT(1) INIT('0'B) STATIC; /* BIT FLAG OFF */ 02850000
 /* */ 02860000
 /* */ 02870000
 /* */ 02880000
 /* */ 02890000
 /* */ 02900000
/****** */
/* INPUT / OUTPUT BUFFER VARIABLES DECLARATION */ 02910000
/****** */
/****** */

DCL
 COMMENT BIT(1) INIT('0'B), /* COMMENT ENCOUNTERED? */ 02920000
 CURPTR FIXED BIN(15) INIT(0), /* Curr Locn In Output @35*/ 02930000
 DB2LOC2 VAR CHAR(16) INIT(' '), /* Remote DB2 Loc Name @03*/ 02940000
 ENDSTR BIT(1) INIT('0'B), /* End Of Statement Flag */ 02950000
 EODIN BIT(1) INIT('0'B), /* End Of Input Data Flag */ 02960000
 ERR FIXED BIN(15) INIT(0), /* The Current Return Code*/ 02970000
 EXIT BIT(1) INIT('0'B), /* Program Exit Indicator */ 02980000
 I FIXED BIN(15) INIT(0), /* Loop Counter Variable */ 02990000
 INCOL FIXED BIN(15) INIT(0), /* Current Input Column */ 03000000
 INPUT(INPUTL) CHAR(1), /* Current Input Data */ 03010000
 J FIXED BIN(15) INIT(0), /* Loop Counter Variable */ 03020000
 K FIXED BIN(15) INIT(0), /* Loop Counter Variable */ 03030000
 KK FIXED BIN(15) INIT(0), /* Loop Counter Variable */ 03040000
 OSTMTLN FIXED BIN(15) INIT(0), /* # Of Output Lines Need */ 03050000
 /* */ 03060000
 /* */ 03070000
 /* */ 03080000
 /* */ 03090000
 /* */ 03100000
 /* */ 03110000
 /* */ 03120000
 /* */ 03130000
 /* */ 03140000
 /* */ 03150000
 /* */ 03160000
 /* */ 03170000
 /* */ 03180000
 /* */ 03190000
 /* */ 03200000
 /* */ 03210000
 /* */ 03220000
 /* */ 03230000
 /* */ 03240000
 /* */ 03250000
 /* */ 03260000
 /* */ 03270000
 /* */ 03280000
 /* */ 03290000
 /* */ 03300000
 /* */ 03310000
/****** */
/* BUILT IN FUNCTIONS DECLARATIONS */ 03320000
/****** */
/****** */

DCL
 ADDR BUILTIN, /* Function To Return The Address */ 03330000
 CHAR BUILTIN, /* Returns Char Representation */ 03340000
 LENGTH BUILTIN, /* Returns Length Of A String */ 03350000
 MIN BUILTIN, /* Function To Return Minimum */ 03360000
 NULL BUILTIN, /* Null Value */ 03370000
 SUBSTR BUILTIN, /* Function To Return Substring */ 03380000
 PLIRETC BUILTIN, /* Function To Set Return Code */ 03390000
 PLIRETV BUILTIN, /* PL/I Return Code Value */ 03400000
 UNSPEC BUILTIN, /* Ignores Variable Typing */ 03410000

```

```

/********************* 03320000
/* DECLARE BUFFER AREAS FOR THE SQLCA AND THE SQLDA */
/********************* 03330000
/********************* 03340000
/********************* 03350000
EXEC SQL INCLUDE SQLCA; /* DEFINE THE SQLCA */
 */ 03360000
 */ 03370000
/********************* 03380000
/* MESSAGE FORMATTING ROUTINE AND VARIABLES DECLARAIONS */
/********************* 03390000
/********************* 03400000
DCL
 DSNTIAR ENTRY EXTERNAL OPTIONS(ASM INTER RETCODE); 03410000
DCL
 MSGBLEN FIXED BIN(15) INIT(10); /* MAX # SQL MESSAGES */
 */ 03420000
 */ 03430000
DCL
 01 MESSAGE, /* RETURNED MESSAGES AREA */
 02 MESSAGELEN FIXED BIN(15) /* MESSAGE BUFFER LENGTH
 INIT(0),
 */ 03440000
 02 MESSAGEGET(MSGBLEN) CHAR(MAXPAGWD) /* SQLCA MSGS SPACE
 INIT(' ');
 */ 03450000
 */ 03460000
 */ 03470000
 */ 03480000
 */ 03490000
 */ 03500000
/********************* 03510000
/* BUFFER DECLARATION FOR THE INPUT STATEMENT */
/********************* 03520000
/* *** NOTE *** : THE CHARACTER SIZE MUST BE EXPLICIT FOR THE
/* PRECOMPILER */
/********************* 03540000
/********************* 03550000
 */ 03560000
DCL
 INPLLEN FIXED BIN(15) INIT(100), /* LENGTH OF PRINT STMT */
 STMTBUF VAR CHAR(4096) INIT(' '), /* STATEMENT STRING
 STMTLEN FIXED BIN(15) INIT(0), /* STMT STRING LENGTH
 STMTMAX FIXED BIN INIT(4096); /* STATEMENT BUFFER
 /* MAXIMUM LENGTH
 */ 03570000
 */ 03580000
 */ 03590000
 */ 03600000
 */ 03610000
 */ 03620000
 */ 03630000
/********************* 03640000
/* FILE DECLARATIONS */
/********************* 03650000
/********************* 03660000
 */ 03670000
DCL
 SYSIN FILE STREAM INPUT, /* INPUT FILE
 SYSPRINT FILE STREAM OUTPUT /* OUTPUT FILE
 ENV(FB,RECSIZE(PAGEWIDTH),BLKSIZE(PAGEWIDTH));
%PAGE;
/********************* 03680000
/* MAIN PROGRAM */
/********************* 03690000
/********************* 03700000
/* GENERAL INITIALIZATION */
/********************* 03710000
 */ 03720000
 */ 03730000
 */ 03740000
 */ 03750000
 */ 03760000
 */ 03770000
 */ 03780000
RETCODE = ZERO; /* INITIALIZE THE RETURN CODE
WRNING = NO; /* INITIALIZE PRINTING SQLCA ON
 /* WARNING FLAG
MESSAGEL = MSGBLEN * MAXPAGWD; /* SET MESSAGE BUFFER LENGTH
DB2LOC2 = PARMs; /* INPUT PARAMETER IS THE REMOTE
 /* DB2 LOCATION NAME
 */ 03790000
 */ 03800000
 */ 03810000
 */ 03820000
 */ 03830000
 */ 03840000
 */ 03850000
/********************* 03860000
/* INPUT PROCESSING INITIALIZATION */
/********************* 03870000
/********************* 03880000
 */ 03890000
EXIT = NO; /* DON'T EXIT-CONTINUE PROCESSING
EODIN = NO; /* NOT AT THE END OF INPUT DATA
INPUT = NULLCHAR; /* NULL THE INPUT DATA ARRAY
INCOL = INPUTL+ONE; /* SET COLUMN TO 73 TO INDICATE A
 /* NEW LINE IS TO BE READ IN
 /* READRTN
 */ 03900000
 */ 03910000
 */ 03920000
 */ 03930000
 */ 03940000
 */ 03950000
 */ 03960000
 */ 03970000
%PAGE;
/********************* 03980000
/* READ THE FIRST COMMAND STATEMENT TO BE PROCESSED */
/********************* 03990000
/********************* 04000000
 */ 04010000
CALL READRTN;
/********************* 04020000
/* MAIN LOOP. CONTINUE PROCESSING DB2 COMMANDS UNTIL THE END OF */
/********************* 04030000
/* DATA IS REACHED OR A SEVERE ERROR HAS BEEN ENCOUNTERED */
/********************* 04040000
 */ 04050000
 */ 04060000
 */ 04070000
PRC:
DO WHILE (EXIT = NO & RETCODE < SEVERE);
 ERR = ZERO; /* CLEAR THE CURRENT RETURN CODE
 /* INCLUDE OUTPUT HEADINGS
 CALL READRTN; /* READ NEXT STATEMENT
END; /* END PRC
 */ 04080000
 */ 04090000
 */ 04100000
 */ 04110000
 */ 04120000
 */ 04130000

```

```

GOTO STOPRUN; /* EXIT */ 04140000
%PAGE; /* */ 04150000
HEX2CHAR: /* */ 04160000
/* PROEDURE TO PRINT THE IFI RETURN CODE IN HEX */ 04170000
/****** */
PROCEDURE(INPUT) RETURNS(CHAR(8)); /* RESULTS RETURNED IN */ 04180000
 /* CHARACTER FORMAT */ 04190000
 /* RETURN CODE IN BINARY */ 04200000
 /****** */
DECLARE INPUT BIT(31),
 I1 BIT(4) DEF INPUT, /**/ 04210000
 I2 BIT(4) DEF INPUT POSITION(4), /**/ 04220000
 I3 BIT(4) DEF INPUT POSITION(8), /**/ 04230000
 I4 BIT(4) DEF INPUT POSITION(12),/**/ 04240000
 I5 BIT(4) DEF INPUT POSITION(16),/**/ 04250000
 I6 BIT(4) DEF INPUT POSITION(20),/**/ 04260000
 I7 BIT(4) DEF INPUT POSITION(24),/**/ 04270000
 I8 BIT(4) DEF INPUT POSITION(28),/**/ 04280000
 HEXES CHAR(16) INIT('0123456789ABCDEF'), /**/ 04290000
 OUTPUT CHAR(8), /**/ 04300000
 OUTPUT1(8) CHAR(1) DEFINED(OUTPUT); /**/ 04310000
 /**/ 04320000
 /**/ 04330000
 /**/ 04340000
 /**/ 04350000
OUTPUT1(1)=SUBSTR(HEXES,I1+1,1); /*1ST BYTE OF RET CODE IN HEX */ 04360000
OUTPUT1(2)=SUBSTR(HEXES,I2+1,1); /*2ND BYTE OF RET CODE IN HEX */ 04370000
OUTPUT1(3)=SUBSTR(HEXES,I3+1,1); /*3RD BYTE OF RET CODE IN HEX */ 04380000
OUTPUT1(4)=SUBSTR(HEXES,I4+1,1); /*4TH BYTE OF RET CODE IN HEX */ 04390000
OUTPUT1(5)=SUBSTR(HEXES,I5+1,1); /*5TH BYTE OF RET CODE IN HEX */ 04400000
OUTPUT1(6)=SUBSTR(HEXES,I6+1,1); /*6TH BYTE OF RET CODE IN HEX */ 04410000
OUTPUT1(7)=SUBSTR(HEXES,I7+1,1); /*7TH BYTE OF RET CODE IN HEX */ 04420000
OUTPUT1(8)=SUBSTR(HEXES,I8+1,1); /*8TH BYTE OF RET CODE IN HEX */ 04430000
RETURN (OUTPUT); /* RETURN THE OUTPUT RESULT*/ 04440000
END HEX2CHAR; /**/ 04450000
 /**/ 04460000
 /**/ 04470000
%PAGE; /****** */ 04480000
/* PROEDURE TO PRINT THE SQLCA ERROR INDICATION AND CLEAR OUT THE */ 04490000
/* SQLCA. OUTPUT MOST OF THE DATA ON AN EXCEPTION BASIS */ 04500000
/****** */ 04510000
 /**/ 04520000
PRINTCA: PROCEDURE; /**/ 04530000
 /**/ 04540000
/****** */
/* PROCESS SQL OUTPUT MESSAGE */ 04550000
/****** */ 04560000
/****** */ 04570000
 /**/ 04580000
CALL DSNTIAR (SQLCA, MESSAGE, MAXPAGWD); /* FORMAT ANY MESSAGES */ 04590000
IF PLIRETV ^= ZERO THEN /* IF THE RETURN CODE ISN'T ZERO */ 04600000
DO; /* ISSUE AN ERROR MESSAGE */ 04610000
 PUT EDIT (' *** RETURN CODE ', PLIRETV, /*@35*/ 04620000
 ' FROM MESSAGE ROUTINE DSNTIAR.') /**/ 04630000
 (COL(1), A(17), F(8), A(30)); /* ISSUE THE MESSAGE */ 04640000
 RETCODE = SEVERE; /* SET THE RETURN CODE */ 04650000
END; /* END ISSUE AN ERROR MESSAGE */ 04660000
 /**/ 04670000
DO I = ONE TO MSGBLEN /* PRINT OUT THE DSNTIAR BUFFER */ 04680000
WHILE (MESSAGET(I) ^= BLANK); /* PRINT NON BLANK LINES */ 04690000
 PUT EDIT (MESSAGET(I)) (COL(2), A(MAXPAGWD)); /**/ 04700000
END; /**/ 04710000
 /**/ 04720000
RETCODE = SEVERE; /* SET THE RETURN CODE */ 04730000
 /**/ 04740000
END PRINTCA; /**/ 04750000
 /**/ 04760000
 /**/ 04770000
 /**/ 04780000
 /**/ 04790000
/****** */
/* THIS PROCEDURE READS THE DATA FROM THE USER AND OBTAINS A DB2 */ 04800000
/* COMMAND TO PASS TO DSN8EP2 FOR EXECUTION VIA THE IFI CALL */ 04810000
/****** */ 04820000
 /**/ 04830000
 /**/ 04840000
READRTN: PROCEDURE;
DCL
 CONTLINE FIXED BIN(15) /* CONTINUATION LINE - INPUT STMT */ 04850000
 INIT(0), /* IS MORE THAN 72 CHARACTERS */ 04860000
 DQUOTFLAG BIT(1) /* DOUBLE QUOTE ("") ENCOUNTERED? */ 04870000
 INIT('0'B), /**/ 04880000
 FIRSTCHAR BIT(1) /* FIRST NON BLANK CHAR? */ 04890000
 INIT('0'B), /**/ 04900000
 LASTCHAR CHAR(1) /* LAST CHARACTER IN THE BUFFER */ 04910000
 INIT(' '), /**/ 04920000
 /**/ 04930000
 /**/ 04940000
 /**/ 04950000

```

```

MOVECHAR BIT(1) /* MOVE CHAR INTO STMT BUFFER? */ 04960000
 INIT('0'B),
NBLK FIXED BIN(15) /* NUMBER OF BLANKS FOUND */ 04970000
 INIT(0),
NEWFSET FIXED BIN(15) /* FIRST POSITION OF THE COMMAND */ 04980000
 INIT(0),
NEWSTMT BIT(1) /* NEW STMT TO BE PROCESSED? */ 04990000
 INIT('0'B),
QUOTEFLAG BIT(1) /* QUOTE (') ENCOUNTERED? */ 05000000
 INIT('0'B); 05010000
/****** */
/* ENDFILE CONDITIONS */ 05020001
/****** */
05030000
05040000
05050000
05060000
05070000
05080000
05090000
05100000
ON ENDFILE(SYSIN) /* PROCESS EOF ON INPUT FILE */ 05110000
BEGIN; /* END OF FILE */ 05120000
 IF LENGTH(STMTBUF) = 0 THEN
 DO; /* LENGTH(STMTBUF) = 0 */ 05130000
 EXIT = YES; /* NO STMT TO PROCESS, */ 05140000
 GOTO ENDRD; /* SO END THE PROGRAM */ 05150000
 END;
 ELSE; /* PROCESS THE CURRENT STATEMENT */ 05160000
 DO; /* LENGTH(STMTBUF) ^= 0 */ 05170000
 EODIN = YES; /* SIGNAL END_OF_DATA */ 05180000
 ENDSTR = YES; /* SIGNAL END_OF_STRING */ 05190000
 GOTO CHKCOMM; /* PROCESS CURRENT COMMAND */ 05200000
 END;
 END; /* END LENGTH(STMTBUF) ^= 0 */ 05210000
 /* END END OF FILE */ 05220000
 /* END */ 05230000
 /* END */ 05240000
 /* END */ 05250000
/****** */
/* BEGIN READRTN PROCESSING */ 05260000
/****** */
05270000
05280000
05290000
NEWSTMT= YES; /* NEW STMT IS BEING PROCESSED */ 05300000
%PAGE; 05310000
 05320000
 05330000
/****** */
/* READ IN THE INPUT STATEMENT */ 05340000
/****** */
05350000
05360000
05370000
RD: 05380000
 05390000
DO WHILE (NEWSTMT = YES); 05400000
 05410000
/****** */
/* NO MORE INPUT DATA (EOF) SO RETURN TO CALLER */ 05420000
/****** */
05430000
05440000
05450000
IF EODIN = YES THEN 05460000
 DO; /* END_OF DATA */ 05470000
 EXIT = YES; /* EXIT PROGRAM */ 05480000
 LEAVE RD; /* LEAVE THE LOOP */ 05490000
 END; /* END END_OF DATA */ 05500000
 05510000
/****** */
/* PROCESS THE STATEMENT */ 05520000
/****** */
05530000
05540000
05550000
ELSE /* MORE INPUT TO PROCESS */ 05560000
 DO;
 NEWSTMT = NO; /* TURN NEW STATEMENT FLAG OFF */ 05570000
 CONTLINE = ZERO; /* CLEAR MULTILINE STMT COUNTER */ 05580000
 ENDSTR = NO; /* NOT AT THE END OF THE STRING */ 05590000
 QUOTEFLAG = NO; /* INITIALIZE QUOTE FLAG */ 05600000
 DQUOTFLAG = NO; /* INITIALIZE DOUBLE QUOTE FLAG */ 05610000
 STMTLEN = ZERO; /* INITIALIZE THE STMT LENGTH */ 05620000
 STMTBUF = NULLCHAR; /* INIT STMT BUFFER TO NULLS */ 05630000
 LASTCHAR = NULLCHAR; /* INIT. LAST CHARACTER TO NULL */ 05640000
 COMMENT = NO; /* INITIALIZE THE COMMENT FLAG */ 05650000
 FIRSTCHAR = NO; /* INIT. FIRST CHAR TO NO */ 05660000
 NBLK = ZERO; /* INIT. BLANK COUNT TO 0 */ 05670000
 05680000
 05690000
/****** */
/* READ AND PROCESS A NEW STATEMENT */ 05700000
/****** */
05710000
05720000
05730000
DO WHILE (ENDSTR = NO); /* PUT INPUT STMT IN STMT BUFFER */ 05740000
 05750000
/****** */
/* IF THE COLUMN BEING PROCESSED IS GREATER THAN THE */ 05760000
 05770000

```

```

/* LENGTH OF THE INPUT LINE THEN READ THE NEXT LINE */ 05780000
/***/ 05790000
05800000
05810000
IF INCOL > INPUTL THEN 05820000
DO; /* GET SYSIN DATA */ 05830000
 GET EDIT (INPUT) (COL(1), (INPUTL) A(1)); /* */
 INCOL = ONE; /* POINT TO FIRST CHARACTER */ 05840000
 IF FIRSTCHAR = YES THEN /* FIRST CHAR SET? */ 05850000
 CONTLINE = CONTLINE + 1; /* INCREMENT INPUT LINE CTR */ 05860000
 END; 05870000
05880000
/***/ 05890000
/* THE CHARACTER IN COLUMN ONE IS AN ASTERISK OR THE */ 05900000
/* CHARACTERS IN COLUMNS 1 AND 2 ARE '--'. CONSIDER THIS */ 05910000
/* LINE TO BE A COMMENT. PRINT THE LINE AND RETRIEVE THE */ 05920000
/* NEXT INPUT LINE. */ 05930000
/***/ 05940000
05950000
IF INCOL = 1 & (INPUT(1) = ASTERISK 05960000
 | (INPUT(1) = HYPHEN & INPUT(2) = HYPHEN)) 05970000
 & STMTLEN = 0 THEN 05980000
DO; /* STATEMENT IS A COMMENT */ 05990000
 DO J = 1 TO INPUTL; /* PUT ENTIRE LINE INTO STMTBUF */ 06000000
 STMTBUF = STMTBUF || INPUT(J); 06010000
 END; 06020000
 STMTLEN = LENGTH(STMTBUF); 06030000
 ENDSTR = YES; /* INDICATE END OF A STRING */ 06040000
 NEWSTMT = YES; /* NEW STMT SHOULD BE READ */ 06050000
 INCOL = INPUTL + ONE; /* SET INDEX TO 73 TO FORCE */ 06060000
 /* THE NEXT STMT TO BE READ */ 06070000
 COMMENT= ^COMMENT; /* SET COMMENT INDICATOR ON */ 06080000
 END; /* END STATEMENT IS A COMMENT */ 06090000
06100000
/***/ 06110000
/* PROCESS THE INPUT STATEMENT */ 06120000
/***/ 06130000
06140000
ELSE 06150000
DO;
06160000
06170000
/***/ 06180000
/* MOVE THE CHARACTER FROM THE INPUT DATA INTO THE */ 06190000
/* STATEMENT BUFFER UNTIL AN END OF LINE CHARACTER */ 06200000
/* OR SEMICOLON IS ENCOUNTERED */ 06210000
/***/ 06220000
06230000
DO J = INCOL TO INPUTL WHILE (^ENDSTR); 06240000
06250000
/***/ 06260000
/* PREPROCESS ANY DOUBLE QUOTATION MARKS ("). IF THE */ 06270000
/* DOUBLE QUOTATION MARK IS CONTAINED BETWEEN */ 06280000
/* QUOTATION MARKS ('), THE QUOTATION MARK IS */ 06290000
/* CONSIDERED TO BE THE STRING DELIMITER. THE */ 06300000
/* DQUOTFLAG WILL NOT BE SET. IN THIS CASE THE */ 06310000
/* DOUBLE QUOTATION MARK IS CONSIDERED TO BE PART OF */ 06320000
/* THE STRING */ 06330000
/***/ 06340000
06350000
IF INPUT(J) = DQUOTE THEN 06360000
DO; /* INPUT(J)=DQUOTE */ 06370000
 IF ^QUOTEFLAG THEN /* NOT DELIMITED BY QUOTES */ 06380000
 /* THEN DOUBLE */ 06390000
 /* QUOTES ARE */ 06400000
 DQUOTFLAG = ^DQUOTFLAG; /* THE DELIMITER */ 06410000
 END; /* END INPUT(J) = DQUOTE */ 06420000
06430000
/***/ 06440000
/* PREPROCESS ANY QUOTATION MARKS ('). IF THE */ 06450000
/* QUOTATION MARK IS CONTAINED BETWEEN DOUBLE */ 06460000
/* QUOTATION MARKS ("), THE DOUBLE QUOTATION MARK IS */ 06470000
/* CONSIDERED TO BE THE STRING DELIMITER. THE */ 06480000
/* QUOTEFLAG WILL NOT BE SET. IN THIS CASE THE */ 06490000
/* QUOTATION MARK IS CONSIDERED TO BE PART OF THE */ 06500000
/* STRING. */ 06510000
/***/ 06520000
06530000
IF INPUT(J) = QUOTE THEN 06540000
DO; /* INPUT(J) = QUOTE */ 06550000
 IF ^DQUOTFLAG THEN /* NOT DELIMITED BY */ 06560000
 /* DOUBLE QUOTES THEN */ 06570000
 /* SINGLE QUOTES ARE THE */ 06580000
 QUOTEFLAG = ^QUOTEFLAG; /* DELIMITER */ 06590000

```

```

END; /* END INPUT(J) = QUOTE */ 06600000
 06610000
/****** */
/* PROCESS A HYPHEN IF FOUND. THE HYPHEN IS */ 06620000
/* CONSIDERED PART OF A STRING IF A DELIMITER FLAG */ 06630000
/* IS SET. IF THE FOLLOWING CHARACTER IS A HYPHEN, */ 06640000
/* MOVE THE REMAINING CHARACTERS TO THE STATEMENT */ 06650000
/* BUFFER. */ 06660000
/****** */
IF (INPUT(J) = HYPHEN) & /*INPUT CHAR IS '-' */ 06670000
(J < INPUTL) & /* STILL MORE & */ 06710000
^QUOTEFLAG & /* NOT CURRENTLY IN */ 06720000
^DQUOTFLAG THEN /* DELIMITED STRING THEN */ 06730000
DO; /* LOOK FOR '--' */ 06740000
 IF INPUT(J+1) = HYPHEN /* FOUND '--' */ 06750000
 DO; /* DO NOT MOVE CHARACTERS */ 06760000
 MOVECHAR = NO; /* INTO THE STATEMENT BUFFER*/ 06770000
 END;
 IF (INPUT(J+1) = HYPHEN) & /*MOVECHAR = NO) THEN /* COMMENT FOUND */ 06790000
 (MOVECHAR = NO) THEN /* COMMENT FOUND */ 06800000
 DO; /* STATEMENT IS A COMMENT*/ 06810000
 DO J = 1 TO INPUTL; 06820000
 STMTBUF = STMTBUF || INPUT(J); 06830000
 END; /* PUT ENTIRE LINE INTO STMTBUF */ 06840000
 STMTLEN = LENGTH(STMTBUF); 06850000
 ENDSTR = YES; /* INDICATE END OF A STRING */ 06860000
 NEWSTMT = YES; /* NEW STMT SHOULD BE READ */ 06870000
 INCOL = INPUTL + ONE; /* SET INDEX TO 73 */ 06880000
 /* TO FORCE THE NEXT STATEMENT */ 06890000
 /* TO BE READ */ 06900000
 COMMENT= ^COMMENT; /* SET THE COMMENT */ 06910000
 /* INDICATOR ON */ 06920000
 END; /* END STATEMENT IS A COMMENT */ 06930000
 END; /* END LOOK FOR '--' */ 06940000
/****** */
/* PROCESS THE END-OF-STRING IF A SEMICOLON IS */ 06950000
/* FOUND. THE SEMICOLON CANNOT BE CONTAINED WITHIN */ 06960000
/* A DELIMITED STRING. THE ACCEPTABLE DELIMITERS */ 06970000
/* ARE QUOTE OR DOUBLE QUOTE MARKS. */ 06980000
/****** */
IF (INPUT(J) = SEMICOLON) & ^DQUOTFLAG & 07020000
^QUOTEFLAG THEN /* SEMICOLON & NOT */ 07030000
 ENDSTR = ^ENDSTR; /* DELIMITED THEN SET END */ 07040000
 /* OF STRING */ 07050000
/****** */
/* NOT THE END OF THE STRING, PROCESS THE STATEMENT */ 07060000
/****** */
ELSE
 DO; 07100000
 /****** */
 /* MOVE ALL NON BLANK CHARACTERS INTO THE DB2 */ 07130000
 /* COMMAND STATEMENT BUFFER */ 07140000
 /****** */
 IF INPUT(J)^= BLANK THEN 07180000
 DO; 07190000
 MOVECHAR = YES; 07200000
 FIRSTCHAR = YES; 07210000
 NBLK = ZERO; 07220000
 END; 07230000
 /****** */
 /* A BLANK SHOULD BE MOVED IN THE FOLLOWING CASES: */ 07250000
 /* 1. IF THE BLANK IS IN A DELIMITED STRING */ 07260000
 /* 2. IF AN INPUT STATEMENT SPANS MORE THAN */ 07270000
 /* ONE LINE AND THE PREVIOUS LINE HAD A */ 07280000
 /* CHARACTER IN COLUMN 72 AND THE CURRENT */ 07290000
 /* LINE HAS BLANKS BEFORE THE FIRST WORD */ 07300000
 /****** */
 ELSE /* BLANK CHARACTER FOUND */ 07360000
 DO; 07370000
 IF QUOTEFLAG | DQUOTFLAG | 07380000
 (CONTLINE >= 1 & J = 1 & NBLK = 0) THEN 07390000
 DO; /* BLANK IS DELIMITED, MOVE */ 07400000
 MOVECHAR = YES; /* IT INTO STMT BUFFER*/ 07410000

```

```

 NBLK = NBLK + ONE; /* & INC BLANK COUNT */ 07420000
 END;
 ELSE /* BLANK NOT DELIMITED */ 07430000
 DO;
 NBLK = NBLK + ONE; /* INCREASE BLANK CTR */ 07440000
 IF (NBLK = ONE) & (FIRSTCHAR = YES) THEN 07450000
 MOVECHAR = YES;
 ELSE 07460000
 DO;
 MOVECHAR = NO; 07470000
 END;
 END; /* END BLANK NOT DELIMITED */ 07480000
 /* END BLANK CHARACTER FOUND */ 07490000
 END; 07500000
 07510000
 07520000
 END; /* END BLANK NOT DELIMITED */ 07530000
 /* END BLANK CHARACTER FOUND */ 07540000
 END; 07550000
 **** 07560000
 /* IF MOVECHAR IS SET THEN MOVE THE INPUT */ 07570000
 /* CHARACTER INTO STATEMENT BUFFER AREA */ 07580000
 **** 07590000
 **** 07600000
 IF MOVECHAR = YES THEN 07610000
 DO;
 **** 07620000
 **** 07630000
 **** 07640000
 /* WHEN THE STATEMENT LENGTH IS TOO LONG,THE */ 07650000
 /* STATEMENT CANNOT BE PROCESSED. A RETURN */ 07660000
 /* CODE IS SET TO INDICATE NO FURTHER */ 07670000
 /* PROCESSING SHOULD BE DONE. AN ERROR */ 07680000
 /* MESSAGE WILL BE PUT OUT. */ 07690000
 **** 07700000
 **** 07710000
 STMTLEN = LENGTH(STMTBUF); 07720000
 IF STMTLEN = STMTMAX THEN /* STMT TOO LONG */ 07730000
 DO;
 RETCODE = SEVERE; /* SET RETURN CODE */ 07740000
 PUT EDIT(' *** ERROR: STATEMENT GREATER ',
 ' THAN ',STMTMAX,' CHARACTERS. ',
 'STMT: ') /* @35*/ 07750000
 (COL(1),A(31),A(5),F(4),A(13),
 A(7)); /* @35*/ 07760000
 PUT EDIT((SUBSTR(STMTBUF,KK,
 MIN(100,STMTLEN-KK+1))
 DO KK = 1 TO STMTLEN BY 100))
 (COL(2),A(100)); /* @35*/ 07770000
 LEAVE RD; 07780000
 END; /* END STMT TOO LONG */ 07790000
 STMTBUF = STMTBUF || INPUT(J);
 END; /* MOVE CHARACTER INTO BUFFER */ 07800000
 LASTCHAR = INPUT(J); /* SAVE THIS CHARACTER */ 07810000
 END; /* END CHARACTER NOT A SEMICOLON */ 07820000
 END; /* END DO J = INCOL TO INPUTL */ 07830000
 INCOL = J; /* UPDATE THE INPUT COLUMN */ 07840000
 END; /* END DO WHILE (ENDSTR = NO) */ 07850000
 **** 07860000
 **** 07870000
 **** 07880000
 **** 07890000
 **** 07900000
 **** 07910000
 **** 07920000
 **** 07930000
 **** 07940000
 **** 07950000
 **** 07960000
 /* CHECK WHETHER THE COMMAND ENTERED IS A COMMENT. IF NOT, */ 07970000
 /* PRINT THE DB2 COMMAND INPUT STATEMENT. */ 07980000
 **** 07990000
 **** 08000000
CHKCOMM:
IF ^COMMENT THEN 08010000
DO; 08020000
 STMTLEN = LENGTH(STMTBUF);
 NEWOFSET = ONE;
END; 08030000
 08040000
 08050000
 **** 08060000
 /* PRINT OUT THE DB2 COMMAND INPUT STATEMENT */ 08070000
 **** 08080000
PUT SKIP; 08090000
IF ^COMMENT THEN 08100000
DO; 08110000
 PUT SKIP; /*@35*/ 08120000
 PUT EDIT (' *** INPUT STATEMENT: ') (COL(1), A); /*@35*/
 J = STMTLEN; /*@35*/ 08130000
 PUT EDIT ((SUBSTR(STMTBUF,KK,MIN(INPLLEN,J-KK+1))
 DO KK = 1 TO STMTLEN BY INPLLEN))
 (A(INPLLEN),COL(1)); 08140000
 08150000
 08160000
 08170000
 08180000
END; 08190000
ELSE /*@35*/ 08200000
DO; /*@35*/
 J = STMTLEN; /*@35*/
 PUT EDIT ((SUBSTR(STMTBUF,KK,MIN(INPLLEN,J-KK+1))
 DO KK = 1 TO STMTLEN BY INPLLEN)) /*@35*/ 08210000
 /*@35*/ 08220000
 /*@35*/ 08230000

```

```

 (COL(2),A(INPLLEN),COL(1)); /*@35*/ 08240000
 END; /*@35*/ 08250000
 IF ^COMMENT THEN
 STMTBUF = SUBSTR(STMTBUF,ONE,STMTLEN);
 /***** *****/
 /* UPDATE THE OUTPUT LINE COUNTER */
 /***** *****/
 /***** *****/
 OSTMLN = STMTLEN/INPLLEN; /* # LINES NEEDED FOR */
 /* INPUT STMT */
 IF OSTMLN * INPLLEN ^= STMTLEN THEN
 OSTMLN = OSTMLN + ONE;
 /***** *****/
 /* CHECK THAT THE DB2 COMMAND BEGINS WITH A HYPHEN. */
 /* IF NOT, CALL BADCMD AND ISSUE AN ERROR */
 /* MESSAGE. */
 /***** *****/
 /***** *****/
 IF ^COMMENT THEN
 DO; /* STATEMENT NOT A COMMENT */
 /***** *****/
 /* HANDLE BAD IFI CALL SYNTAX */
 /***** *****/
 IF SUBSTR(STMTBUF,ONE,ONE) ^= '-' THEN /* NO HYPHEN */
 DO;
 PUT SKIP;
 PUT SKIP EDIT(' *** SYNTAX FOR DB2 COMMAND ',/*@35*/
 'IS NOT VALID.') /*@35*/
 (COL(1),A(28),A(13));
 PUT SKIP EDIT(' *** A VALID COMMAND MUST ', /*@35*/
 'BEGIN WITH A HYPHEN.') /*@35*/
 (COL(1),A(26),A(20));
 RETCODE = RETERR; /* SET RET CODE TO 8 */
 END; /* END NO HYPHEN */
 /***** *****/
 /* COMMAND SYNTAX IS CORRECT */
 /***** *****/
 ELSE
 DO; /* A VALID */
 INPUTCMD = SUBSTR(STMTBUF,ONE,STMTLEN); /* COMMAND*/
 /*SO MAKE CALL*/
 /***** *****/
 /* CONNECT TO THE DB2 REMOTE LOCATION */
 /***** *****/
 EXEC SQL CONNECT TO :DB2LOC2; /* CONNECT TO */
 /* REMOTE LOCATION */
 IF SQLCODE < 0 THEN /* SQL ERROR? @42*/
 DO; /* YES, ERROR FOUND*/
 PUT EDIT (' *** CONNECTION TO ',DB2LOC2, /*@35*/
 ' NOT SUCCESSFUL:')
 (COL(1), A(19), A(16), A(16));
 CALL PRINTCA; /* PRINT ERROR MSG */
 GOTO STOPRUN; /* END PROGRAM */
 END; /* END ERROR FOUND */
 /***** *****/
 /* CALL THE STORED PROCEDURE PROGRAM DSN8EP2 */
 /***** *****/
 RETURN_IND = -1; /*@35*/
 EXEC SQL CALL DSN8.DSN8EP2(:INPUTCMD,
 :IFCA_RET_HEX,
 :IFCA_RES_HEX,
 :BUFF_OVERFLOW, /*@35*/
 :RETURN_BUFF:RETURN_IND); /*@35*/
 IF SQLCODE < 0 THEN /* SQL ERROR? @42*/
 DO; /* YES ERROR FOUND*/
 PUT EDIT (' *** CALL TO DSN8EP2 NOT SUCCESSFUL:')
 (COL(1),A(36));
 IF SQLCODE = -911 | SQLCODE = -918 /*@42*/
 | SQLCODE = -919 | SQLCODE = -965 /*@42*/
 THEN /* CHECK FOR SPECIFIC ERRORS */
 /* THAT REQUIRE A ROLL BACK @42*/
 DO; /* YES, ROLL BACK REQUIRED @42*/
 CALL PRINTCA; /* PRINT ERROR MSG @42*/
 PUT EDIT (' *** ISSUE ROLLBACK WORK ',
 'BECAUSE STORED PROCEDURE ',
 'CALL NOT SUCCESSFUL')
 (COL(1), A(25), A(25), A(19));
 /* PRINT ROLLBACK WORK MESSAGE @42*/
 EXEC SQL ROLLBACK WORK; /* EXECUTE ROLLBACK*/
 /* WORK STMT @42*/
 END; /* END ROLL BACK REQUIRED @42*/

```

```

 CALL PRINTCA; /* PRINT ERROR MSG */ 09060000
 GOTO STOPRUN; /* END PROGRAM */ 09070000
 END; /* END ERROR FOUND */ 09080000
/****** */
/* CALL THE RESULTS PROC TO PROCESS THE RETURN CODE, THE REASON */ 09090000
/* CODE AND THE RESULTS MESSAGE OF THE COMMAND EXECUTED BY IFI. */ 09100000
/* NEXT, INITIALIZE THE VARIABLES TO PROCESS THE NEXT DB2 COMMAND. */ 09120000
/****** */
 CALL RESULTS; /* PROCESS THE RESULTS */ 09140000
 END; /* END VALID COMMAND */ 09150000
 NEWFSET = ZERO; /* RESET CHARACTER PTR */ 09160000
 NEWSTMT = YES; /* RESET FOR NEW STMT */ 09170000
 END; /* END STATEMENT NOT A COMMENT */ 09180000
 END; /* END ELSE MORE INPUT */ 09190000
 END; /* END DO WHILE NEW STMT */ 09200000
09210000
ENDRD:;
END READRTN;
09220000
09230000
09240000
09250000
09260000
/****** */
/* PROCESS THE DB2 COMMAND RESULTS FROM THE IFCA RETURN BUFFER */ 09270000
/****** */
RESULTS: PROCEDURE;
09300000
DCL
M0LENGTH CHAR(2) INIT(' '), /* LENGTH OF CMD RESULT */ 09320000
M1LENGTH BIT(16) INIT('0'B), /* INTERNALLY STORED LNG */ 09330000
M2LENGTH FIXED BIN(15) INIT(0), /* LENGTH OF MESSAGE N */ 09340000
BEGINSTR FIXED BIN(15) INIT(1), /* CHAR 1 POINTER */ 09350000
TOTBYTES FIXED BIN(31) INIT(0); /* MSG BYTE COUNT */ 09360000
09370000
IFCA_RET_CODE = HEX2CHAR(IFCA_RET_HEX); /* RETURN CODE IN HEX */ 09380000
IFCA_RES_CODE = HEX2CHAR(IFCA_RES_HEX); /* REASON CODE IN HEX */ 09390000
TOTBYTES = 0; /* INITIALIZE COUNTER */ 09400000
BEGINSTR = 1; /* INITIALIZE POINTER */ 09410000
09420000
IF IFCA_RET_HEX ^= 0 THEN /* IF THE RETURN CODE ISN'T ZERO */ 09430000
 /* ISSUE AN ERROR MESSAGE */ 09440000
DO;
 PUT EDIT(' *** RETURN CODE=',SUBSTR(IFCA_RET_CODE,7,2), /*@35*/
 ' REASON CODE=',IFCA_RES_CODE,' FROM IFI REQUEST')
 (COL(1),A(17),A(2),A,A(8),A); /*@35*/
END; /* END ISSUE AN ERROR MESSAGE */ 09490000
09500000
IF LENGTH(RETURN_BUFF) ^= 0 THEN /*@35*/
 /* DON'T PRINT UNLESS SOME DATA RET. */ 09510000
09520000
DO;
 PUT SKIP; /*@35*/ 09540000
 PUT SKIP EDIT(' *** IFI RETURN AREA:') /*@35*/
 (COL(1),A); /*@35*/ 09550000
09560000
/****** */
/* PROCESS THE UNFORMATTED COMMAND RESULTS FROM THE IFI CALL. */ 09570000
/* GET THE LENGTH OF EACH RESULT LINE FROM THE FIRST TWO */ 09580000
/* BYTES. PUT IT IN USABLE FORM. PRINT THE RESULTS FROM */ 09590000
/* THE FIRST LINE. UPDATE THE POINTER AND THE COUNTERS AND */ 09600000
/* REPEAT UNTIL ALL BYTES FROM IFCA_BYTES_MOVED HAVE BEEN */ 09610000
/* PROCESSED. */ 09620000
09630000
/****** */
CURPTR = 0; /* START OF DATA IN RET AREA@35*/ 09650000
REMBYTES = LENGTH(RETURN_BUFF); /* NUMBER OF BYTES TO PROC@35*/ 09660000
DO WHILE (REMBYTES > 0); /* RETURN AREA PRINT LOOP @35*/
 PRTBUF = SUBSTR(RETURN_BUFF,CURPTR,OUTLEN); /*@35*/
 SUBSTR(PRTBUF,1,1) = BLANK; /* BLANK FIRST COLUMN TO @35*/ 09680000
 /* AVOID CARRIAGE CTRL PROB */ 09690000
 PUT SKIP EDIT (PRTBUF) (COL(1),A(OUTLEN)); /*@35*/ 09710000
 CURPTR = CURPTR + OUTLEN; /*@35*/ 09720000
 REMBYTES = REMBYTES - OUTLEN; /*@35*/ 09730000
END; /*@35*/ 09740000
09750000
END; /* END IFCA_BYTES_MOVED ^= 0 */ 09760000
09770000
IF BUFF_OVERFLOW = 1 THEN /* COULDN'T GET ALL DATA @35*/ 09780000
DO;
 PUT SKIP EDIT (' *** INSUFFICIENT SPACE TO RECEIVE ', /*@35*/
 'ALL OUTPUT FROM IFI RETURN AREA.')
 (A(35),A(32)); /*@35*/ 09790000
09800000
 IF RETCODE < RETWRN THEN
 RETCODE = RETWRN; /*@35*/ 09820000
 END; /*@35*/ 09830000
09840000
IF IFCA_RET_HEX > RETCODE THEN /* CHECK RETURN CODES */ 09850000
 RETCODE = IFCA_RET_HEX; /* USE THE HIGHEST ONE */ 09860000
09870000

```

```

IF IFCA_RET_HEX = SEVERE THEN /* IF RETURN CODE = 12 */ 09880000
 GOTO STOPRUN; /* STOP PROGRAM EXECUTION*/ 09890000
 /* END RESULTS PROC */ 09900000
END RESULTS; /* **** */ 09910000
/* SET THE PL/I RETURN CODE AND TERMINATE PROCESSING */ 09920000
/* **** */ 09930000
/* **** */ 09940000
 09950000
 09960000
STOPRUN:
IF RETCODE >= SEVERE THEN /*@35*/ 09970000
 DO; /*@35*/ 09980000
 PUT SKIP;
 PUT SKIP EDIT (' *** SEVERE ERROR OCCURRED. ',
 'PROGRAM IS TERMINATING.') /*@35*/ 10000000
 (A(28),A(23)); /*@35*/ 10010000
 END; /*@35*/ 10020000
 /*@35*/ 10030000
CALL PLIRETC(RETCODE); /* SET PLI RETURN CODE */ 10040000
END DSN8EP1; /* END PROGRAM */ 10050000

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8EP2

UTILIZANDO LA INTERFAZ DE RECURSO DE INSTRUMENTALIZACIÓN (IFI), PROCESE UN MANDATO DE Db2 QUE SE PASA DESDE EL PROGRAMA SOLICITANTE DSN8EP1.

```

DSN8EP2: PROCEDURE(INPUTCMD, IFCA_RET_HEX, IFCA_RES_HEX,
 BUFF_OVERFLOW, RETURN_BUFF, NULL_ARRAY)
OPTIÖNS(MAIN NOEXECOPS);

/* **** */
* MODULE NAME = DSN8EP2 (SAMPLE PROGRAM) * 00010000
* * 00020000
* * 00030000
* * 00040000
* * 00050000
* DESCRIPTIVE NAME = STORED PROCEDURE SERVER PROGRAM * 00060000
* * 00070000
* LICENSED MATERIALS - PROPERTY OF IBM * 00080000
* 5675-DB2 * 00090000
* (C) COPYRIGHT 1982, 2000 IBM CORP. ALL RIGHTS RESERVED. * 00100000
* * 00110000
* STATUS = VERSION 7 * 00120000
* * 00130000
* FUNCTION = * 00140000
* USING THE INSTRUMENTATION FACILITY INTERFACE (IFI), * 00150000
* PROCESS A DB2 COMMAND WHICH IS PASSED FROM THE DSN8EP1 * 00160000
* REQUESTER PROGRAM. * 00170000
* * 00180000
* NOTES = * 00190000
* DEPENDENCIES = NONE * 00200000
* * 00210000
* RESTRICTIONS = * 00220000
* 1. THE INSTRUMENTATION FACILITY COMMUNICATION AREA * 00230000
* (IFCA) CONTAINS INFORMATION REGARDING THE SUCCESS * 00240000
* OF THE CALL AND PROVIDES FEEDBACK. * 00250000
* THIS AREA MUST BE MAINTAINED TO INCLUDE ANY CHANGES * 00260000
* TO THE MAPPING MACRO DSNDIFCA. * 00270000
* * 00280000
* * 00290000
* * 00300000
* MODULE TYPE = PROCEDURE * 00310000
* PROCESSOR = * 00320000
* ADMF PRECOMPILER * 00330000
* PL/I MVS/VM (FORMERLY PL/I SAA AD/CYCLE) * 00340000
* MODULE SIZE = 1K * 00350000
* ATTRIBUTES = RE-ENTERABLE * 00360000
* * 00370000
* ENTRY POINT = DSN8EP2 * 00380000
* PURPOSE = SEE FUNCTION * 00390000
* LINKAGE = INVOKED VIA EXEC SQL CALL. * 00400000
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION: * 00410000
* SYMBOLIC LABEL/NAME = INPUTCMD * 00420000
* DESCRIPTION = DB2 COMMAND TO BE PROCESSED BY IFI. * 00430000
* INPUT STATEMENTS FROM THE INPUTCMD * 00440000
* PARAMETER WILL BE PASSED TO THE * 00450000
* TEXT_OR_COMMAND FIELD OF THE OUTPUT_AREA * 00460000
* IN THE DSN8EP1 PROGRAM. * 00470000
* * 00480000
* OUTPUT = PARAMETERS EXPLICITLY RETURNED: * 00490000
* SYMBOLIC LABEL/NAME = IFCA_RET_HEX * 00500000

```

```

* SYMBOLIC LABEL/NAME = IFCA_RES_HEX * 00510000
* SYMBOLIC LABEL/NAME = IFCA_BYTES_MOVED * 00520000
* DESCRIPTION = COMMUNICATION AREAS BETWEEN THE * 00530000
* APPLICATION PROGRAM AND IFI * 00540000
* SYMBOLIC LABEL/NAME = RETURN_BUFF * 00550000
* DESCRIPTION = DB2 COMMAND RESPONSE FROM IFI * 00560000
* THE RETURN CODE, REASON CODE, AND THE * 00570000
* BYTES MOVED FROM THE IFCA AND THE * 00580000
* RTRN_BUFF FIELD FROM THE IFI RETURN AREA * 00590000
* WILL BE PASSED VIA THE IFCA_RET_HEX, * 00600000
* IFCA_RES_HEX, IFCA_BYTES_MOVED, AND * 00610000
* RETURN_BUFF PARAMETERS. * 00620000
* * 00630000
* EXIT NORMAL = * 00640000
* NO ERRORS WERE FOUND IN THE SOURCE AND NO * 00650000
* ERRORS OCCURRED DURING PROCESSING. * 00660000
* * 00670000
* NORMAL MESSAGES = * 00680000
* * 00690000
* EXIT-ERROR = * 00700000
* ERRORS WERE FOUND IN THE SOURCE, OR OCCURRED DURING * 00710000
* PROCESSING. * 00720000
* * 00730000
* RETURN CODE = 4 - WARNING-LEVEL ERRORS DETECTED. * 00740000
* WARNING FOUND DURING EXECUTION. * 00750000
* REASON CODE = NONE OR FROM IFI * 00760000
* * 00770000
* RETURN CODE = 8 - ERRORS DETECTED. * 00780000
* ERROR FOUND DURING EXECUTION. * 00790000
* REASON CODE = NONE OR FROM IFI * 00800000
* * 00810000
* RETURN CODE = 12 - SEVERE ERRORS DETECTED. * 00820000
* UNABLE TO OPEN FILES. * 00830000
* INTERNAL ERROR, ERROR MESSAGE ROUTINE RETURN CODE. * 00840000
* STATEMENT IS TOO LONG. * 00850000
* BUFFER OVERFLOW. * 00860000
* REASON CODE = NONE OR FROM IFI * 00870000
* * 00880000
* ABEND CODES = NONE * 00890000
* * 00900000
* ERROR MESSAGES = * 00910000
* * 00920000
* EXTERNAL REFERENCES = * 00930000
* ROUTINES/SERVICES = NONE * 00940000
* DATA-AREAS = NONE * 00950000
* CONTROL-BLOCKS = NONE * 00960000
* * 00970000
* PSEUDOCODE = * 00980000
* DSN8EP2: PROCEDURE. * 00990000
* GET THE RETURN AREA SIZE FOR COMMAND REQUESTS. * 01000000
* ALLOCATE THE REQUESTED RETURN AREA. * 01010000
* FORMAT THE OUTPUT AREA WITH THE REQUESTED COMMAND. * 01020000
* ISSUE COMMAND REQUEST. * 01030000
* PASS RESULTS TO THE OUTPUT PARAMETERS. * 01040000
* * 01050000
* CHANGE ACTIVITY = * 01060000
* 7/05/95 CHANGED THE OUTPUT STRING LENGTH FROM VARYING * 01070000
* TO FIXED 80 BYTE STRINGS PN72035 @47 KFF0347* 01080000
* 04/17/00 INITIALIZE STORAGE TO PREVENT RETURN CODE=04, * 01090000
* REASON CODE=00E60804 FROM IFI PQ36800* 01100000
* 05/22/03 FIX CODE HOLE CLOSED BY VA AND ENTERPRISE PL/I PQ44916* 01101000
* ******/ 01110000
%PAGE; 01120000
/*******/ 01130000
/* VARIABLE DECLARATIONS */ 01140000
/*******/ 01150000
01160000
DCL COMMAND CHAR(8) INIT(' '); /* USER SPECIFIED DB2 COMMAND */ 01170000
/*******/ 01180000
/* BUILT-IN VARIABLES */ @47*/ 01190000
/*******/ 01200000
01210000
DCL /*@47*/ 01220000
 ADDR BUILTIN, /* ADDRESS OF A DATA AREA @47*/ 01230000
 LENGTH BUILTIN, /* RETURNS LENGTH OF A STRING @47*/ 01240000
 MOD BUILTIN, /* RETURNS MODULO VALUE @47*/ 01250000
 STORAGE BUILTIN, /* FUNCTION TO GET SOME SPACE @47*/ 01260000
 SUBSTR BUILTIN, /* FUNCTION TO RETURN SUBSTRING @47*/ 01270000
 UNSPEC BUILTIN; /* IGNORES VARIABLE TYPING @47*/ 01280000
/*******/ 01290000
/* DECLARATION FOR INPUT AND OUTPUT PARAMETERS */ 01300000
/*******/ 01310000

```

```

DCL 01320000
FUNCTION CHAR(8) INIT(' '), /* FUNC PARM FOR IFI @47*/ 01330000
INPUTCMD CHAR(4096) VARYING, /* DB2 COMMAND @47*/ 01340000
IFCA_RET_HEX FIXED BIN(31), /* IFI RETURN CODE @47*/ 01350000
IFCA_RES_HEX FIXED BIN(31), /* IFI REASON CODE @47*/ 01360000
BUFF_OVERFLOW FIXED BIN(31), /* RETURN BUFF BYTES @47*/ 01370000
 /* RETURNED FROM CALL */ 01380000
NULL_ARRAY(5) FIXED BIN(15), /* INDICATOR ARRAY @47*/ 01390000
RETURN_BUFF CHAR(8320) VARYING, /* PASSED BUFFER @47*/ 01400000
RETURN_LEN FIXED BIN(15) INIT(8320) STATIC; /* LENGTH @47*/ 01410000
 /* OF PASSED BUFFER */ 01420000
 /* 01430000
/***** */
/* WORKING VARIABLES @47*/ 01440000
/***** */
DCL 01450000
REMBYTES FIXED BIN(15) INIT(0), /* NUM BYTES TO BE @47*/ 01460000
 /* PROCESSED IN RTRN AREA*/ 01470000
CMDLEN FIXED BIN(15) INIT(0), /* NUM BYTES IN A @47*/ 01480000
 /* RETURNED CMD STRING */ 01490000
 /* RETURN AREA */ 01500000
01 FIXED_BUFF BASED(ADDR(RETURN_BUFF)), 01510000
02 FIXED_LEN FIXED BIN(15), 01520000
02 FIXED_TEXT CHAR(4160), 01530000
 /* OVERLAY OF PASSED BUF FOR @47*/ 01540000
 /* MOVING DATA FROM RETURN AREA */ 01550000
FILLBYTS FIXED BIN(15) INIT(0), /* NUMBER OF FILL BYTES NEED @47*/ 01560000
 /* TO MAKE RECORD LENGTHS IN */ 01570000
 /* OUTPUT EQUAL TO BUFROWLN */ 01580000
 /* 01590000
NUMFULL FIXED BIN(15) INIT(0), /* NUMBER OF FULL LINES IN @47*/ 01600000
 /* PASSED AREA */ 01610000
 /* LENGTH OF NON-FULL LINE */ 01620000
PARTROW FIXED BIN(15) INIT(0), 01630000
 /* 01640000
J FIXED BIN(15) INIT(0), /* LOOP COUNTER @47*/ 01650000
 /* 01660000
BUFPOSI FIXED BIN(15) INIT(0), /* POSITION IN RETURN BUFFER @47*/ 01670000
 /* 01680000
BUFPOS0 FIXED BIN(15) INIT(0), /* POSITION IN PASSED BUFFER @47*/ 01690000
 /* 01700000
LEN_CHAR CHAR(2) INIT(' '), /* LENGTH BYTES IN COMMAND @47*/ 01710000
 /* RESULT STRING */ 01720000
LEN_BIT BIT(16) INIT('0'B), /* LENGTH IN BITS FOR @47*/ 01730001
 /* CONVERSION */ 01740000
LEN_BIN FIXED BIN(15) INIT(0), /* LENGTH IN BINARY @47*/ 01750000
 /* 01760000
SPACE_LEFT FIXED BIN(15) INIT(0); /* BYTES LEFT IN BUFFER @47*/ 01770000
 /* 01780000
/***** */
/* CONSTANTS @47*/ 01790000
/***** */
DCL 01800000
BLANK CHAR(1) INIT(' ') STATIC, /* BUFFER PADDING @47*/ 01810000
BUFROWLN FIXED BIN(15) INIT(80) STATIC; /* LNGTH OF A LINE @47*/ 01820000
 /* PASSED TO INVOKER */ 01840000
 /* 01850000
/***** */
/* DECLARE IFI CALL MACRO DSNWLI */ 01860000
/***** */
 /* 01870000
 /* 01880000
 /* 01890000
DCL 01900000
DSNWLI ENTRY EXTERNAL OPTIONS(ASM INTER RETCODE);
 /* ENTRY POINT IN LANGUAGE INTERFACE */ 01910000
 /* MODULES TO HANDLE IFC API CALLS. */ 01920000
 /* 01930000
%PAGE;
/***** */
/* IFCA - (INSTRUMENTATION FACILITY COMMUNICATION */ 01940000
/* AREA) CONTAINS INFORMATION REGARDING THE */ 01950000
/* SUCCESS OF THE CALL AND PROVIDES FEEDBACK*/ 01960000
/* INFORMATION TO THE APPLICATION PROGRAM. */ 01970000
/* */ 01980000
/* */ 01990000
/* */ 02000000
/* WARNING: THIS AREA MUST BE MAINTAINED TO INCLUDE*/ 02010000
/* ANY CHANGES TO THE MAPPING MACRO */ 02020000
/* DSNDIFCA */ 02030000
/* */ 02040000
/***** */
 /* 02050000
 /* 02060000
 /* 02070000
DCL 01 IFCA,
02 LNGTH /* LENGTH OF IFCA, INCL LENGTH FIELD*/ 02080000
 /* FIXED BIN(15) INIT(0), */ 02090000
02 UNUSED /* */ 02100000
 /* FIXED BIN(15) INIT(0), */ 02110000
02 EYE_CATCHER /* USED TO VERIFY THE IFCA BLOCK. */ 02120000
 /* 02130000

```

```

 CHAR(4) INIT('IFCA'), 02140000
02 OWNER_ID /* TO ESTAB OWNERSHIP OF AN OPN DEST*/ 02150000
 CHAR(4) INIT(' '), 02160000
02 IFCARC1 /* RETURN CODE FOR THE IFC API CALL.*/ 02170000
 FIXED BIN(31) INIT(0), 02180000
02 IFCARC2 /* REASON CODE FOR THE IFC API CALL.*/ 02190000
 FIXED BIN(31) INIT(0), 02200000
02 BYTES_MOVED /* BYTES OF RECORD WHICH WERE MOVED.*/ 02210000
 FIXED BIN(31) INIT(0), 02220000
02 EXCESS_RECDS /* BYTES OF RECORD WHICH DID NOT FIT*/ 02230000
 FIXED BIN(31) INIT(0), 02240000
02 OPN_WRTSEQ_NUM /* LAST OPN WRTR SEQ# FOR READA FUNC*/ 02250000
 FIXED BIN(31) INIT(0), 02260000
02 NUM_RECDS_LOST /* RECORDS LOST INDICATOR. */ */ 02270000
 FIXED BIN(31) INIT(0), 02280000
02 OPN_NAME_FOR_READA /* OPN NAME USED FOR READA REQUEST */ 02290000
 CHAR(4) INIT(' '), 02300000
02 OPN_NAMES_AREA /* AREA CONTAINING UP TO 8 OPN NAMES*/ 02310000
 03 OPN_LNGTH /* LENGTH OF OPN NAMES RETURNED + 4*/ 02320000
 FIXED BIN(15) INIT(0), 02330000
03 UNUSED_2 02340000
 FIXED BIN(15) INIT(0), 02350000
03 ARRAY_OPN_NAMES(8)/* AREA FOR OPN NAMES RETURNED */ */ 02360000
 CHAR(4) INIT(' '), 02370000
02 TRACE_NOS_AREA /* AREA CONTAINING UP TO 8 TRACE #'S*/ 02380000
 03 TRACE_LNGTH /* LENGTH OF TRACE NO.S RETURNED + 4*/ 02390000
 FIXED BIN(15) INIT(0), 02400000
03 UNUSED_3 02410000
 FIXED BIN(15) INIT(0), 02420000
03 ARRAY_TRACE_NOS(8)/* AREA FOR TRACE NUMBERS RETURNED */ 02430000
 CHAR(2) INIT(' '), 02440000
02 DIAGNOS_AREA /* DIAGNOSTIC AREA. */ */ 02450000
 03 DIAGNOS_LNGTH /* DIAGNOSTIC LENGTH. */ */ 02460000
 FIXED BIN(15) INIT(0), 02470000
03 UNUSED_4 02480000
 FIXED BIN(15) INIT(0), 02490000
03 DIAGNOS_DATA /* DIAGNOSTIC DATA. */ */ 02500000
 CHAR(80) INIT(' '); 02510000
 02520000
DCL 01 OUTPUT_AREA,
02 LNGTH /* LENGTH OF APPL PGM REC TO WRITE */ 02530000
 FIXED BIN(15) INIT(0), 02540000
02 UNUSED
 FIXED BIN(15) INIT(0), 02550000
02 TEXT_OR_COMMAND /* ACTUAL COMMAND OR RECORD TEXT. */ 02560000
 CHAR(254) INIT(' '); 02570000
 02580000
02590000
02600000
DCL 01 RETURN_AREA CTL, /* COMMAND RESULT AREA */ 02610000
02 LNGTH /* NUMBER OF BYTES */ */ 02620000
 FIXED BIN(31), 02630000
02 RTRN_BUFF /* OUTPUT BUFFER */ */ 02640000
 CHAR(*); 02650000
 02660000
/*****GENERAL INITIALIZATION*****
/* GENERAL INITIALIZATION */ 02670000
/*****GENERAL INITIALIZATION*****
/* GENERAL INITIALIZATION */ 02680000
/*****GENERAL INITIALIZATION*****
/* GENERAL INITIALIZATION */ 02690000
 02700000
FUNCTION = 'COMMAND'; /* SET FUNCTION FOR IFI CALL */ 02710000
IFCA.LNGTH = STORAGE(IFCA); /* BYTES USED IN MEMORY */ 02720000
IFCA.EYE_CATCHER = 'IFCA'; /* EYE CATCHER */ */ 02730000
IFCA.OWNER_ID = 'LOC2'; /* DB2 LOCATION 1=LOCAL, 2=REMOTE*/ 02740000
FREE RETURN_AREA; /* FREE STORAGE AND THEN */ 02750000
 /* ALLOCATE STORAGE FOR THE */ 02760000
ALLOCATE 1 RETURN_AREA; /* RETURN AREA */ */ 02770000
2 LNGTH, 02780000
2 RTRN_BUFF CHAR(4096); 02790000
 02800000
RTRN_BUFF = ' '; /* CLEAR THE RETURN BUFFER */ 02810000
RETURN_AREA.LNGTH = 4096; /* LENGTH OF RETURN BUFFER */ 02820000
TEXT_OR_COMMAND=BLANK; /* CLEAR THE DB2 COMMAND AREA*/ 02830000
OUTPUT_AREA.UNUSED = '00000000'B; /* CLEAR THE UNUSED AREA */ 02840000
OUTPUT_AREA.LNGTH = LENGTH(INPUTCMD)+4; /* GET REAL LENGTH OF */ 02850000
OUTPUT_AREA.TEXT_OR_COMMAND = INPUTCMD; /* ACTUAL DB2 COMMAND */ 02860000
 02870000
/*****IFI CALL*****
/* MAKE THE IFI CALL VIA THE DSNWLI MACRO */ 02880000
/*****IFI CALL*****
/* MAKE THE IFI CALL VIA THE DSNWLI MACRO */ 02890000
/*****IFI CALL*****
/* MAKE THE IFI CALL VIA THE DSNWLI MACRO */ 02900000
 02910000
CALL DSNWLI (FUNCTION, IFCA, RETURN_AREA, OUTPUT_AREA); 02920000
 02930000
/*****COPY SELECTED VARIABLES FROM IFI COMMAND RESULTS TO OUTPUT */
/* COPY SELECTED VARIABLES FROM IFI COMMAND RESULTS TO OUTPUT */ 02940000
 02950000

```

```

/* PARAMETER VARIABLES TO PASS TO REQUESTER PROGRAM FOR PROCESSING. */ 02960000
/*** 02970000
 02980000
IFCA_RET_HEX = IFCA.IFCARC1; /* RETURN CODE IN BINARY */ 02990000
IFCA_RES_HEX = IFCA.IFCARC2; /* REASON CODE IN BINARY */ 03000000
BUFF_OVERFLOW = 0; /* PLENTY OF ROOM IN BUFF SO FAR @47*/ 03010000
BUFPOSI = 1; /* INIT POSITION IN RETURN AREA @47*/ 03020000
BUFPOSO = 1; /* INIT POSITION IN PASSED BUFF @47*/ 03030000
/*** 03040000
/* COPY RECORDS FROM THE RETURN AREA TO THE CALLER'S BUFFER. @47*/ 03050000
/* PAD EACH RECORD IN THE CALLER'S BUFFER WITH BLANKS SO ITS @47*/ 03060000
/* LENGTH IS A MULTIPLE OF BUFROWLN. @47*/ 03070000
/*** 03080000
 03090000
IF IFCA.BYTES_MOVED ^= 0 THEN /*@47*/ 03100000
DO; /* IF ANYTHING TO COPY @47*/ 03110000
 DO WHILE (BUFPOSI <= IFCA.BYTES_MOVED - 2); /*@47*/ 03120000
 /* COPY TEXT TO PASSED BUF @47*/ 03130000
 LEN_CHAR = (SUBSTR(RETURN_AREA.RTRN_BUFF,BUFPOSI,2)); /*@47*/ 03140000
 /* GET LENGTH BYTES @47*/ 03150000
 LEN_BIT = UNSPEC(LEN_CHAR); /*@47*/ 03160000
 /* CONVERT TO BIT STRING @47*/ 03170000
 LEN_BIN = LEN_BIT; /*@47*/ 03180000
 /* THEN CONVERT TO BINARY @47*/ 03190000
 /* CALC BYTES LEFT IN PASSED@47*/ 03200000
 LEN_BIN = LEN_BIN - 4; /* TAKE LENGTH BYTES OFF LEN@47*/ 03210000
 SPACE_LEFT = (LEN_BIN / BUFROWLN) * BUFROWLN; /*@47*/ 03220000
 IF MOD(LEN_BIN,BUFROWLN) > 0 THEN /*@47*/ 03230000
 SPACE_LEFT = SPACE_LEFT + BUFROWLN; /*@47*/ 03240000
 IF BUFPOSI + SPACE_LEFT - 1 > RETURN_LEN THEN /*@47*/ 03250000
 BUFF_OVERFLOW = 1; /* INDICATE BUFFER IS FULL @47*/ 03260000
 IF BUFF_OVERFLOW = 1 THEN /*@47*/ 03270000
 LEAVE; /* CAN'T COPY MORE, GET OUT @47*/ 03280000
 BUFPOSI = BUFPOSI + 4; /* MOVE PAST LENGTH BYTES @47*/ 03290000
 IF BUFPOSI + LEN_BIN - 1 > IFCA.BYTES_MOVED THEN /*@47*/ 03300000
 LEAVE; /* AT END OF BUFFER @47*/ 03310000
 NUMFULL = LEN_BIN / BUFROWLN; /* NUMBER OF FULL LINES @47*/ 03320000
 PARTROW = MOD(LEN_BIN,BUFROWLN); /* LENGTH OF PARTIAL LINE @47*/ 03330000
 FILLBYTS = BUFROWLN - PARTROW; /* NUMBER OF PAD BYTES NEED@47*/ 03340000
 IF NUMFULL > 0 THEN /* MOVE ALL COMPLETE LINES @47*/ 03350000
 DO J = 1 TO NUMFULL; /*@47*/ 03360000
 SUBSTR(FIXED_BUFF.FIXED_TEXT,BUFPOSO,BUFROWLN) = /*@47*/ 03370000
 SUBSTR(RETURN_AREA.RTRN_BUFF,BUFPOSI,BUFROWLN); /*@47*/ 03380000
 BUFPOSO = BUFPOSO + BUFROWLN; /* MOVE PAST STRG IN OUTP@47*/ 03390000
 BUFPOSI = BUFPOSI + BUFROWLN; /* MOVE PAST STRG IN INPT@47*/ 03400000
 REMBYTES = REMBYTES - BUFROWLN; /* CALCULATE BYTES LEFT@47*/ 03410000
 END; /*@47*/ 03420000
 IF PARTROW > 0 THEN /*@47*/ 03430000
 DO; /* MOVE PARTIAL LINE @47*/ 03440000
 SUBSTR(FIXED_BUFF.FIXED_TEXT,BUFPOSO,PARTROW) = /*@47*/ 03450000
 SUBSTR(RETURN_AREA.RTRN_BUFF,BUFPOSI,PARTROW); /*@47*/ 03460000
 BUFPOSI = BUFPOSI + PARTROW; /* MOVE PAST STR IN INPUT@47*/ 03470000
 BUFPOSO = BUFPOSO + PARTROW - 1; /* MOVE TO END OF @47*/ 03480000
 /* STRING IN OUTPUT @47*/ 03490000
 SUBSTR(FIXED_BUFF.FIXED_TEXT,BUFPOSO,1) = BLANK; /*@47*/ 03500000
 /* REPLACE THE NEW LINE @47*/ 03510000
 /* CHARACTER IN THE LAST */ 03520000
 /* POSITION WITH A BLANK */ 03530000
 BUFPOSO = BUFPOSO + 1; /* MOVE PAST STRG IN OUTP@47*/ 03540000
 REMBYTES = REMBYTES - PARTROW; /* CALCULATE BYTES LEFT @47*/ 03550000
 END; /*@47*/ 03560000
 IF PARTROW > 0 THEN /* FILL UP SPACE WITH BLK@47*/ 03570000
 DO;
 DO J = BUFPOSO TO (BUFPOSO + FILLBYTS - 1); /*@47*/ 03590000
 SUBSTR(FIXED_BUFF.FIXED_TEXT,J,1) = ' '; /*@47*/ 03600000
 END; /*@47*/ 03610000
 BUFPOSO = BUFPOSO + FILLBYTS; /* MOVE PAST BLANKS @47*/ 03620000
 END; /*@47*/ 03630000
 END; /* COPY TEXT TO PASSED BUF @47*/ 03640000
 FIXED_BUFF.FIXED_LEN = BUFPOSO - 1; /*@47*/ 03650000
 /* GET BYTES IN PASSED BUF @47*/ 03660000
 END; /* IF ANYTHING TO COPY @47*/ 03670000
END DSN8EP2; /* END PROGRAM */ 03680000

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8EPU

PASE SENTENCIAS DE PROGRAMA DE UTILIDAD DE Db2 PARA QUE LAS EJECUTE EL PROGRAMA DE PROCEDIMIENTO ALMACENADO DSNUTILS.

```
DSN8EPU: PROC OPTIONS(MAIN);
/***
 * MODULE NAME = DSN8EPU (SAMPLE PROGRAM) *
 * *
 * DESCRIPTIVE NAME = STORED PROCEDURE REQUESTER PROGRAM *
 * *
 * LICENSED MATERIALS - PROPERTY OF IBM *
 * 5625-DB2 *
 * (C) COPYRIGHT 1992, 2003 IBM CORP. *
 * *
 * STATUS = VERSION 8 *
 * *
 * FUNCTION = *
 * *
 * PASS DB2 UTILITY STATEMENTS TO BE EXECUTED BY THE STORED *
 * PROCEDURE PROGRAM DSNUTILS. GET INPUT FROM 'SYSIN'. *
 * PASS THE STATEMENT AND RECEIVE THE OUTPUT RESULTS *
 * VIA A RETURNED CURSOR. WRITE THE RESULTS TO 'SYSPRINT'. *
 * *
 * DEPENDENCIES = NONE *
 * *
 * RESTRICTIONS = *
 * *
 * INPUT = *
 * *
 * 1. INPUT MUST BE OF THE FORM *
 * *
 * Uid='',Restart='',Utility='REORG TABLESPACE', *
 * CopyDSN1='SYSADM.COPYDDN.DSN8D51A.DSN8S51E', *
 * CopyDEVT1='SYSDA',CopySpace1=10, *
 * Utstmt=
 * ' REORG TABLESPACE DSN8D51A.DSN8S51E
 * SORTDEVT SYSDA SORTNUM 2
 * SHRLEVEL CHANGE
 * DEADLINE 2010-2-4-03.15.00
 * MAPPINGTABLE DSN8510.MAP_TBL
 * MAXRO 240 LONGLOG DRAIN DELAY 900
 * ';
 * *
 * MODULE TYPE = PROCEDURE *
 * PROCESSOR = *
 * ADMF PRECOMPILER *
 * PL/I MVS/VM (FORMERLY PL/I SAA AD/CYCLE) *
 * MODULE SIZE = 2K *
 * ATTRIBUTES = RE-ENTERABLE *
 * *
 * ENTRY POINT = DSN8EPU *
 * PURPOSE = SEE FUNCTION *
 * LINKAGE = STANDARD MVS PROGRAM INVOCATION. *
 * INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION: *
 * SYMBOLIC LABEL/NAME = SYSIN *
 * DESCRIPTION = DDNAME OF SEQUENTIAL DATA SET CONTAINING *
 * DSNUTILS STORED PROCEDURE PARAMETERS. *
 * OUTPUT = PARAMETERS EXPLICITLY RETURNED: *
 * SYMBOLIC LABEL/NAME = SYSPRINT *
 * DESCRIPTION = DDNAME OF SEQUENTIAL OUTPUT DATA SET TO *
 * CONTAIN RESULTS OF THE UTILITIES EXECUTED. *
 * *
 * EXIT NORMAL = *
 * NORMAL MESSAGES = *
 * EXIT-ERROR = *
 * *
 * ABEND CODES = NONE *
 * ERROR MESSAGES = *
 * *
 * EXTERNAL REFERENCES = *
 * ROUTINES/SERVICES = NONE *
 * DATA-AREAS = NONE *
 * CONTROL-BLOCKS = *
 * SQLCA - SQL COMMUNICATION AREA *
 * *
```

```

* PSEUDOCODE =
* DSN8EPU: PROCEDURE.
* DECLARATIONS.
* INITIALIZE VARIABLES.
* GET THE INPUT PARAMETERS AND COPY TO SYSPRINT.
* EXEC SQL CALL SYSPROC.DSNUTILS.
* DO UNTIL SQLCODE > 0.
* EXEC SQL FETCH FROM RESULT SET.
* PRINT RESULT SET TO SYSPRINT.
* END.
*
* NOTICE =
* THIS SAMPLE PROGRAM USES DB2 UTILITIES. SOME UTILITY FUNCTIONS*
* ARE ELEMENTS OF SEPARATELY ORDERABLE PRODUCTS. SUCCESSFUL USE*
* OF A PARTICULAR SAMPLE MAY BE DEPENDENT UPON THE OPTIONAL *
* PRODUCT BEING LICENSED AND INSTALLED IN YOUR ENVIRONMENT. *
*
* CHANGE ACTIVITY =
* PQ24720 - Add FILTRDSN and Fix I/O for seq #ed input *
* PQ44916 - Fix code hole closed by VA and Enterprise PL/I *
* d54292 - Check for unexpected SQLCODE in FETCH loop *
******/ 00980000
DCL SYSPRINT FILE OUTPUT STREAM; 00990000
01000000
DCL SYSIN FILE INPUT STREAM ENV(F RECSIZE(80)); 01010000
01020000
DCL 01 SYSIN_REC,
 05 UTIL_OPTS CHAR(72), 01030000
 05 SEQ_NOS CHAR(08); 01040000
01050000
01060000
DCL SYSIN_EOF BIT(01) INIT('0'B); 01070000
ON ENDFILE(SYSIN)
 SYSIN_EOF = '1'B;
01080000
01090000
01100000
DCL UTIL_OPTS_BUFF VARYING CHAR(32760) INIT(''); 01110000
01120000
DCL ADDR BUILTIN; 01130000
DCL NULL BUILTIN; 01140000
DCL PLIRETC BUILTIN; 01150000
01160000
DCL UID CHAR(16) VARYING; /* UTILITY ID */ 01170000
DCL RESTART CHAR(8) VARYING; /* RESTART */ 01180000
DCL UTSTMT CHAR(32704) VARYING; 01190000
DCL RETCODE FIXED BIN(31); 01200000
DCL UTILITY CHAR(20) VARYING; 01210000
DCL RECDSN CHAR(44) VARYING,
 RECDEVT CHAR(8), 01220000
 RECSpace FIXED BIN(15); 01230000
01240000
DCL DISCDSN CHAR(44) VARYING,
 DISCDEVT CHAR(8), 01250000
 DISCSPACE FIXED BIN(15); 01260000
01270000
DCL PNCHDSN CHAR(44) VARYING,
 PNCHDEVT CHAR(8), 01280000
 PNCHSpace FIXED BIN(15); 01290000
01300000
DCL COPYDSN1 CHAR(44) VARYING,
 COPYDEVT1 CHAR(8), 01310000
 COPYSPACE1 FIXED BIN(15); 01320000
01330000
DCL COPYDSN2 CHAR(44) VARYING,
 COPYDEVT2 CHAR(8), 01340000
 COPYSPACE2 FIXED BIN(15); 01350000
01360000
DCL RCPYDSN1 CHAR(44) VARYING,
 RCPYDEVT1 CHAR(8), 01370000
 RCPYSPACE1 FIXED BIN(15); 01380000
01390000
DCL RCPYDSN2 CHAR(44) VARYING,
 RCPYDEVT2 CHAR(8), 01400000
 RCPYSPACE2 FIXED BIN(15); 01410000
01420000
DCL WORKDSN1 CHAR(44) VARYING,
 WORKDEVT1 CHAR(8), 01430000
 WORKSPACE1 FIXED BIN(15); 01440000
01450000
DCL WORKDSN2 CHAR(44) VARYING,
 WORKDEVT2 CHAR(8), 01460000
 WORKSPACE2 FIXED BIN(15); 01470000
01480000
DCL MAPDSN CHAR(44) VARYING,
 MAPDEVT CHAR(8), 01490000
 MAPSPACE FIXED BIN(15); 01500000
01510000
DCL ERRDSN CHAR(44) VARYING,
 ERRDEVT CHAR(8), 01520000
 ERSPACE FIXED BIN(15); 01530000
01540000
DCL FILTRDSN CHAR(44) VARYING,
 FILTRDEVT CHAR(8), 01550000
 FILTRSPACE FIXED BIN(15); 01560000
01570000

```

```

DCL RESULTS SQL TYPE IS RESULT_SET_LOCATOR VARYING; 01580000
DCL SEQNO FIXED BIN(31); 01590000
DCL TEXT CHAR(122) VARYING; 01600000
EXEC SQL INCLUDE SQLCA; 01610000
Uid=''; 01620000
Restart=''; 01630000
Utstmt=''; 01640000
RetCode = 0; 01650000
Utility=''; 01660000
RecDSN=''; RecDEVT=''; RecSpace=0; 01670000
DiscDSN=''; DiscDEVT=''; DiscSpace=0; 01680000
PnchDSN=''; PnchDEVT=''; PnchSpace=0; 01690000
CopyDSN1=''; CopyDEVT1=''; CopySpace1=0; 01700000
CopyDSN2=''; CopyDEVT2=''; CopySpace2=0; 01710000
RcpyDSN1=''; RcpyDEVT1=''; RcpySpace1=0; 01720000
RcpyDSN2=''; RcpyDEVT2=''; RcpySpace2=0; 01730000
WorkDSN1=''; WorkDEVT1=''; WorkSpace1=0; 01740000
WorkDSN2=''; WorkDEVT2=''; WorkSpace2=0; 01750000
MapDSN=''; MapDEVT=''; MapSpace=0; 01760000
ErrDSN=''; ErrDEVT=''; ErrSpace=0; 01770000
FiltrDSN=''; FiltrDEVT=''; FiltrSPACE=0; 01780000
 01790000
/* Collect DSNUTILS options from SYSIN records, columns 1-72 */ 01800000
GET COPY EDIT(UTIL_OPTS,SEQ_NOS) (A(72),A(8)); 01810000
DO WHILE(^SYSIN_EOF);
 UTIL_OPTS_BUFF = UTIL_OPTS_BUFF || UTIL_OPTS; 01820000
 GET COPY EDIT(UTIL_OPTS,SEQ_NOS)(A(72),A(8)); 01830000
END; /* DO WHILE(^SYSIN_EOF); */ 01840000
 01850000
 01860000
/* Assign DSNUTILS options from inputted settings in UTIL_OPTS_BUFF */ 01870000
GET STRING(UTIL_OPTS_BUFF) DATA; 01880000
 01890000
/* Call DSNUTILS stored procedure to process the inputted settings */ 01900000
EXEC SQL
 CALL SYSPROC.DSNUTILS(:UID, :RESTART,
 :UTSTMT,
 :RETCODE,
 :UTILITY,
 :RECDSN ,:RECDEVT ,:RECSpace ,
 :DISCDSN ,:DISCDEVT ,:DISCSPACE ,
 :PNCHDSN ,:PNCHDEVT ,:PNCHSPACE ,
 :COPYDSN1,:COPYDEVT1,:COPYSPACE1,
 :COPYDSN2,:COPYDEVT2,:COPYSPACE2,
 :RCPYDSN1,:RCPYDEVT1,:RCPYSPACE1,
 :RCPYDSN2,:RCPYDEVT2,:RCPYSPACE2,
 :WORKDSN1,:WORKDEVT1,:WORKSPACE1,
 :WORKDSN2,:WORKDEVT2,:WORKSPACE2,
 :MAPDSN ,:MAPDEVT ,:MAPSPACE ,
 :ERRDSN ,:ERRDEVT ,:ERRSPACE ,
 :FILTRDSN,:FILTRDEVT,:FILTRSPACE);
 01920000
IF SQLCODE < 0 THEN 01930000
 DO;
 PUT SKIP EDIT('CALL SQLCA')(A); 01940000
 PUT SKIP DATA(SQLCA); 01950000
 CALL PLIRETC(8); /* SET PLI RETURN CODE */ 01960000
 RETURN; 01970000
 END; 01980000
 01990000
 02000000
 02010000
 02020000
 02030000
 02040000
 02050000
 02060000
 02070000
IF SQLCODE < 0 THEN 02080000
 DO;
 PUT SKIP EDIT('CALL SQLCA')(A); 02090000
 PUT SKIP DATA(SQLCA); 02100000
 CALL PLIRETC(8); /* SET PLI RETURN CODE */ 02110000
 RETURN; 02120000
 END; 02130000
 02140000
 02150000
EXEC SQL
 ASSOCIATE LOCATOR (:RESULTS) WITH PROCEDURE SYSPROC.DSNUTILS; 02160000
IF SQLCODE < 0 THEN 02170000
 DO;
 PUT SKIP EDIT('ASSOCIATE LOCATOR SQLCA')(A); 02180000
 PUT SKIP DATA(SQLCA); 02190000
 CALL PLIRETC(8); /* SET PLI RETURN CODE */ 02200000
 RETURN; 02210000
 END; 02220000
 02230000
 02240000
 02250000
EXEC SQL
 ALLOCATE SYSPRINT CURSOR FOR RESULT SET :RESULTS; 02260000
IF SQLCODE < 0 THEN 02270000
 DO;
 PUT SKIP EDIT('ALLOCATE SYSPRINT SQLCA')(A); 02280000
 PUT SKIP DATA(SQLCA); 02290000
 CALL PLIRETC(8); /* SET PLI RETURN CODE */ 02300000
 RETURN; 02310000
 END; 02320000
 02330000
 02340000
 02350000
FETCHLOOP:
DO UNTIL(SQLCODE < 0 | SQLCODE = 100);
 EXEC SQL
 FETCH SYSPRINT INTO :SEQNO, :TEXT;
 02360000
 02370001
 02380000
 02390000

```

```

IF (SQLCODE >= 0) 02400000
 & (SQLCODE ^= 100) THEN 02410000
 DO;
 PUT SKIP EDIT(TEXT)(A); 02420000
 END;
 IF (SQLCODE ^= 0) 02430000
 & (SQLCODE ^= 100) THEN 02440000
 DO;
 PUT SKIP EDIT('FETCH SYSPRINT SQLCA')(A); 02450000
 PUT SKIP DATA(SQLCA); 02460000
 END;
 END FETCHLOOP; 02470000
IF SQLCODE < 0 THEN 02480000
DO;
 CALL PLIRETC(8); /* SET PLI RETURN CODE */ 02490000
 RETURN; 02500000
END;

PUT SKIP DATA(RetCode); 02510000
CALL PLIRETC(RetCode); /* SET PLI RETURN CODE */ 02520000
*/ 02530000
END DSN8EPU; 02540000
 02550000
 02560000
 02570000
 02580000
*/ 02590000
 02600000

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8ED1

Pasa mandatos de Db2 recibidos desde la entrada estándar al procedimiento almacenado DSN8ED2 para su ejecución.

```

/***/
/* Module name = DSN8ED1 (sample program) */
/* */
/* DESCRIPTIVE NAME = Stored procedure result set requester pgm */
/* */
/* LICENSED MATERIALS - PROPERTY OF IBM */
/* */
/* 5645-DB2 */
/* (C) COPYRIGHT 1998 IBM CORP. ALL RIGHTS RESERVED. */
/* */
/* STATUS = VERSION 6 */
/* */
/* Function = */
/* */
/* Pass DB2 commands received from standard input to */
/* stored procedure DSN8ED2 for execution. Receive the */
/* command results from DSN8ED2 as result set. Unload the */
/* result set and print the contents to standard output. */
/* */
/* Dependencies = None */
/* */
/* Restrictions = */
/* */
/* 1. DB2 commands must be preceded by a hyphen and */
/* followed by a semicolon. Lines with an asterisk */
/* in the first column or two hyphens as the first */
/* nonblank characters are interpreted as comment */
/* lines. Two hyphens placed after the command text in */
/* in a line indicate that the rest of the line is */
/* comments only. */
/* */
/* 2. A command may be no more than 4096 bytes. */
/* */
/* Input = */
/* 1. A single input parameter that indicates the location */
/* of the stored procedure. The contents must be a */
/* valid DB2 location name of at most 16 characters. */
/* */
/* 2. Lines of length INPUTL from standard input. */
/* Only the first INPUSED bytes are used. Lines are */
/* considered to be either command text or comments. */
/* Command text begins with a hyphen and ends with a */
/* semicolon. Comments begin with an asterisk in */
/* column one or two hyphens as the first two nonblank */
/* characters. Command text may span lines, but comment */
/* text may not. */
/* */
/* Output = */

```

```

/*
 Lines of length OUTLEN to standard output. */
/*
 Each line contains one of the following: */
/*
 a. Command text. */
/*
 b. Results returned by the DB2 for MVS/ESA command */
 processor after the command is issued. */
/*
*/
/*
 Module type = C program */
/*
 Processor = */
/*
 ADMF Precompiler */
/*
 C/370 */
/*
 Module size = See linkedit output */
/*
 Attributes = Not reentrant or reusable */
/*
*/
/*
 Entry point = DSN8ED1 */
/*
 Purpose = See Function */
/*
 Linkage = Standard MVS program invocation, one parameter. */
/*
*/
/*
 Exit normal = */
/*
 Return code 0 on normal completion. */
/*
*/
/*
 Normal messages = */
/*
*/
/*
 *** Input statement: <DB2 command input statement text> */
/*
 *** IFI return area: <Results of DB2 command execution> */
/*
*/
/*
*/
/*
 Exit-error = */
/*
 Return code = 4 - Warnings occurred. */
/*
 - The DB2 for MVS/ESA Instrumentation Facility Interface */
 (IFI) invocation of the DB2 command resulted in a */
 return code 4. The accompanying reason code indicates */
 the specific problem. */
/*
*/
/*
 Return code = 8 - Errors occurred. */
/*
 - The DB2 for MVS/ESA Instrumentation Facility Interface */
 (IFI) invocation of the DB2 command resulted in a */
 return code 8. The accompanying reason code indicates */
 the specific problem. */
/*
*/
/*
 Return code = 12 - Severe errors occurred. */
/*
 - Input parameter 1 did not contain the name of the */
 DB2 server where the stored procedure resides. */
/*
 - The input dataset (SYSIN) did not contain any data. */
/*
 - Command input did not begin with a hyphen. */
/*
 - Command input was not ended with a semicolon. */
/*
 - An input statement contained more than STMTMAX */
 bytes. */
/*
 - Connection to the stored procedure location failed. */
/*
 - The SQL CALL statement to the stored procedure failed. */
/*
 - The DB2 for MVS/ESA Instrumentation Facility Interface */
 (IFI) invocation of the DB2 command resulted in a */
 return code 12. The accompanying reason code indicates */
 the specific problem. */
/*
 - The call to the stored procedure, DSN8ED2, succeeded */
 but DSN8ED2 experienced SQL problems. The formatted */
 SQL error message appears in SYSPRINT. */
/*
 - The call to the stored procedure, DSN8ED2, succeeded */
 but no result set was returned. SYSPRINT messages */
 should provide more information. */
/*
 - A result set was returned by DSN8ED2 but one of the */
 following occurred (see SYSPRINT messages for details): */
/*
 - The locator variable could not be associated with */
 the result set. */
/*
 - The result set cursor could not be allocated. */
/*
 - No data could be fetched from the result set cursor. */
/*
*/
/*
 Abend codes = None */
/*
*/
/*
 Error messages = */
/*
 - *** ERROR: No server name provided - DSN8ED1 ended. */
/*
 - *** ERROR: No input records found - DSN8ED1 ended. */
/*
 - *** ERROR: Syntax for DB2 command is invalid. */
/*
 *** A valid command ends with a semicolon. */
/*
 - *** ERROR: Syntax for DB2 command is invalid. */
/*
 *** A valid command begins with a hyphen. */
/*
 - *** ERROR: Statement length is greater than the ____ */
 character maximum. */
/*
 - *** ERROR: Connection to server <location> was unsuc- */
 cessful. */
/*
 - *** ERROR: Call to stored procedure DSN8ED2 failed; */
/*

```

```

/*
/* diagnostics follow. */
/* - *** ERROR: The following diagnostics were returned by */
/* stored procedure DSN8ED2. */
/* - *** ERROR: DSNTIAR could not format the message. */
/* SQLCODE is ____, SQLERRM is _____ ... */
/* - *** WARNING: Call to stored procedure DSN8ED2 succeeded */
/* but no result set was returned. */
/* - *** WARNING: IFI error codes returned by DSN8ED2. */
/* Return code=___, reason code=___. from */
/* IFI request. */
/* - *** WARNING: ____ records were lost because the IFI */
/* return area in stored procedure DSN8ED2 */
/* is too small to accomodate this request. */
/* ** Increase the IFI return area (RETURN_LEN) */
/* in DSN8ED2 and then recompile/relink/rebind */
/* before resubmitting this request. */
*/
/* - *** Syntax for DB2 command is invalid. */
/* *** A valid command must begin with a hyphen. */
*/
/* - *** Statement length is greater than the <STMTMAX> */
/* character maximum. */
/* - *** Connection to <location> unsuccessful. */
/* *** SQLCODE is <sqlcode>. */
/* - *** Call to DSN8ED2 unsuccessful. */
/* *** SQLCODE is <sqlcode>. */
/* - *** Insufficient space to receive all output from IFI */
/* return area. */
/* - *** Return code=<return code>, reason code=<reason code> */
/* from IFI request. */
/* - *** Severe error occurred. Program is terminating. */
*/
/* External references = */
/* Routines/services = */
/* none */
/* Data areas = */
/* none */
/* Control blocks = */
/* SQLCA - SQL communication area */
*/
/* Pseudocode = */
*/
/* DSN8ED1:
/* - Extract the name of the DB2 server where stored procedure */
/* DSN8ED2 resides from input parameter number 1. */
/* - Call build_DB2_command to create a logical DB2 command record */
/* from one or more records from SYSIN. */
/* - If a command was created successfully, do the following until */
/* all input has been processed or severe errors occur. */
/* - Call connect_to_sp_server to connect to the DB2 server */
/* specified in the first input parameter. */
/* - Call send_DB2_command_to_sp to invoke stored procedure */
/* DSN8ED2 to process the command. */
/* - Call output_results_from_sp to unload the result set */
/* from DSN8ED2 to SYSPRINT. */
/* - Call build_DB2_command to create the next logical DB2 */
/* command record from SYSIN records. */
/* End DSN8ED1 */
*/
/* build_DB2_command:
/* - Read a record from SYSIN */
/* - Do the following until either a full command is built, end */
/* of file is reached, or an error occurs:
/* - if the first byte of the record is '*' or the first two */
/* nonblank bytes of the record are '--' then the whole */
/* record is a comment. Disregard it. */
/* - if '--' is encountered after nonblanks are found then the */
/* rest of the record is a comment and can be disregarded. */
/* - else if a semicolon is found inside a delimited string */
/* call copy_byte_to_cmd_buf to add it to the command string. */
/* - a delimited string is one that starts but has not yet */
/* terminated with a single quote or a double quote */
/* - else if a semicolon is found outside a delimited string */
/* then the command is complete. */
/* - else if a nonblank is found then call copy_byte_to_cmd_buf */
/* to add it to the command string. */
/* - else if a blank is found inside a delimited string then */
/* call copy_byte_to_cmd_buf to add it to the command string. */
/* - else if a blank is found outside a delimited string and */
/* the preceding byte was nonblank then call copy_byte_to_ */
/* cmd_buf to add it to the command string. */
/* - else if a blank is found outside a delimited string and */
*/

```

```

/*
 * the preceding byte was blank then disregard the blank. */
/* - if the input record is exhausted before a terminating */
/* semicolon is found, read the next input record. */
/* - When a command is created successfully, call echo_DB2_command*/
/* to output the reformatted command. */
/* - Check the command to ensure that it starts with a hyphen. */
/* End build_DB2_command */
/*
/* copy_byte_to_cmd_buf */
/* - append the current byte of the input record to the end of */
/* the command string and update length of command string. */
/* - if command string exceeds buffer size, issue a message and */
/* end DSN8ED1. */
/* End copy_byte_to_cmd_buf */
/*
/* echo_DB2_command */
/* - output the reformatted DB2 command to SYSPRINT */
/* End echo_DB2_command */
/*
/* connect_to_sp_server */
/* - invoke SQL to CONNECT to the DB2 server where stored proc- */
/* edure DSN8ED2 resides. */
/* - if the CONNECT fails, issue a message and end DSN8ED1. */
/* End connect_to_sp_server */
/*
/* send_DB2_command_to_sp */
/* - invoke SQL to call stored procedure DSN8ED2 to process the */
/* contents of the command buffer. */
/* - analyze the resultant SQLCODE, IFI return and result codes, */
/* and buffer overload and error parameters returned by DSN8ED2.*/
/* End send_DB2_command_to_sp */
/*
/* output_results_from_sp */
/* - associate a DB2 locator variable with the result set from */
/* stored procedure DSN8ED2. */
/* - allocate a cursor to the result set */
/* - fetch each row from the result set and output it to SYSPRINT */
/* End output_results_from_sp */
/*
/* sql_error */
/* - invoke DSNTIAR to format the current SQL code and print the */
/* messages to SYSPRINT. */
/* - if DSNTIAR cannot detail the code, output the SQLCODE and */
/* SQLERRM to SYSPRINT. */
/* End sql_error */
/*
/* Change activity = */
/* none */
/* **** **** **** **** **** **** **** **** **** **** **** */

/* **** **** C library definitions **** **** */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* **** **** Constants **** **** */
#define INPUTL 81 /* Length of input line */
#define INPUSED 72 /* Bytes used in an input line */
#define LOCLEN 16 /* Length of input parm (loc) */
#define OUTLEN 81 /* Length of output line */
#define RETWRN 4 /* Warning return code */
#define RETERR 8 /* Error return code */
#define RETSEV 12 /* Severe error return code */
#define STMTMAX 4096 /* Maximum statement length */
#define ASTERISK '*' /* Comment indicator */
#define BLANK ' ' /* Blank */
#define HYPHEN '-' /* Hyphen */
#define NULLCHAR '\0' /* Null character */
#define QUOTE '\"' /* Quotation mark */
#define DQUOTE '\"' /* Double quote */
#define SEMICOLON ';' /* SQL stmt terminator */

enum flag {No, Yes}; /* Settings for flags */

/* **** **** Program Argument List **** **** */
char *parms[]; /* Contains input parameter */

/* **** **** Standard Input/Output **** **** */
FILE *sysin; /* Input statements */
char input[INPUTL]; /* Current input data */
char *inires; /* Result of gets invocation */

```

```

FILE *sysprint; /* Command results/error msgs */

/****** Working variables *****/
short int c; /* pointer to command buffer */
enum flag dquotflag; /* '"' delimiter status */
short int i; /* pointer to input buffer */
short int j; /* miscellaneous counter, ptr */
enum flag endstr; /* End of statement flag */
enum flag input_eof; /* End of input data flag */
enum flag quoteflag; /* '"' delimiter status */

/****** DB2 Host Variables *****/
EXEC SQL BEGIN DECLARE SECTION;
char db2loc2[17]; /* Remote DB2 location name */
long int ifca_ret_hex; /* Return code from IFI call */
long int ifca_res_hex; /* Reason code from IFI call */
long int xs_bytes_hex; /* No. of bytes not returned */
long int rc; /* All-purpose return var */

struct {
 short int sp_err_blen; /* Error msg buffer length */
 char sp_err_txt[880]; /* Error msg text */
 } sp_err_buf; /* Error message buffer from */
 /* stored procedure */

short int sperind1; /* Indicator vars for parm 1 */
short int sperind2; /* Indicator vars for parm 2 */
short int sperind3; /* Indicator vars for parm 3 */
short int sperind4; /* Indicator vars for parm 4 */
short int sperind5; /* Indicator vars for parm 5 */

struct {
 short int cmdlen; /* Statement length */
 char cmdtxt[4096]; /* Statement text */
 } cmdbuf; /* Statement buffer passed to */
 /* stored procedure */

 /* Result set locator */

static volatile SQL TYPE IS RESULT_SET_LOCATOR *DSN8ED2_rs_loc;

long int rs_sequence; /* Result set table data sequ */
char rs_data[80]; /* Result set data buffer */
 /* - length is OUTLEN - 1 */

EXEC SQL END DECLARE SECTION;

/****** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

/****** main routine *****/
** main routine
***** int main(int argc, char *argv[]) */proc*/
{
 /* initialize working variables */
 cmdbuf.cmdlen = 0; /* Nothing in command buf yet */
 input_eof = No; /* Not at end of input */
 rc = 0; /* No errors yet */

 sperind1 = -1; /* Clear null indicator var 1 */
 sperind2 = -1; /* Clear null indicator var 2 */
 sperind3 = -1; /* Clear null indicator var 3 */
 sperind4 = -1; /* Clear null indicator var 4 */
 sperind5 = -1; /* Clear null indicator var 5 */

 /* get input parameter (name of server where stored proc resides) */
 for(j=1; j<argc; j++) /* break out the input parms */
 parms[j] = argv[j];

 for(j=0; j<LOCLEN; j++) /* Extract name of DB2 server */
 db2loc2[j] = *(parms[1]+j); /* where sp resides */

 if(db2loc2[1] == BLANK) /* If no server specified, */
 { /* issue error */
 printf(" *** ERROR: No server name provided - DSN8ED1 ended.\n");
 rc = RETSEV;
 }
}

```

```

db2loc2[j]=NULLCHAR; /* Null-terminate the string */

/***
* build the first DB2 command from one or more input records *
***/
if(rc < RETSEV)
{
 build_DB2_command();
 if(input_eof == Yes && rc < RETSEV)
 {
 printf(" *** ERROR: No input records found - DSN8ED1 ended.\n");
 rc = RETSEV;
 }
}

/***
* If a command was built successfully, connect to the DB2 server *
* where the stored procedure resides, send the command to the *
* stored procedure, output the results, and build the next com- *
* mand, if any. *
***/
while(input_eof == No && rc < RETSEV)
{
 connect_to_sp_server(); /* connect to the server */
 if(rc < RETSEV) /* if successful */
 send_DB2_command_to_sp(); /* invoke the stored proc */
 if(rc < RETSEV) /* if successful */
 output_results_from_sp(); /* out the results */
 if(rc < RETSEV) /* if successful */
 build_DB2_command(); /* process the next input */
}

printf("\n\n *** DSN8ED1 completed; highest return code was %d\n",
 rc);

return(rc); /* put return code in ctl blk */

} /* end of main program */

/***
** Build a DB2 command from one or more physical input records **
***/
build_DB2_command() /*proc*/
{
 /* initialize working variables */
 for(i=0; i<INPUTL; i++) /* Blank the input array */
 input[i] = ' ';

 c = 0; /* marks pos'n in command buf */
 cmdbuf.cmdlen = 0; /* no. of bytes in cmd buffer */
 dquotflag = No; /* flags delimiter stat of ''' */
 endstr = No; /* flags end of log inp records */
 i = 0; /* marks pos'n in input buffer */
 quoteflag = No; /* flags delimiter stat of ''' */

 /* read the first physical record of the command from input */
 inres = gets(input);
 if(inres == NULL) /* If end of file reached */
 input_eof = Yes; /* then all finished */

 /* parse the current input record for DB2 command parts */
 while(endstr == No && input_eof == No && rc < RETSEV)
 {
 /* If 1st char in a line is '*' -OR- the 1st two non-blank chars
 * in a line are '--', this is a comment line. Don't copy it to
 * the command buffer; request next line.
 */
 if(i == 0 && input[0] == ASTERISK)

```

```

 i = INPUSED;
else if(cmdbuf.cmdlen == 0
 && input[0] == HYPHEN && input[1] == HYPHEN)
 i = INPUSED;

/***
* Otherwise, this must be a command line. Parse it into the com-
* mand buffer while looking for delimiters and the end of stmt. *
***/
else
 while(i < INPUSED && endstr == No && rc < RETSEV)
 {
 if(input[i] == DQUOTE) /* if double quote found */
 if(dquotflag == Yes) /* and is already a delimiter */
 dquotflag = No; /* note end of delimited str */
 else if(quoteflag == No)/* else if it's not delimited */
 dquotflag = Yes; /* then it's a delimiter */

 if(input[i] == QUOTE) /* if single quote found */
 if(quoteflag == Yes) /* and is already a delimiter */
 quoteflag = No; /* -note end of delimited str */
 else if(dquotflag == No)/* else if it's not delimited */
 quoteflag = Yes; /* then it's a delimiter */

 if(input[i] == HYPHEN /* if '---' found in current */
&& input[i+1] == HYPHEN /* and next byte */
&& i < INPUSED /* not at end of input line */
&& quoteflag == No /* and not in a delimited */
&& dquotflag == No) /* string then rest is comment*/
 i = INPUSED; /* ignore it; rqst next line*/

 else if(input[i] == SEMICOLON/* else if semicolon found */
&& quoteflag == No /* and it's not delimited */
&& dquotflag == No) /* then command is complete */
 endstr = Yes; /* fall through */

 else if(input[i] != BLANK)/* else if non-blank found */
 copy_byte_to_cmd_buf();/* copy it to command buffer */

 else if(input[i] == BLANK /* else if blank found */
&& (quoteflag == Yes /* and it's in a delimited */
|| dquotflag == Yes)) /* string */
 copy_byte_to_cmd_buf();/* copy it to command buffer */

 else if(input[i] == BLANK /* else if blank found */
&& c > 0 /* and something's in cmd buf*/
&& cmdbuf.cmdtxt[c-1]!=BLANK)/* and prev cmd byte nonblank */
 copy_byte_to_cmd_buf();/* copy it to command buffer */

 else; /* swallow all other blanks */

 i++; /* bump pos'n in input record */

 } /* end while(i<INPUSED && endstr == No && rc<RETSEV) */

/***
* if current physical record is exhausted but the current log-
* ical record is still incomplete, get the next physical record *
***/
if(i >= INPUSED && endstr == No && rc < RETSEV)
{
 for(i=0; i<INPUTL; i++) /* Blank the input array */
 input[i] = ' ';
 i = 0; /* reset pointer to input buff*/
 inres = gets(input); /* Read the next physical rec */
 if(inres == NULL) /* If end of file reached */
 {
 /* current logical rec inmplt*/
 input_eof = Yes; /* don't ask for more */
 printf(" *** ERROR: Syntax for DB2 command is invalid.\n");
 printf(" *** A valid command ends with a");
 printf(" semicolon.\n");
 rc = RETSEV; /* stop the program */
 }
}
} /* end while(endstr == No && input_eof == No && rc < RETSEV) */

/***
* display the reformatted command (if one exists)
* echo_DB2_command();
***/
if(cmdbuf.cmdlen > 0)
 echo_DB2_command();

```

```

/***** verify that the command has a valid syntax *****
* verify that the command has a valid syntax *
***** if(endstr == Yes && input_eof == No && rc < RETSEV)
if(cmdbuf.cmdtxt[0] != HYPHEN)
{
 printf(" *** ERROR: Syntax for DB2 command is invalid.\n");
 printf(" *** A valid command begins with a hyphen.\n");
 rc = RETSEV;
}
} /* end of build_DB2_command() */

/***** Copy the current byte of current input line to command buffer *****
** Copy the current byte of current input line to command buffer **
***** copy_byte_to_cmd_buf() /*proc*/
{
 cmdbuf.cmdtxt[c++] = input[i];
 cmdbuf.cmdlen = c;

/***** if entry is too long for command buffer, issue message and quit ***
* if entry is too long for command buffer, issue message and quit *
***** if(cmdbuf.cmdlen >= STMTMAX)
{
 printf(" *** ERROR: Statement length is greater than the");
 printf(" %d character maximum.\n",STMTMAX);
 rc = RETSEV;
}
} /* end of copy_byte_to_cmd_buf() */

/***** Connect to the server where the stored procedure resides *****
** Connect to the server where the stored procedure resides **
***** connect_to_sp_server() /*proc*/
{
 EXEC SQL CONNECT TO :db2loc2;
 if(SQLCODE != 0)
 {
 printf(" *** ERROR: Connection to server %s was unsuccessful.\n",
 db2loc2);
 sql_error(" *** Connection to server unsuccessful");
 rc = RETSEV;
 }
} /* end of connect_to_sp_server() */

/***** Process the current DB2 command built from the input file *****
** Process the current DB2 command built from the input file **
***** send_DB2_command_to_sp() /*proc*/
{
 sperind1 = 0; /* tell DB2 to transmit */
 /* contents of parm 1 */
 EXEC SQL CALL DSN8.DSN8ED2(:cmdbuf :sperind1,
 :ifca_ret_hex :sperind2,
 :ifca_res_hex :sperind3,
 :xs_bytes_hex :sperind4,
 :sp_err_buf :sperind5);

/***** verify the SQL return code returned by the stored procedure *****
* verify the SQL return code returned by the stored procedure *
***** if(SQLCODE == 0)
{
 printf(" *** WARNING: Call to stored procedure DSN8ED2");
 printf(" succeeded\n");
 printf(" but no result set was returned.\n");
 if(rc < RETERR)
 rc = RETERR;
}
else if(SQLCODE == 466)
{
 printf(" *** A result set was returned by stored procedure");
}

```

```

 printf(" DSN8ED2.\n");
}
else
{
 printf(" *** ERROR: Call to stored procedure DSN8ED2 failed;");
 printf(" diagnostics follow.\n");
 sql_error("*** Stored procedure call unsuccessful.");
 rc = RETSEV;
}

/***
* verify the IFI return code returned by the stored procedure *
***/
if(sperind2 != -1 && ifca_ret_hex != 0)
{
 printf(" *** WARNING: IFI error codes returned by DSN8ED2.\n");
 printf(" *** Return code=%0X ", ifca_ret_hex);
 printf(" *** reason code=%0X from IFI request.\n", ifca_res_hex);
 if(ifca_ret_hex > rc)
 rc = ifca_ret_hex;
}

/***
* if IFI return buffer was too small, output a message *
***/
if(sperind4 != -1 && xs_bytes_hex != 0)
{
 printf(" *** WARNING: %d bytes were lost", xs_bytes_hex);
 printf(" because the IFI return area in stored\n");
 printf(" *** procedure DSN8ED2 is too small");
 printf(" to accomodate this request.\n");
 printf(" *** ** Increase the IFI return area");
 printf(" (RETURN_LEN) in DSN8ED2 and then\n");
 printf(" *** recompile/relink/rebind before");
 printf(" resubmitting the request.\n");
 if(rc < RETWRN)
 rc = RETWRN;
}

/***
* output any data from the error message buffer *
***/
if(sperind5 != -1)
{
 printf(" *** ERROR: The following diagnostics were returned by");
 printf(" stored procedure DSN8ED2.\n \n");
 for(j = 0; j < sp_err_buf.sp_err_blen; j++)
 printf("%c",sp_err_buf.sp_err_txt[j]);
 printf("\n");
 if(rc < RETSEV)
 rc = RETSEV;
}

} /* end of send_DB2_command_to_sp() */

/***
** Write out the DB2 command that has been built from input records **
***/
echo_DB2_command()
{
 short int c; /* local ptr to command buffer*/
 short int k,kk,l; /* counters and loop control */
 printf(" \n \n *** Input Statement:\n");

 c = 0;
 /* calculate no. of full print lines the cmd uses and print them */
 kk = cmdbuf.cmdlen / (OUTLEN - 1);
 for(k=1; k<kk; k++)
 {
 printf(" ");
 for(l=0; l<(OUTLEN-1); l++)
 {
 printf("%c",cmdbuf.cmdtxt[c++]);
 }
 printf("\n");
 }
}

```

```

/*********************

* calculate no. of partial print lines the cmd uses; print them *

kk = cmdbuf.cmdlen % (OUTLEN - 1);

if(kk > 0)

{

 printf(" ");

 for(k=1; k<=kk; k++)

 {

 printf("%c",cmdbuf.cmdtxt[c++]);

 }

 printf("\n");

}

} /* end of echo_DB2_command() */

/*********************

** Output the contents of the result set returned by the stored **

** procedure. **

output_results_from_sp() */proc*/

{

 /*****

 * local initialization *

 for(j=0; j<(OUTLEN-1); j++) /* Blank the input array */

 rs_data[j] = BLANK; /* Initialize result string */

 rs_sequence = 0; /* Initialize data sequence */

 printf(" \n \n *** IFI return area:\n");

 /*****

 * associate a locator variable with the result *

 EXEC SQL ASSOCIATE LOCATOR /* Associate the result set */

 (:DSN8ED2_rs_loc) /* locator with a host var. */

 WITH PROCEDURE DSN8.DSN8ED2; /* */

 if (SQLCODE != 0) /* If unsuccessful then */

 { /* - Say so */

 sql_error("*** Associate result set locator call unsuccessful.");

 /* - Print the sqlcode */

 rc = RETSEV; /* - Flush remainder of proc */

 } /* */

 /*****

 * allocate the result set cursor *

 if(rc < RETSEV) /* Or if okay so far then */

 { /* */

 EXEC SQL ALLOCATE DSN8ED2_RS_CSR /* - Allocate a cursor to read*/

 CURSOR FOR /* - Allocate a cursor to read*/

 RESULT SET :DSN8ED2_rs_loc; /* the result set locator */

 if (SQLCODE != 0) /* - If unsuccessful then */

 { /* - Say so */

 sql_error("*** Allocate result set cursor call unsuccessful.");

 /* - Print the sqlcode */

 rc = RETSEV; /* - Flush remainder of proc*/

 } /* */

 } /* */

 /*****

 * fetch first row from the result set *

 if(rc < RETSEV) /* Or if okay so far then */

 {

 EXEC SQL FETCH DSN8ED2_RS_CSR /* - Fetch first row (if any) */

 INTO :rs_sequence, :rs_data; /* from the result set csr */

 if (SQLCODE != 0) /* - If unsuccessful then */

 { /* - Say so */

 sql_error("*** Priming fetch of result set cursor unsuccessful");

 /* - Print the sqlcode */

 rc = RETSEV; /* - Flush remainder of proc*/

 } /* */

 } /* */

 /*****

 * output the contents of the result set *

```

```

while(SQLCODE == 0 && rc < RETSEV) /* Or if okay so far then */
{
 printf("%s\n", rs_data); /* -- Output current line */
 /*
 EXEC SQL FETCH DSN8ED2_RS_CSR /* -- Get the next one from */
 INTO :rs_sequence, :rs_data; /* the result set cursor */
 */
}

/***
* check for successful processing of result set

if (SQLCODE != 100 && rc < RETSEV) /* If unsuccessful then */
{
 /* - Say so */
 sql_error("*** Fetch of result set cursor unsuccessful.");
 /* - Print the sqlcode */
 rc = RETSEV;
 /* - Set return code */
}

} /* end of output_results_from_sp() */

/***
***** SQL error handler

#pragma linkage(dsntiar, OS)

sql_error(char locmsg[]) /*proc*/
{
#define DATA_DIM 10 /* Number of message lines */
 struct error_struct { /* DSNTIAR message structure */
 short int error_len;
 char error_text[DATA_DIM][OUTLEN-1];
 } error_message = {DATA_DIM * (OUTLEN-1)};

 extern short int dsntiar(struct sqlca *sqlca,
 struct error_struct *msg,
 int *len);

 short int rc; /* DSNTIAR Return code */
 int j; /* Loop control */
 static int lrecl = OUTLEN - 1; /* Width of message lines */

 /* print the locator message */
 printf("%.80s\n", locmsg);

 /* format and print the SQL message */
 rc = dsntiar(&sqlca, &error_message, &lrecl);
 if(rc == 0)
 for(j=0; j<=DATA_DIM; j++)
 printf("%.80s\n", error_message.error_text[j]);
 else
 {
 printf(" *** ERROR: DSNTIAR could not format the message\n");
 printf(" *** SQLCODE is %d\n", SQLCODE);
 printf(" *** SQLERRM is \n");
 for(j=0; j<sqlca.sqlerrml; j++)
 printf("%c", sqlca.sqlerrmc[j]);
 printf("\n");
 }

} /* end of sql_error */

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

### DSN8ED2

Utiliza la Interfaz de recurso de instrumentación (IFI) para procesar un mandato de Db2 que se ha pasado desde DSN8ED1, el programa solicitante.

```

/***** Module name = DSN8ED2 (sample program) 00010000
* Module name = DSN8ED2 (sample program) 00020000
* Descriptive name = Stored procedure result set server program 00030000
* LICENSED MATERIALS - PROPERTY OF IBM 00040000
* 5675-DB2 00050000
* (C) COPYRIGHT 1997, 2000 IBM CORP. ALL RIGHTS RESERVED. 00060000
* STATUS = VERSION 7 00070000
* Function: 00080000
* Use the Instrumentation Facility Interface (IFI) to process 00090000
* a DB2 command which has been passed from DSN8ED1, the *
* requester program. Load the responses to a temporary DB2 *
* table and return them as a result set. 00100000
* Notes: 00110000
* Dependencies = 00120000
* 1. Must be linked and run under LE/370 00130000
* 2. Requires global temporary table DSN8.DSN8ED2_RS_TBL 00140000
* (created by sample job DSNTEJ6T) 00150000
* Restrictions = 00160000
* 1. The Instrumentation Facility Communication Area 00170000
* (IFCA) contains information regarding the success 00180000
* of the call and provides feedback. 00190000
* This area must be maintained to include any changes 00200000
* to the mapping macro DSNDIFCA. 00210000
* 2. A command may be no more than 4096 bytes. 00220000
* Module type = C program 00230000
* Processor = 00240000
* ADMF precompiler 00250000
* C/370 00260000
* Module size = See linkedit output 00270000
* Attributes = Not re-entrant nor re-usable 00280000
* Entry Point = CEESTART (LE/370) 00290000
* Purpose = See function 00300000
* Linkage = SIMPLE WITH NULLS 00310000
* Invoked via EXEC SQL call 00320000
* Input = Parameters explicitly passed to this function: 00330000
* Symbolic label/name = ARGV[1] (puts inputcmd) 00340000
* Description = DB2 command to be processed by IFI. 00350000
* Input statements from this parameter 00360000
* will be passed to the text_or_command 00370000
* field of the output_area of the IFI 00380000
* utility for processing. 00390000
* Output = Parameters explicitly returned: 00400000
* Symbolic label/name = ARGV[2] (gets ifca_ret_hex) 00410000
* - IFI return code, in hex 00420000
* Symbolic label/name = ARGV[3] (gets ifca_res_hex) 00430000
* - IFI reason code, in hex 00440000
* Symbolic label/name = ARGV[4] (gets xs_bytes_hex) 00450000
* - Excess bytes not returned, in hex 00460000
* Symbolic label/name = ARGV[5] (gets errmsg_buf) 00470000
* - Formatted SQL error messages 00480000
* Symbolic label/name = ARGV[6] (gets indvar) 00490000
* - DB2 indicator variables 00500000
* Output = Result set returned: 00510000
* Result set cursor name = DSN8ED2_RS_CSR 00520000
* - Formatted responses from IFI for input command 00530000
* Exit normal = 00540000
* No errors were found in the passed DB2 command and no 00550000
* errors occurred during processing. 00560000
* Normal messages = 00570000
* Exit-error = 00580000
* Errors were found in the passed DB2 command or occurred 00590000
* during processing. 00600000
* Return codes: n/a 00610000
* Error messages = see under output 00620000
* 00630000
* 00640000
* 00650000
* 00660000
* 00670000
* 00680000
* 00690000
* 00700000
* 00710000
* 00720000
* 00730000
* 00740000
* 00750000
* 00760000
* 00770000
* 00780000
* 00790000

```

```

* External references = * 00800000
* Routines/services = none * 00810000
* Data areas = none * 00820000
* Control blocks = none * 00830000
* * 00840000
* Pseudocode = * 00850000
* DSN8ED2: Main * 00860000
* - get the passed DB2 command. * 00870000
* - calculate the return area size for command requests. * 00880000
* - allocate the requested return area. * 00890000
* - format the output area with the requested command. * 00900000
* - issue the command request to IFI. * 00910000
* - create the temporary table to hold the result set. * 00920000
* - call sql_error if an unexpected SQLCODE is encountered * 00930000
* - extract the responses from the IFI return buffer and * 00940000
* insert them to the result set table. * 00950000
* - call sql_error if an unexpected SQLCODE is encountered * 00960000
* - open the cursor to the result set table and exit. * 00970000
* - call sql_error if an unexpected SQLCODE is encountered * 00980000
* End DSN8ED2 * 00990000
* * 01000000
* sql_error * 01010000
* - invoke DSNTIAR to format the current SQL code and put the * 01020000
* messages to output parameter ARGV[5]. * 01030000
* - if DSNTIAR cannot detail the code, put the SQLCODE and the * 01040000
* SQLERRM to output parameter ARGV[5]. * 01050000
* End sql_error * 01060000
***** */ 01070000
01080000
01090000
***** C library definitions *****/
#pragma runopts(plist(mvs)) 01100000
#include <stdio.h> 01110000
#include <stdlib.h> 01120000
#include <string.h> 01130000
#pragma linkage(dsnwli,OS) 01140000
01150000
01160000
***** Constants *****/
#define BLANK ' ' /* Buffer padding */ 01170000
#define BUFRWLN 80 /* Length of a report line */ 01180000
#define DATA_DIM 10 /* Number of message lines */ 01190000
#define HYPHEN '-' /* Hyphen */ 01200000
#define LINEFEED '\n' /* Linefeed character */ 01210000
#define NULLCHAR '\0' /* Null character */ 01220000
#define RETSEV 12 /* Severe error return code */ 01230000
#define RETURN_LEN 8320 /* Length of IFI return buffer*/ 01240000
01250000
01260000
***** Program Argument List *****/
struct inp { /* Arg1 (in): Command stmt */ 01270000
 short int incrlen; /* - Input stmt length */ 01280000
 char incmtxt[4096]; /* - Input stmt text */ 01290000
} inputcmd; /* */ 01300000
struct inp *inpptr; /* Pointer to input struct */ 01310000
01320000
01330000
long int ifca_ret_hex; /* Arg2 (out): IFI return code*/ 01340000
long int ifca_res_hex; /* Arg3 (out): IFI reason code*/ 01350000
01360000
long int xs_bytes_hex; /* Arg4 (out): # records lost */ 01370000
01380000
char errmsg[DATA_DIM+1][BUFRWLN]; /* Arg5 (out): error messages */ 01390000
01400000
01410000
short int locind[5]; /* Arg6 (out): indicator vars */ 01420000
01430000
***** Working variables *****/
char *parg1[]; /* Pointer to argument 1 */ 01440000
01450000
int *parg2; /* Pointer to argument 2 */ 01460000
01470000
int *parg3; /* Pointer to argument 3 */ 01480000
01490000
char *parg5[] [BUFRWLN]; /* Pointer to argument 5 */ 01500000
01510000
short int *parg6; /* Pointer to argument 6 */ 01520000
01530000
01540000
01550000
***** DB2 Host Variables *****/
EXEC SQL BEGIN DECLARE SECTION;
long int rs_sequence; /* Result set data sequence */ 01560000
char rs_data[81]; /* Result set data buffer */ 01570000
01580000
/* - length is BUFRWLN+1 for */
/* C NUL-terminator byte) */ 01590000
01600000
01610000
EXEC SQL END DECLARE SECTION;

```

```

***** DB2 SQL Communication Area *****
EXEC SQL INCLUDE SQLCA; 01620000
01630000
01640000
***** DB2 SQL Cursor Declarations *****
EXEC SQL DECLARE DSN8ED2_RS_CSR 01650000
CURSOR WITH RETURN WITH HOLD FOR
SELECT RS_SEQUENCE, RS_DATA 01660000
 FROM DSN8.DSN8ED2_RS_TBL /* <- Created in job DSNTEJ6T */ 01670000
 ORDER BY RS_SEQUENCE; 01680000
 01690000
 01700000
 01710000
***** main routine *****
/*int main(int argc, char *argv[]) /* Argument count and list */
{*/
/****** Constants *****/
char eye[4] /* Const for IFI eye catcher */ 01720000
= {'I','F','C','A'}; 01730000
char loc[4] /* Const for IFI location */ 01740000
= {'L','O','C','2'}; 01750000
 01760000
 01770000
 01780000
/****** Working variables *****/
short int numfull; /* No. of lines in area passed*/ 01790000
 /* from IFI that have BUFROWLN*/ 01800000
 /* or more bytes */ 01810000
short int partrow; /* No. of lines in area passed*/ 01820000
 /* from IFI that have less */ 01830000
 /* than BUFROWLN bytes */ 01840000
 01850000
short int i, j, k; /* Loop control vars */ 01860000
 01870000
 01880000
short int partrow; /* No. of lines in area passed*/ 01890000
 /* from IFI that have less */ 01900000
 /* than BUFROWLN bytes */ 01910000
 01920000
short int i, j, k; /* Loop control vars */ 01930000
 01940000
char *curbyte; /* Pointer to current byte in */ 01950000
 /* return area */ 01960000
 01970000
short int len_bin; /* Length of buffer, in binary*/ 01980000
char lenbyt1; /* 1st byte of length */ 01990000
char lenbyt2; /* 2nd byte of length */ 02000000
 02010000
 02020000
***** IFI Argument List *****
char function[9]; /* First parm for IFI call */ 02030000
 02040000
 02050000
 02060000
***** IFCA - (Instrumentation Facility Communication Area) contains *****
* information regarding the success of the call to IFI and * 02070000
* provides feedback information to the application program. * 02080000
* * 02090000
* * 02100000
* * 02110000
* WARNING: This area must be maintained to include any changes to * 02120000
* the mapping macro DSNDIFCA. * 02130000
***** 02140000
 02150000
typedef struct {
 short int lnghth; /* Second parm for IFI call */ 02160000
 /* Length of the IFCA, */ 02170000
 /* including length field */ 02180000
 short int unused1; /* Reserved */ 02190000
 char eye_catcher[4]; /* Valid eye catcher of IFCA */ 02200000
 /* used to verify IFCA block */ 02210000
 char owner_id[4]; /* Used to establish ownership*/ 02220000
 /* of an OPN destination */ 02230000
 long int ifcarc1; /* Rtrn code for IFC API call */ 02240000
 long int ifcarc2; /* Reason cd for IFC API call */ 02250000
 long int bytes_moved; /* Bytes of recrd rtrnd by IFI*/ 02260000
 long int excess_bytes; /* Bytes that did not fit */ 02270000
 long int opn_writ_seq_num; /* Last OPN writer seqn numbr */ 02280000
 /* rtrnd for a READA function*/ 02290000
 long int num_recds_lost; /* Records lost indicator */ 02300000
 char opn_name_for_reada[4]; /* OPN nm used for READA requ */ 02310000
 struct {
 short int opn_lnghth; /* Area with up to 8 OPN names*/ 02320000
 short int unused2; /* Length+4 of OPN names rtrnd*/ 02330000
 char array_opn_names[4][8]; /* Reserved */ 02340000
 /* Area for OPN names returned*/ 02350000
 } opn_names_area; 02360000
 02370000
 struct {
 short int trace_lnghth; /* Area with up to 8 trace nos*/ 02380000
 short int unused3; /* Length+4 of trace nos rtrnd*/ 02390000
 char array_trace_nos[2][8]; /* Reserved */ 02400000
 /* Area for trace nos returned*/ 02410000
 } trace_nos_area; 02420000
 02430000
}

```

```

struct { /* Diagnosticd area */ 02440000
 short int diagnos_lngth; /* Diagnostics length */ 02450000
 short int unused4; /* Reserved */ 02460000
 char diagnos_data[80]; /* Diagnostics data */ 02470000
} diagnos_area; 02480000
} ifca; **** end IFCA typedef ****/ 02490000
 02500000
ifca *pi; /* Pointer to IFCA structure */ 02510000
 02520000
typedef struct { /* Third pair for IFI call */ 02530000
 short int lngth; /* Length+4 of text or command*/ 02540000
 short int unused; /* Reserved */ 02550000
 char text_or_command[254]; /* Actual cmd or record text */ 02560000
} output_area; 02570000
 02580000
output_area *po; /* Pointer to IFI output area */ 02590000
 02600000
typedef struct { /* Fourth parm for IFI call */ 02610000
 long int lngth; /* Length+4 of IFI return area*/ 02620000
 char rtrn_buff[RETURN_LEN]; /* IFI return area */ 02630000
} return_area; 02640000
 02650000
return_area *pr; /* Pointer to IFI return area */ 02660000
 02670000
/** 02680000
* initialize working variables */ 02690000
**/ 02700000
rc = 0; /* Initialize return code */ 02710000
lastrc = 0; /* Initialize return code */ 02720000
 02730000
for(i=0; i<DATA_DIM+1; i++) /* clear error message buffer */ 02740000
 for(j=0; j<BUFSIZE; j++)
 errormsg[i][j] = BLANK;
 02750000
 02760000
 02770000
/** 02780000
* get input parameter (command for IFI) from caller */ 02790000
**/ 02800000
parg1[1] = argv[1]; /* Command text from caller */ 02810000
curbyte = parg1[1]; /* Get pointer to input struct*/ 02820000
 02830000
/** 02840000
* determine the length of the command text */ 02850000
**/ 02860000
inputcmd.incmlen = 0; /* 02870000
i = 0; 02880000
while(*(curbyte) != NULLCHAR && i < 4096) 02890000
{
 inputcmd.incmtxt[i] = *curbyte;
 i++;
 curbyte++;
 inputcmd.incmlen++;
}
 02900000
 02910000
 02920000
 02930000
 02940000
 02950000
 02960000
/** 02970000
* initialize the IFI parameters */ 02980000
**/ 02990000
strncpy(function,"COMMAND \0",9); /* Set constant */ 03000000
 03010000
pi = malloc(sizeof(ifca)); /* Point to IFCA structure */ 03020000
pi->length = sizeof(ifca); /* Note length of IFCA area */ 03030000
for(i=0; i<4; i++) 03040000
{
 pi->eye_catcher[i] = eye[i]; /* Initialize eye catcher */ 03050000
 pi->owner_id[i] = loc[i]; /* DB2 Loc: 1=Local, 2=Remote */ 03060000
 03070000
 03080000
 03090000
pr = malloc(sizeof(return_area)); /* Point to IFI return area */ 03100000
for(i=0; i<RETURN_LEN; i++) 03110000
 pr->rtrn_buff[i] = BLANK; /* Clear the return buffer */ 03120000
pr->length = RETURN_LEN; /* Length of return buffer */ 03130000
 03140000
po = malloc(sizeof(output_area)); /* Point to IFI command area */ 03150000
po->length = inputcmd.incmlen+4; /* Note length of command text*/ 03160000
for(i=0; i<254; i++) /* Copy in command */ 03170000
 po->text_or_command[i] = inputcmd.incmtxt[i];
 03180000
 03190000
/** 03200000
* make the IFI call via the DSNWLI macro */ 03210000
**/ 03220000
dsnwli(function,pi,pr,po);
 03230000
 03240000
/** 03250000

```

```

* copy IFI command status codes to output parms * 03260000
***** */ 03270000
ifca_ret_hex = pi->ifcarc1; /* IFI Return code in binary */ 03280000
ifca_res_hex = pi->ifcarc2; /* IFI Reason code in binary */ 03290000
xs_bytes_hex = pi->excess_bytes; /* Bytes that did not fit */ 03300000
 03310000
***** */ 03320000
* Extract records from the IFI return area and place them in a * 03330000
* table for transmission to the caller via a result set * 03340000
***** */ 03350000
if(pi->bytes_moved != 0) /* If data was returned by IFI*/ 03360000
{
 **** */ 03370000
 /* First, clear any residue from the result set table * 03390000
***** */ 03400000
 EXEC SQL DELETE
 FROM DSN8.DSN8ED2_RS_TBL;
 03469980
 if(SQLCODE != 0 /* 0 because everything is ok */ 03499970
 & SQLCODE != +88) /* +88 because all rows del'd */ 03529960
 sql_error("*** SQL error when clearing temp table ...");
 03559950
 03620000
 rs_sequence = 0; /* Init result set sequence no*/ 03630000
 for(k=0; k<BUFROWLN; k++) /* Clear result set data var */ 03640000
 rs_data[k] = BLANK;
 03650000
 03660000
***** */ 03670000
* The IFI return buffer contains one or more variable length * 03680000
* records. Each record consists of a 4-byte length component * 03690000
* followed by a text component. The length component contains * 03700000
* the length of the text component plus 4 to account for its * 03710000
* own length. * 03720000
***** */ 03730000
* Extract the length of the 1st record in the buffer and sub- * 03740000
* tract 4 bytes to obtain the length of just the text portion. * 03750000
***** */ 03760000
curbyte = &(pr->rtrn_buff[0]); /* Point to 1st byte in buffer*/ 03770000
lenbyt1 = *(curbyte); /* Set 1st byte of length */ 03780000
lenbyt2 = *(curbyte+1); /* Set 2nd byte of length */ 03790000
 03800000
len_bin = ((short int)lenbyt1) * 10 + ((short int)lenbyt2); 03810000
len_bin = len_bin - 4; /* Discount size of length fld*/ 03820000
 03830000
***** */ 03840000
* For each IFI record returned, create one or more records of * 03850000
* length BUFROWLN and insert them to the result set table * 03860000
***** */ 03870000
while((rc < RETSEV) && (pi->bytes_moved - len_bin) > 2)
{
 curbyte = curbyte + 4; /* Update position in buffer */ 03900000
 03910000
 if(((short int)(curbyte - &(pr->rtrn_buff[0])) + len_bin - 1)
 > pi->bytes_moved)
 break; /* At end of buffer */ 03940000
 03950000
 numfull = len_bin / BUFROWLN; /* No. rows of BUFROWLN bytes */ 03960000
 partrow = len_bin % BUFROWLN; /* No. of bytes leftover */ 03970000
 03980000
***** */ 03990000
* Move all complete lines * 04000000
***** */ 04010000
if(numfull > 0)
{
 for(j=0; j<numfull; j++)
 {
 for(i=0; i<BUFROWLN; i++) /* Clear result set tbl buffer*/ 04050000
 rs_data[i] = BLANK;
 04060000
 for(i=0; i<BUFROWLN; i++)
 {
 rs_data[i] = *curbyte; /* Build result set table rec */ 04090000
 curbyte++; /* Bump ptr into IFI rtrn buff*/ 04100000
 }
 04110000
 04120000
 rs_sequence++; /* Bump result set tbl seqno */ 04130000
 EXEC SQL INSERT /* Insert to the table */ 04140000
 INTO DSN8.DSN8ED2_RS_TBL
 (RS_SEQUENCE,RS_DATA)
 VALUES(:rs_sequence,:rs_data);
 04150000
 04160000
 04170000
 04180000
 04190000
 04200000
 04210000
***** */ 04220000
* Move leftover bytes to one last result set table record * 04230000

```

```

***** ****
if(rc < RETSEV && partrow > 0) 04240000
{
 for(i=0; i<BUFROWLN; i++) /* Clear result set tbl buffer*/ 04250000
 rs_data[i] = BLANK; 04260000
 for(i=0; i<partrow; i++)
 {
 rs_data[i] = *curbyte; /* Build result set table rec */ 04270000
 curbyte++; /* Bump ptr into IFI rtrn buff*/ 04280000
 }
 rs_data[i-1] = BLANK; /* Discard linefeed char */ 04290000
 rs_sequence++; /* Bump result set tbl sequ no*/ 04300000
 EXEC SQL INSERT /* Insert to the table */ 04310000
 INTO DSN8.DSN8ED2_RS_TBL 04320000
 (RS_SEQUENCE,RS_DATA) 04330000
 VALUES(:rs_sequence,:rs_data); 04340000
 if(SQLCODE != 0) 04350000
 sql_error("*** SQL error when inserting partial line ...");04360000
 } /* End-move partial line */ 04370000
 04380000
 04390000
 04400000
 04410000
 04420000
 04430000
 04440000
 04450000
 04460000
 * Advance to next record in the IFI buffer, extract its length,* 04470000
 * and subtract 4 bytes to get length of text portion * 04480000
 **** **** **** **** **** **** **** 04490000
 lenbyt1 = *(curbyte); /* Set 1st byte of length */ 04500000
 lenbyt2 = *(curbyte+1); /* Set 2nd byte of length */ 04510000
 04520000
 len_bin = ((short int)lenbyt1) * 10 + ((short int)lenbyt2); 04530000
 len_bin = len_bin - 4; /* Discount for length field */ 04540000
 04550000
} /* End of copying IFI return text to result set table */ 04560000
 04570000
 04580000
 * Open the cursor to the result set table on the way out * 04590000
 **** **** **** **** **** **** **** 04600000
 if(rc < RETSEV) 04610000
 {
 04620000
 EXEC SQL OPEN DSN8ED2_RS_CSR; 04630000
 if(SQLCODE != 0) 04640000
 sql_error("*** SQL error when opening result set cursor ...");04650000
 } 04660000
 04670000
 } /* End of if data was returned by IFI */ 04680000
 04690000
 04700000
 * Set output arguments and DB2 locator variables * 04710000
 **** **** **** **** **** **** **** 04720000
 par2 = (int *)argv[2]; /* locate and recast 2nd arg */ 04730000
 par2 = ifca_ret_hex; / assign it ifca return cd */ 04740000
 locind[1] = 0; /* tell DB2 to transmit it */ 04750000
 04760000
 par3 = (int *)argv[3]; /* locate and recast 3rd arg */ 04770000
 par3 = ifca_res_hex; / assign it ifca reason cd */ 04780000
 locind[2] = 0; /* tell DB2 to transmit it */ 04790000
 04800000
 par4 = (int *)argv[4]; /* locate and recast 4th arg */ 04810000
 par4 = xs_bytes_hex; / and assign it bytes lost */ 04820000
 locind[3] = 0; /* tell DB2 to transmit it */ 04830000
 04840000
 if(errmsg[0][0] == BLANK) /* if no error message exists*/ 04850000
 locind[4] = -1; /* -tell DB2 not to send one */ 04860000
 else /* otherwise copy it over and*/ 04870000
 {
 /* tell DB2 to transmit it */ 04880000
 par5[0][0] = argv[5]; /* -locate the 5th func arg */ 04890000
 curbyte = par5[0][0]; /* -set helper pointer */ 04900000
 for(i=0; i<DATA_DIM+1; i++) /* -parse a row, looking for */ 04910000
 {
 /* the end of its msg text */ 04920000
 j = 0; 04930000
 while(errmsg[i][j] != NULLCHAR && j < BUFROWLN) 04940000
 {
 curbyte = errmsg[i][j++]; / -copy nonnull bytes */ 04950000
 curbyte++; 04960000
 }
 errmsg[i][j] = LINEFEED; /* -add linefd to end of row */ 04970000
 } /* End of for(i=0; i<DATA_DIM+1; i++) */ 04980000
 04990000
 05000000
 05010000
 curbyte = NULLCHAR; / -null-terminate the buffer*/ 05020000
 locind[4] = 0; /* -tell DB2 to transmit it */ 05030000
 } /* End of if(errmsg[0][0] != BLANK) */ 05040000
 05050000

```

```

parg6 = (short int *)argv[6]; /* locate and recast 6th arg */ 05060000
for(j=0; j<5; j++) /* copy over null-ind array */ 05070000
{
 *parg6 = locind[j];
 parg6++;
} /* return control to caller */ 05110000
 05120000
} /* end of main */ 05130000
 05140000
/*** 05150000
* SQL error handler *
**/ 05160000
/*** 05170000
#pragma linkage(dsntiar, OS) 05180000
 05190000
sql_error(char locmsg[]) 05200000
{
 05210000
 05220000
 /* DSNTIAR message structure */ 05230000
 struct error_struct {
 short int error_len;
 char error_text[DATA_DIM][BUFROWLN];
 } error_message = {DATA_DIM * BUFROWLN};
 05240000
 05250000
 05260000
 05270000
 05280000
extern short int dsntiar(struct sqlca *sqlca,
 struct error_struct *msg,
 int *len);
 05290000
 05300000
 05310000
 05320000
char *curbyte; /* Pointer to current byte in */ 05330000
 /* error_message */ 05340000
 05350000
short int tiar_rc; /* DSNTIAR Return code */ 05360000
int i; /* Loop control */ 05370000
static int lrecl = BUFROWLN; /* Width of message lines */ 05380000
 05390000
/*** 05400000
* indicate that a fatal error has occurred *
**/ 05410000
 05420000
rc = RETSEV; 05430000
 05440000
/*** 05450000
* copy locator message to the error message return buffer *
**/ 05460000
 05470000
strcpy(errmsg[0],locmsg); 05480000
 05490000
/*** 05500000
* format the SQL message and move it to the err msg rtn buffer *
**/ 05510000
 05520000
tiar_rc = dsntiar(&sqlca, &error_message, &lrecl); 05530000
 05540000
if(tiar_rc == 0)
 for(i=0; i<DATA_DIM; i++)
{
 strncpy(errmsg[i+1],error_message.error_text[i],BUFROWLN);
 05550000
 05560000
 05570000
 05580000
 05590000
 05600000
 05610000
}
else
{
 strcpy(errmsg[1],"DSNTIAR could not detail the SQL error");
 strcpy(errmsg[2],"*** SQLCODE is ");
 strcat(errmsg[3],(char *)SQLCODE);
 strcpy(errmsg[4],"*** SQLERRM is ");
 for(i=0; i<sqlca.sqlerrmrl; i++)
 errmsg[5][i],sqlca.sqlerrmc[i];
}
 05620000
 05630000
 05640000
 05650000
 05660000
 05670000
 05680000
 05690000
} /* end of sql_error */ 05700000

```

## Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8EC1

Demuestra cómo un procedimiento almacenado de e Db2 s puede utilizar IMS Open Database Access (ODBA) para conectarse a IMS DBCTL y acceder a IMS datos.

CBL	APOST,LIST,RENT	00000100
	IDENTIFICATION DIVISION.	00000200
	PROGRAM-ID. DSN8EC1	00000300

```

***** DSN8EC1 - DB2 Sample ODBA Stored Procedure *****
*
* Module Name = DSN8EC1
*
* Descriptive Name = DB2 Sample Application
* DB2 Sample ODBA Stored Procedure
* Batch
* Cobol
*
*LICENSED MATERIALS - PROPERTY OF IBM
*5675-DB2
*(C) COPYRIGHT 1999, 2000 IBM CORP. ALL RIGHTS RESERVED.
*
*STATUS = VERSION 7
*
* Function = Demonstrates how a DB2 stored procedure can use
* IMS Open Database Access (ODBA) to connect to
* IMS DBCTL and access IMS data.
*
* In particular, this program allows its client
* to add, retrieve, update, and delete entries in
* the IMS IVP telephone directory database,
* DFSIVD1.
*
*
* Notes = The following conditions must be satisfied:
* (1) DSN8EC1 is registered in DB2 on a server that also
* has an IMS subsystem operating at IMS/ESA V6 or a
* subsequent release (required for ODBA).
* (2) The following IMS IVP parts are available on that IMS
* subsystem:
* (1) DFSIVD1, the IMS IVP telephone directory database
* (2) DFSIVP64, the IMS IVP Cobol PSB for BMP access to
* DFSIVD1
* (3) DSN8EC1 must be run a WLM-established stored proce-
* dures address space only
* (4) The WLM environment associated with DSN8EC1 in SYSIBM.* 00003100
* SYSPROCEDURES is started by a proc that references
* the IMS reslib in both the STEPLIB DD concatenation
* and in the DFSRESLB DD. See the DB2 Installation
* Guide for more information.
*
* Module Type = Cobol Program
* Processor = DB2 for OS/390 precompiler, IBM Cobol
* Module Size = See linkedit output
* Attributes = Re-entrant
*
* Entry Point = DSN8EC1
* Purpose = See function
* Linkage = Standard MVS program invocation
* Input = Parameters explicitly passed to this function: * 00005300
* TDBCTLID PIC X(8)
* - IMS subsystem id
* COMMAND PIC X(8)
* - Action to perform: ADD, UPD, DIS, DEL
* LAST-NAME PIC X(10)
* FIRST-NAME PIC X(10)
* EXTENSION PIC X(10)
* ZIP-CODE PIC X(7)
*
* Output = Parameters explicitly passed by this function * 00006600
* COMMAND PIC X(8)
* - Action performed: ADD, UPD, DIS, DEL
* LAST-NAME PIC X(10)
* FIRST-NAME PIC X(10)
* EXTENSION PIC X(10)
* ZIP-CODE PIC X(7)
* AIBRETRN PIC S9(9) COMP
* - Return code from IMS AIB call
* AIBREASN PIC S9(9) COMP
* - Reason code from IMS AIB call
* ERROR-CALL PIC X(4)
* - DL/I command that failed
*
* Exit-Normal = Return Code 0 Normal Completion
*
* Exit-Error = Return Code 0 Abnormal Completion
*
* Error Messages = None: Errors are signaled by means of
* SQLCODEs and DL/I codes returned to the * 00008400
* * 00008500

```

```

* client. * 00008600
* * 00008700
* External References = * 00008800
* Routines/Services = * 00008900
* AERTDLI - DL/I interface for ODBA * 00009000
* * 00009100
* Data areas = None * 00009200
* * 00009300
* Control Blocks = * 00009400
* AIB - DL/I Application Interface Block * 00009500
* * 00009600
* Tables = None * 00009700
* * 00009800
* * 00009900
* Change Activity = None * 00010000
* * 00010100
* * 00010200
* *Pseudocode*
* PROCEDURE A00000-ODBA-SP
* Call B10000-ALLOCATE-AIB to allocate the IMS AIB * 00010600
* Call B20000-PREPARE-REQUEST to format input from the client* 00010700
* Call B30000-PROCESS-REQUEST to access data on IMS * 00010800
* Call C31000-ADD-ENTRY if client passed ADD request * 00010900
* Call D31100-INSERT-TO-DB to process IMS ISRT * 00011000
* Call C32000-UPDATE-ENTRY if client passed UPD request * 00011100
* Call D32100-GET-HOLD-UNIQUE-FROM-DB for IMS GHU * 00011200
* Call D32200-REPLACE-IN-DB for IMS REPL * 00011300
* Call C33000-DELETE-ENTRY if client passed DEL request * 00011400
* Call D32100-GET-HOLD-UNIQUE-FROM-DB for IMS GHU * 00011500
* Call D33200-DELETE-FROM-DB for IMS DLET * 00011600
* Call C34000-DISPLAY-ENTRY if client passed DIS request* 00011700
* Call D34100-GET-UNIQUE-FROM-DB for IMS GU * 00011800
* Call B40000-DEALLOCATE-AIB to pend unit of work on IMS * 00011900
* *
----- 00012100
----- 00012200
----- 00012300
----- 00012400
----- 00012500
----- 00012600
----- 00012700
----- 00012800
----- 00012900
----- 00013000
----- 00013100
----- 00013200
----- 00013300
----- 00013400
***** * 00013500
* DL/I-related declarations 00013600
***** * 00013700
* Application Interface Block(AIB) mapping 00013800
 01 AIB.
 02 AIBID PIC X(8). 00014000
 02 AIBLEN PIC 9(9) USAGE BINARY. 00014100
 02 AIBSFUNC PIC X(8). 00014200
 02 AIBRSNM1 PIC X(8). 00014300
 02 AIBRSNM2 PIC X(8). 00014400
 02 AIBRESV1 PIC X(8). 00014500
 02 AIBOALEN PIC 9(9) USAGE BINARY. 00014600
 02 AIBOAUSE PIC 9(9) USAGE BINARY. 00014700
 02 AIBRESV2 PIC X(12). 00014800
 02 AIBRETRN PIC 9(9) USAGE BINARY. 00014900
 02 AIBREASN PIC 9(9) USAGE BINARY. 00015000
 02 AIBRESV3 PIC X(4). 00015100
 02 AIBRESA1 USAGE POINTER. 00015200
 02 AIBRESA2 USAGE POINTER. 00015300
 02 AIBRESA3 USAGE POINTER. 00015400
 02 AIBRESV4 PIC X(40). 00015500
 02 AIBSAVE OCCURS 18 TIMES. 00015600
 USAGE POINTER. 00015700
 02 AIBTOKN OCCURS 6 TIMES. 00015800
 USAGE POINTER. 00015900
 02 AIBTOKC PIC X(16). 00016000
 02 AIBTOKV PIC X(16). 00016100
 02 AIBTOKA OCCURS 2 TIMES. 00016200
 PIC 9(9) USAGE BINARY. 00016300
 00016400
* Segment Search Argument (SSA)
 01 SSA.
 02 SEGMENT-NAME PIC X(8) VALUE 'A1111111'. 00016500
 00016600
 00016700

```

```

02 SEG-KEY-NAME PIC X(11) VALUE '(A1111111 ='. 00016800
02 SSA-KEY PIC X(10). 00016900
02 FILLER PIC X VALUE ''). 00017000
00017100
* Initializers 00017200
77 SSA1 PIC X(9) VALUE 'A1111111 '. 00017300
77 APSBNME PIC X(8) VALUE 'DFSIVP6'. 00017400
77 DPCBNME PIC X(8) VALUE 'TELEPCB1'. 00017500
77 VAIBID PIC X(8) VALUE 'DFSAIB '. 00017600
77 SFPREP PIC X(4) VALUE 'PREP'. 00017700
00017800
* DL/I function codes 00017900
77 GET-UNIQUE PIC X(4) VALUE 'GU '. 00018000
77 GET-HOLD-UNIQUE PIC X(4) VALUE 'GHU '. 00018100
77 GET-NEXT PIC X(4) VALUE 'GN '. 00018200
77 ISRT PIC X(4) VALUE 'ISRT'. 00018300
77 DLET PIC X(4) VALUE 'DLET'. 00018400
77 REPL PIC X(4) VALUE 'REPL'. 00018500
77 APSB PIC X(4) VALUE 'APSB'. 00018600
77 DPSB PIC X(4) VALUE 'DPSB'. 00018700
77 APPERR PIC X(3) VALUE '264'. 00018800
77 INVCMD PIC X(3) VALUE '440'. 00018900
77 NOKEY PIC X(3) VALUE '218'. 00019000
00019100
***** 00019200
* I/O area for datacase handling 00019300
***** 00019400
01 IOAREA. 00019500
02 IO-BLANK PIC X(37) VALUE SPACES. 00019600
02 IO-DATA REDEFINES IO-BLANK. 00019700
03 IO-LAST-NAME PIC X(10). 00019800
03 IO-FIRST-NAME PIC X(10). 00019900
03 IO-EXTENSION PIC X(10). 00020000
03 IO-ZIP-CODE PIC X(7). 00020100
02 IO-FILLER PIC X(3) VALUE SPACES. 00020200
02 IO-COMMAND PIC X(8) VALUE SPACES. 00020300
00020400
01 DB2IN-COMMAND. 00020500
02 DB2IW-COMMAND PIC X(8). 00020600
02 DB2TEMP-COMMAND REDEFINES DB2IW-COMMAND. 00020700
03 DB2TEMP-IOCMD PIC X(3). 00020800
03 FILLER PIC X(5). 00020900
00021000
***** 00021100
* Miscellaneous variables 00021200
***** 00021300
77 TEMP-ONE PICTURE X(8) VALUE SPACES. 00021400
77 TEMP-TWO PICTURE X(8) VALUE SPACES. 00021500
77 REPLY PICTURE X(16). 00021600
00021700
01 FLAGS. 00021800
02 SET-DATA-FLAG PIC X VALUE '0'. 00021900
88 NO-SET-DATA VALUE '1'. 00022000
02 TADD-FLAG PIC X VALUE '0'. 00022100
88 PROCESS-TADD VALUE '1'. 00022200
00022300
01 COUNTERS. 00022400
02 L-SPACE-CTR PIC 9(2) COMP VALUE 0. 00022500
00022600
01 RUN-STATUS PIC X(4). 00022700
88 NOT-OKAY VALUE 'BAD'. 00022800
88 OKAY VALUE 'GOOD'. 00022900
00023000
00023100
00023200
00023300
00023400
LINKAGE SECTION. 00023500
***** 00023600
* Data area for DB2 Stored Procedures input/output 00023700
***** 00023800
01 DB2IO-TDBCTLID PIC X(8). 00023900
01 DB2IO-COMMAND PIC X(8). 00024000
01 DB2IO-LAST-NAME PIC X(10). 00024100
01 DB2IO-FIRST-NAME PIC X(10). 00024200
01 DB2IO-EXTENSION PIC X(10). 00024300
01 DB2IO-ZIP-CODE PIC X(7). 00024400
00024500
***** 00024600
* Data area for DB2 Stored Procedures output 00024700
***** 00024800
01 DB2OUT-AIBRETRN PIC S9(9) COMP. 00024900
01 DB2OUT-AIBREASN PIC S9(9) COMP. 00024900

```

```

01 DC-ERROR-CALL PIC X(4). 00025000
 00025100
 00025200
 00025300
***** * Stored Procedure parameter list ***** 00025400
 00025500
PROCEDURE DIVISION 00025600
 USING DB2IO-TDBCTLID, 00025700
 DB2IO-COMMAND, 00025800
 DB2IO-LAST-NAME, 00025900
 DB2IO-FIRST-NAME, 00026000
 DB2IO-EXTENSION, 00026100
 DB2IO-ZIP-CODE, 00026200
 DB2OUT-AIBRETRN, 00026300
 DB2OUT-AIBREASN, 00026400
 DC-ERROR-CALL. 00026500
 00026600
 00026700
 00026800
* Main Driver: Process data passed by client and apply the data 00026900
* to the IMS IVP phone book database, DFSIVD1. 00027000
***** 00027100
A00000-ODBA-SP. 00027200
 MOVE 'GOOD' TO RUN-STATUS. 00027300
 PERFORM B10000-ALLOCATE-AIB. 00027400
 IF OKAY THEN 00027500
 PERFORM B20000-PREPARE-REQUEST. 00027600
 IF OKAY THEN 00027700
 PERFORM B30000-PROCESS-REQUEST. 00027800
 IF OKAY THEN 00027900
 PERFORM B40000-DEALLOCATE-AIB. 00028000
 00028100
 STOP RUN. 00028200
 00028300
 00028400
***** 00028500
* Initialize and allocate the Application Interface Block 00028600
***** 00028700
B10000-ALLOCATE-AIB. 00028800
 INITIALIZE AIB. 00028900
 SET AIBRESA1 TO NULLS. 00029000
 SET AIBRESA2 TO NULLS. 00029100
 SET AIBRESA3 TO NULLS. 00029200
 MOVE ZEROES to AIBRETRN. 00029300
 MOVE ZEROES to AIBREASN. 00029400
 MOVE VAIBID to AIBID. 00029500
 MOVE LENGTH OF AIB to AIBLEN. 00029600
 MOVE SPACES to IOAREA. 00029700
 MOVE LENGTH OF IOAREA to AIBOALEN. 00029800
 MOVE SPACES TO AIBSFUNC. 00029900
 MOVE APSBNME to AIBRSNM1. 00030000
 MOVE DB2IO-TDBCTLID to AIBRSNM2. 00030100
 00030200
* Allocate the PSB for the AIB 00030300
 CALL 'AERTDLI' USING APSB, AIB. 00030400
 00030500
 IF AIBRETRN EQUAL ZEROES THEN 00030600
 MOVE 0 TO SET-DATA-FLAG 00030700
 MOVE 0 TO TADD-FLAG 00030800
 ELSE 00030900
 MOVE 'BAD' TO RUN-STATUS 00031000
 MOVE AIBRETRN TO DB2OUT-AIBRETRN 00031100
 MOVE AIBREASN TO DB2OUT-AIBREASN. 00031200
 00031300
 00031400
***** 00031500
* Prepare data passed from client for processing by ODBA 00031600
***** 00031700
B20000-PREPARE-REQUEST. 00031800
 00031900
* Check the leading space in input command and trim it off 00032000
 INSPECT DB2IO-COMMAND 00032100
 TALLYING L-SPACE-CTR FOR LEADING SPACE 00032200
 REPLACING LEADING SPACE BY '*'. 00032300
 IF L-SPACE-CTR > 0 THEN 00032400
 UNSTRING DB2IO-COMMAND 00032500
 DELIMITED BY ALL '*' 00032600
 INTO TEMP-ONE TEMP-TWO 00032700
 MOVE TEMP-TWO TO DB2IO-COMMAND 00032800
 MOVE 0 TO L-SPACE-CTR 00032900
 MOVE SPACES TO TEMP-TWO. 00033000
 00033100

```

```

* Check the leading space in input LAST NAME and trim it off 00033200
INSPECT DB2IO-LAST-NAME 00033300
 TALLYING L-SPACE-CTR FOR LEADING SPACE 00033400
 REPLACING LEADING SPACE BY '*'. 00033500
IF L-SPACE-CTR > 0 THEN 00033600
 UNSTRING DB2IO-LAST-NAME 00033700
 DELIMITED BY ALL '*' 00033800
 INTO TEMP-ONE TEMP-TWO 00033900
MOVE TEMP-TWO TO DB2IO-LAST-NAME 00034000
MOVE 0 TO L-SPACE-CTR 00034100
MOVE SPACES TO TEMP-TWO. 00034200
 00034300

* Check the leading space in input FIRST NAME and trim it off 00034400
INSPECT DB2IO-FIRST-NAME 00034500
 TALLYING L-SPACE-CTR FOR LEADING SPACE 00034600
 REPLACING LEADING SPACE BY '*'. 00034700
IF L-SPACE-CTR > 0 THEN 00034800
 UNSTRING DB2IO-FIRST-NAME 00034900
 DELIMITED BY ALL '*' 00035000
 INTO TEMP-ONE TEMP-TWO 00035100
MOVE TEMP-TWO TO DB2IO-FIRST-NAME 00035200
MOVE 0 TO L-SPACE-CTR 00035300
MOVE SPACES TO TEMP-TWO. 00035400
 00035500

* Check the leading space in input EXTENSION and trim it off 00035600
INSPECT DB2IO-EXTENSION 00035700
 TALLYING L-SPACE-CTR FOR LEADING SPACE 00035800
 REPLACING LEADING SPACE BY '*'. 00035900
IF L-SPACE-CTR > 0 THEN 00036000
 UNSTRING DB2IO-EXTENSION 00036100
 DELIMITED BY ALL '*' 00036200
 INTO TEMP-ONE TEMP-TWO 00036300
MOVE TEMP-TWO TO DB2IO-EXTENSION 00036400
MOVE 0 TO L-SPACE-CTR 00036500
MOVE SPACES TO TEMP-TWO. 00036600
 00036700

* Check the leading space in input ZIP CODE and trim it off 00036800
INSPECT DB2IO-ZIP-CODE 00036900
 TALLYING L-SPACE-CTR FOR LEADING SPACE 00037000
 REPLACING LEADING SPACE BY '*'. 00037100
IF L-SPACE-CTR > 0 THEN 00037200
 UNSTRING DB2IO-ZIP-CODE 00037300
 DELIMITED BY ALL '*' 00037400
 INTO TEMP-ONE TEMP-TWO 00037500
MOVE TEMP-TWO TO DB2IO-ZIP-CODE 00037600
MOVE 0 TO L-SPACE-CTR 00037700
MOVE SPACES TO TEMP-TWO. 00037800
 00037900

* Move the data to IO area for IMS 00038000
MOVE DB2IO-LAST-NAME TO IO-LAST-NAME. 00038100
MOVE DB2IO-COMMAND TO IO-COMMAND. 00038200
MOVE DB2IO-COMMAND TO DB2IN-COMMAND. 00038300
DISPLAY 'TE>DB2TEMP-IOCMD=' DB2TEMP-IOCMD. 00038400
 00038500

* If no command specified, issue error 00038600
IF IO-COMMAND EQUAL SPACES THEN 00038700
 MOVE 'BAD' TO RUN-STATUS 00038800
 MOVE APPERR TO DB2OUT-AIBRETRN 00038900
 MOVE INVCMD TO DB2OUT-AIBREASN 00039000
 00039100

* If no LAST NAME specified, issue error 00039200
ELSE IF IO-LAST-NAME EQUAL SPACES THEN 00039300
 MOVE 'BAD' TO RUN-STATUS 00039400
 MOVE APPERR TO DB2OUT-AIBRETRN 00039500
 MOVE NOKEY TO DB2OUT-AIBREASN 00039600
 00039700
 00039800
***** 00039900
* Process the request from the client 00040000
***** 00040100
B30000-PROCESS-REQUEST. 00040200
 00040300

* If command is ADD, insert a new record 00040400
IF DB2TEMP-IOCMD EQUAL 'ADD' THEN 00040500
 PERFORM C31000-ADD-ENTRY 00040600
 00040700

* If command is TAD, insert a new record and trace with WTO 00040800
ELSE IF DB2TEMP-IOCMD EQUAL 'TAD' THEN 00040900
 MOVE 1 TO TADD-FLAG 00041000
 PERFORM C31000-ADD-ENTRY 00041100
 00041200

* If command is UPD, update existing record for LAST NAME 00041300

```

```

 ELSE IF DB2TEMP-IOCMD EQUAL 'UPD' THEN 00041400
 PERFORM C32000-UPDATE-ENTRY 00041500
 00041600
* If command is DEL, delete record for LAST NAME 00041700
 ELSE IF DB2TEMP-IOCMD EQUAL 'DEL' THEN 00041800
 PERFORM C33000-DELETE-ENTRY 00041900
 00042000
* If command is DIS, display record for LAST NAME 00042100
 ELSE IF DB2TEMP-IOCMD EQUAL 'DIS' THEN 00042200
 PERFORM C34000-DISPLAY-ENTRY 00042300
 00042400
* Otherwise, issue error for unexpected command 00042500
 ELSE
 MOVE 'BAD' TO RUN-STATUS 00042600
 MOVE APPERR TO DB2OUT-AIBRETRN 00042700
 MOVE INVCMRD TO DB2OUT-AIBREASN. 00042800
 00042900
 00043000
 00043100
***** 00043200
* Deallocate the ODBA Application Interface Block 00043300
***** 00043400
B40000-DEALLOCATE-AIB. 00043500
 MOVE APSBNME to AIBRSNM1. 00043600
* PREP keyword, below, tells IMS to move in-flight transactions 00043700
* to in-doubt state, so checkpoint or rollback can be deferred 00043800
* until DB2 stored procedure client issues COMMIT or ROLLBACK 00043900
 MOVE SFPREP to AIBSFUNC. 00044000
 00044100
* Deallocate the PSB for the AIB 00044200
 CALL 'AERTDLI' USING DPSB, AIB. 00044300
 DISPLAY 'AFTER DPSB PREP, DPCBNME=' DPCBNME. 00044400
 DISPLAY 'DPSB PREP AIBRETRN=' AIBRETRN. 00044500
 DISPLAY 'DPSB PREP AIBREASN=' AIBREASN. 00044600
 DISPLAY 'DPSB PREP AIBRSNM1=' AIBRSNM1. 00044700
 DISPLAY 'DPSB PREP AIBRSNM2=' AIBRSNM2. 00044800
 DISPLAY 'DPSB PREP AIBRESA1=' AIBRESA1. 00044900
 DISPLAY 'DPSB PREP AIBRESA2=' AIBRESA2. 00045000
 DISPLAY 'DPSB PREP AIBRESA3=' AIBRESA3. 00045100
 MOVE AIBRETRN TO DB2OUT-AIBRETRN. 00045200
 MOVE AIBREASN TO DB2OUT-AIBREASN. 00045300
 00045400
 00045500
***** 00045600
* Addition request handler 00045700
***** 00045800
C31000-ADD-ENTRY. 00045900
 MOVE DB2IO-FIRST-NAME TO IO-FIRST-NAME. 00046000
 MOVE DB2IO-EXTENSION TO IO-EXTENSION. 00046100
 MOVE DB2IO-ZIP-CODE TO IO-ZIP-CODE. 00046200
 MOVE IO-COMMAND TO DB2IO-COMMAND. 00046300
 00046400
 IF DB2IO-FIRST-NAME EQUAL SPACES 00046500
 OR DB2IO-EXTENSION EQUAL SPACES 00046600
 OR DB2IO-ZIP-CODE EQUAL SPACES THEN 00046700
 MOVE 'BAD' TO RUN-STATUS 00046800
 MOVE APPERR TO DB2OUT-AIBRETRN 00046900
 MOVE INVCMRD TO DB2OUT-AIBREASN. 00047000
 ELSE 00047100
 PERFORM D31100-INSERT-TO-DB. 00047200
 00047300
 00047400
***** 00047500
* Update request handler 00047600
***** 00047700
C32000-UPDATE-ENTRY. 00047800
 MOVE O TO SET-DATA-FLAG. 00047900
 MOVE IO-LAST-NAME TO SSA-KEY. 00048000
 PERFORM D32100-GET-HOLD-UNIQUE-FROM-DB. 00048100
 IF AIBRETRN = ZEROES THEN 00048200
 IF DB2IO-FIRST-NAME NOT = SPACES THEN 00048300
 MOVE 1 TO SET-DATA-FLAG 00048400
 MOVE DB2IO-FIRST-NAME TO IO-FIRST-NAME 00048500
 END-IF 00048600
 IF DB2IO-EXTENSION NOT = SPACES THEN 00048700
 MOVE 1 TO SET-DATA-FLAG 00048800
 MOVE DB2IO-EXTENSION TO IO-EXTENSION 00048900
 END-IF 00049000
 IF DB2IO-ZIP-CODE NOT = SPACES THEN 00049100
 MOVE 1 TO SET-DATA-FLAG 00049200
 MOVE DB2IO-ZIP-CODE TO IO-ZIP-CODE 00049300
 END-IF 00049400
 MOVE IO-COMMAND TO DB2IO-COMMAND. 00049500

```

```

IF NO-SET-DATA THEN 00049600
 PERFORM D32200-REPLACE-IN-DB 00049700
ELSE 00049800
 MOVE 'BAD' TO RUN-STATUS 00049900
 MOVE APPERR TO DB2OUT-AIBRETRN 00050000
 MOVE INVCMD TO DB2OUT-AIBREASN. 00050100
 00050200
 00050300
***** 00050400
* Delete request handler 00050500
***** 00050600
C33000-DELETE-ENTRY. 00050700
 MOVE IO-LAST-NAME TO SSA-KEY. 00050800
 PERFORM D32100-GET-HOLD-UNIQUE-FROM-DB. 00050900
 IF AIBRETRN = ZEROES THEN 00051000
 MOVE IO-COMMAND TO DB2IO-COMMAND 00051100
 PERFORM D33200-DELETE-FROM-DB. 00051200
 00051300
 00051400
***** 00051500
* Display request handler 00051600
***** 00051700
C34000-DISPLAY-ENTRY. 00051800
 MOVE IO-LAST-NAME TO SSA-KEY. 00051900
 DISPLAY 'TE>SSA-KEY=' SSA-KEY. 00052000
 PERFORM D34100-GET-UNIQUE-FROM-DB. 00052100
 IF AIBRETRN = ZEROES THEN 00052200
 MOVE IO-LAST-NAME TO DB2IO-LAST-NAME 00052300
 MOVE IO-FIRST-NAME TO DB2IO-FIRST-NAME 00052400
 MOVE IO-EXTENSION TO DB2IO-EXTENSION 00052500
 MOVE IO-ZIP-CODE TO DB2IO-ZIP-CODE 00052600
 MOVE IO-COMMAND TO DB2IO-COMMAND 00052700
 00052800
 00052900
***** 00053000
* Data base segment insert request handler 00053100
***** 00053200
D31100-INSERT-TO-DB. 00053300
 MOVE DPCBNME to AIBRSNM1. 00053400
 CALL 'AERTDLI' USING ISRT, AIB, IOAREA, SSA1. 00053500
 IF AIBRETRN = ZEROES THEN 00053600
 IF PROCESS-TADD THEN
 DISPLAY 'INSERT IS DONE, REPLY' UPON CONSOLE 00053700
 ACCEPT REPLY FROM CONSOLE 00053900
 MOVE 0 TO TADD-FLAG 00054000
 END-IF
 ELSE
 MOVE 'BAD' TO RUN-STATUS 00054200
 DISPLAY 'ISRT AIBRETRN=' AIBRETRN 00054300
 DISPLAY 'ISRT AIBREASN=' AIBREASN 00054400
 DISPLAY 'ISRT AIBRESA1=' AIBRESA1 00054500
 DISPLAY 'ISRT AIBRESA2=' AIBRESA2 00054600
 DISPLAY 'ISRT AIBRESA3=' AIBRESA3 00054700
 MOVE APPERR TO DB2OUT-AIBRETRN 00054800
 MOVE INVCMD TO DB2OUT-AIBREASN 00054900
 MOVE ISRT TO DC-ERROR-CALL. 00055000
 00055100
 00055200
 00055300
***** 00055400
* Data base segment request handler 00055500
***** 00055600
D32100-GET-HOLD-UNIQUE-FROM-DB. 00055700
 MOVE DPCBNME to AIBRSNM1. 00055800
 CALL 'AERTDLI' USING GET-HOLD-UNIQUE, AIB, IOAREA, SSA. 00055900
 IF AIBRETRN NOT EQUAL ZEROES THEN 00056000
 MOVE 'BAD' TO RUN-STATUS 00056100
 MOVE APPERR TO DB2OUT-AIBRETRN 00056200
 MOVE INVCMD TO DB2OUT-AIBREASN 00056300
 MOVE GET-HOLD-UNIQUE TO DC-ERROR-CALL. 00056400
 00056500
 00056600
***** 00056700
* Data base segment replace request handler 00056800
***** 00056900
D32200-REPLACE-IN-DB. 00057000
 MOVE DPCBNME to AIBRSNM1. 00057100
 CALL 'AERTDLI' USING REPL, AIB, IOAREA. 00057200
 IF AIBRETRN NOT EQUAL ZEROES THEN 00057300
 MOVE 'BAD' TO RUN-STATUS 00057400
 MOVE APPERR TO DB2OUT-AIBRETRN 00057500
 MOVE INVCMD TO DB2OUT-AIBREASN 00057600
 MOVE REPL TO DC-ERROR-CALL. 00057700

```

```

00057800
00057900

* Data base segment delete request handler 00058000
***** 00058100
***** 00058200
D33200-DELETE-FROM-DB. 00058300
 MOVE DPCBNME to AIBRSNM1. 00058400
 CALL 'AERTDLI' USING DLET, AIB, IOAREA. 00058500
 IF AIBRETRN NOT EQUAL ZEROES THEN 00058600
 MOVE 'BAD' TO RUN-STATUS 00058700
 MOVE APPERR TO DB2OUT-AIBRETRN 00058800
 MOVE INVCMD TO DB2OUT-AIBREASN 00058900
 MOVE DLET TO DC-ERROR-CALL. 00059000
 00059100
 00059200

* Data base segment GET-UNIQUE request handler 00059300
***** 00059400
***** 00059500
D34100-GET-UNIQUE-FROM-DB. 00059600
 MOVE DPCBNME to AIBRSNM1. 00059700
 CALL 'AERTDLI' USING GET-UNIQUE, AIB, IOAREA, SSA. 00059800
 IF AIBRETRN NOT EQUAL ZEROES THEN 00059900
 MOVE 'BAD' TO RUN-STATUS 00060000
 DISPLAY 'GU AIBRETRN=' AIBRETRN 00060100
 DISPLAY 'GU AIBREASN=' AIBREASN 00060200
 DISPLAY 'GU AIBRESA1(ADDR PCB)=' AIBRESA1 00060300
 DISPLAY 'GU AIBRESA2=' AIBRESA2 00060400
 DISPLAY 'GU AIBRESA3=' AIBRESA3 00060500
 MOVE APPERR TO DB2OUT-AIBRETRN 00060600
 MOVE INVCMD TO DB2OUT-AIBREASN 00060700
 MOVE GET-UNIQUE TO DC-ERROR-CALL. 00060800

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8EC2

Muestra cómo CALL al procedimiento almacenado ODBA de ejemplo de Db2, DSN8.

```

IDENTIFICATION DIVISION. 00000100
PROGRAM-ID. DSN8EC2. 00000200
00000300

***** DSN8EC2 - DB2 Sample ODBA Stored Procedure Client ***** 00000400
* * 00000500
* Module Name = DSN8EC2 * 00000600
* * 00000700
* Descriptive Name = DB2 Sample Application * 00000800
* Client for DB2 Sample ODBA Stored Proc * 00000900
* Batch * 00001000
* Cobol * 00001100
*
* 00001200
*LICENSED MATERIALS - PROPERTY OF IBM * 00001300
*5675-DB2 * 00001400
*(C) COPYRIGHT 1999, 2000 IBM CORP. ALL RIGHTS RESERVED. * 00001500
*
*STATUS = VERSION 7 * 00001600
*
* Function = Demonstrates how to CALL the DB2 sample ODBA * 00001700
* stored procedure, DSN8.DSN8EC1, for accessing * 00001800
* the IMS IVP telephone directory database, * 00001900
* DFSIVD1. * 00002000
*
* 00002100
* In particular, this program: * 00002200
* (1) Calls DSN8.DSN8EC1, passing an add request * 00002300
* and the data for an entry to be inserted to * 00002400
* DFSIVD1. * 00002500
* (2) Commits the unit of work for both DB2 and * 00002600
* IMS (Note: IMS work is in an "in doubt" * 00002700
* status until the stored procedure client * 00002800
* performs a COMMIT or a ROLLBACK). * 00002900
* (3) Calls DSN8.DSN8EC1 again, passing a display * 00003000
* request for a entry to be retrieved from * 00003100
* DFSIVD1. * 00003200
*
* 00003300
* Notes = NONE * 00003400
* * 00003500
* * 00003600
* * 00003700
* * 00003800

```

```

* Module Type = Cobol Program * 00003900
* Processor = DB2 for OS/390 precompiler, IBM Cobol * 00004000
* Module Size = See linkedit output * 00004100
* Attributes = Re-entrant * 00004200
* * 00004300
* * 00004400
* Entry Point = DSN8EC2 * 00004500
* Purpose = See function * 00004600
* Linkage = Standard MVS program invocation * 00004700
* * 00004800
* Input = Parameters explicitly passed to this function: * 00004900
* PARMS PIC X(25) * 00005000
* * 00005100
* Output = Symbolic label/Name = SYSOUT * 00005200
* Description = Results of ADD and DIS * 00005300
* * 00005400
* Exit-Normal = Return Code 0 Normal Completion * 00005500
* * 00005600
* Exit-Error = Return Code 8 Abnormal Completion * 00005700
* * 00005800
* Error Messages =
* Unexpected SQLCODE from DSN8.DSN8EC1 during * 00005900
* <command> request. <DSNTIAR detail> * 00006000
* Unexpected return code from ODBA: * 00006100
* - Command <command> * 00006200
* - AIB return code <AIBRETRN> * 00006300
* - AIB reason code <AIBREASN> * 00006400
* - DC error call <DC-ERROR-CALL> * 00006500
* * 00006600
* * 00006700
* External References =
* Routines/Services =
* DSN8EC1 - DB2 sample ODBA stored procedure * 00006800
* DSNTIAR - DB2 SQLCODE message formatter * 00006900
* * 00007000
* Data areas = None * 00007100
* * 00007200
* Control Blocks = SQLCA - SQL communication area * 00007300
* * 00007400
* * 00007500
* * 00007600
* * 00007700
* Tables = None * 00007800
* * 00007900
* * 00008000
* Change Activity = None * 00008100
* * 00008200
* * 00008300
* * 00008400
* * 00008500
* *Pseudocode*
* PROCEDURE A00000-ODBA-SP-CLIENT
* Call A30000-ADD-ENTRY to generate add request * 00008600
* Call C31000-CALL-ODBA-SP to handle add request * 00008700
* Call DSN8.DSN8EC1 to perform add request * 00008800
* Call D31100-CHECK-SQLCODE to verify DB2 call * 00008900
* Call E31110-DETAIL-SQL-ERROR to format err * 00009000
* Call F31111-PRINT-SQL-ERROR-MSG * 00009100
* Call D31200-CHECK-AIBCODE to verify IMS state * 00009200
* Call B40000-COMMIT-WORK to commit DB2 work unit * 00009300
* Call B50000-DISPLAY-ENTRY to generate display request * 00009400
* Call C31000-CALL-ODBA-SP to handle display request * 00009500
* Call DSN8.DSN8EC1 to perform display request * 00009600
* Call D31100-CHECK-SQLCODE to verify DB2 call * 00009700
* Call E31110-DETAIL-SQL-ERROR to format err * 00009800
* Call F31111-PRINT-SQL-ERROR-MSG * 00009900
* Call D31200-CHECK-AIBCODE to verify IMS state * 00010000
* * 00010100
* * 00010200
----- 00010300
* 00010400
* 00010500
* 00010600
* ENVIRONMENT DIVISION. 00010700
* CONFIGURATION SECTION. 00010800
* SOURCE-COMPUTER. IBM-370. 00010900
* OBJECT-COMPUTER. IBM-370. 00011000
* 00011100
* INPUT-OUTPUT SECTION. 00011200
* 00011300
* DATA DIVISION. 00011400
* WORKING-STORAGE SECTION. 00011500
* 00011600
* ****
* Fields for receiving 00011700
* ****
* 01 DB2-SERVER-LOCATION-NAME PIC X(16). 00011800
* ****
* 00011900
* 00012000

```

```

01 IMS-SUBSYSTEM-NAME PIC X(8). 00012100
 00012200
***** * Parameter list for invoking sample DB2 stored procedure DSN8EC1 00012400
***** 00012500
01 DB2IO-TDBCTLID PIC X(8). 00012600
01 DB2IO-COMMAND PIC X(8). 00012700
01 DB2IO-LAST-NAME PIC X(10). 00012800
01 DB2IO-FIRST-NAME PIC X(10). 00012900
01 DB2IO-EXTENSION PIC X(10). 00013000
01 DB2IO-ZIP-CODE PIC X(7). 00013100
01 DB2OUT-AIBRETRN PIC S9(9) COMP. 00013200
01 DB2OUT-AIBREASN PIC S9(9) COMP. 00013300
01 DC-ERROR-CALL PIC X(4). 00013400
 00013500
***** 00013600
* Buffer for receiving SQL error messages 00013700
***** 00013800
01 ERROR-MESSAGE. 00013900
02 ERROR-LEN PIC S9(4) COMP VALUE +960. 00014000
02 ERROR-TEXT PIC X(120) OCCURS 10 TIMES
 INDEXED BY ERROR-INDEX. 00014100
 00014200
77 ERROR-TEXT-LEN PIC S9(9) COMP VALUE +120. 00014300
 00014400
***** 00014500
* Job status indicator 00014600
***** 00014700
01 RUN-STATUS PIC X(4). 00014800
88 NOT-OKAY VALUE 'BAD'. 00014900
88 OKAY VALUE 'GOOD'. 00015000
 00015100
***** 00015200
* Include Cobol standard language global variables 00015300
***** 00015400
EXEC SQL INCLUDE SQLCA END-EXEC. 00015500
 00015600
 00015700
 00015800
LINKAGE SECTION. 00015900
 00016000
***** 00016100
* DSN8EC2 invocation parameter list 00016200
***** 00016300
01 PARMs. 00016400
05 PARMs-LEN PIC 9(4) USAGE BINARY. 00016500
05 PARMs-DATA PIC X(25). 00016600
 00016700
 00016800
 00016900
PROCEDURE DIVISION. 00017000
USING PARMs. 00017100
***** 00017200
* Main driver: Use ODBA to add to and display from the IMS IVP DB 00017300
***** 00017400
A00000-ODBA-SP-CLIENT. 00017500
DISPLAY '*****' 00017600
'*****'.
DISPLAY '* DSN8EC2: Sample Client for IMS/ODBA ' 00017800
'DB2 stored procedure sample (DSN8.DSN8EC1) '. 00017900
DISPLAY '*'. 00018000
MOVE 'GOOD' TO RUN-STATUS. 00018100
 00018200
PERFORM B10000-PROCESS-PARMS. 00018300
 00018400
PERFORM B20000-CONNECT-TO-SERVER. 00018500
 00018600
IF OKAY THEN 00018700
 PERFORM B30000-ADD-ENTRY. 00018800
 00018900
IF OKAY THEN 00019000
 PERFORM B40000-COMMIT-WORK. 00019100
 00019200
IF OKAY THEN 00019300
 PERFORM B50000-DISPLAY-ENTRY. 00019400
 00019500
DISPLAY '*****' 00019600
'*****'.
IF NOT-OKAY THEN 00019800
 MOVE 8 to RETURN-CODE. 00019900
 00020000
 00020100
STOP RUN. 00020200

```

```

B10000-PROCESS-PARMS. 00020300
* Process DSN8EC2 invocation parameters 00020400
*****UNSTRING PARMs-DATA 00020500
 DELIMITED BY SPACE 00020600
 INTO DB2-SERVER-LOCATION-NAME 00020700
 IMS-SUBSYSTEM-NAME. 00020800
 MOVE IMS-SUBSYSTEM-NAME TO DB2IO-TDBCTLID. 00020900
 00021000
 00021100
 00021200
 00021300
 00021400
 00021500
B20000-CONNECT-TO-SERVER. 00021600
* Connect to the remote server 00021700
*****DISPLAY '*' Now connecting to ' DB2-SERVER-LOCATION-NAME. 00022000
DISPLAY '*' for access to IMS node ' 00022100
 IMS-SUBSYSTEM-NAME. 00022200
DISPLAY '*'. 00022300
 00022400
EXEC SQL CONNECT TO :DB2-SERVER-LOCATION-NAME END-EXEC. 00022500
IF SQLCODE IS NOT EQUAL TO ZERO THEN 00022600
 PERFORM D31100-CHECK-SQLCODE. 00022700
 00022800
 00022900
B30000-ADD-ENTRY. 00023000
* Generate and add an entry to the IMS IVP database DFSIVD1 00023100
*****MOVE 'ADD' TO DB2IO-COMMAND. 00023200
MOVE 'DOE' TO DB2IO-LAST-NAME. 00023300
MOVE 'JOHN' TO DB2IO-FIRST-NAME. 00023400
MOVE '9-876-5432' TO DB2IO-EXTENSION. 00023500
MOVE '98765' TO DB2IO-ZIP-CODE. 00023600
MOVE 0 TO DB2OUT-AIBRETRN. 00023700
MOVE 0 TO DB2OUT-AIBREASN. 00023800
MOVE 0 TO DC-ERROR-CALL. 00023900
MOVE ' ' TO DC-ERROR-CALL. 00024000
 00024100
PERFORM C31000-CALL-ODBA-SP. 00024200
 00024300
 00024400
IF OKAY THEN 00024500
 DISPLAY '*' Entry for:' 00024600
 DISPLAY '*' - Last Name ' DB2IO-LAST-NAME 00024700
 DISPLAY '*' - First Name ' DB2IO-FIRST-NAME 00024800
 DISPLAY '*' - Extension Number ... ' DB2IO-EXTENSION 00024900
 DISPLAY '*' - Internal Zip Code .. ' DB2IO-ZIP-CODE 00025000
 DISPLAY '*' added successfully to database DFSIVD1.' 00025100
 DISPLAY '*'. 00025200
 00025300
 00025400
B40000-COMMIT-WORK. 00025500
* Commit changes in the IMS telephone database 00025600
*****EXEC SQL COMMIT END-EXEC. 00025700
 00025800
PERFORM D31100-CHECK-SQLCODE. 00025900
 00026000
 00026100
 00026200
 00026300
 00026400
B50000-DISPLAY-ENTRY. 00026500
* Retrieve an entry from IMS IVP database DFSIVD1 00026600
*****MOVE 'DIS' TO DB2IO-COMMAND. 00026700
MOVE 'LAST1' TO DB2IO-LAST-NAME. 00026800
MOVE 'NNNN' TO DB2IO-FIRST-NAME. 00026900
MOVE 'N-NNN-NNNN' TO DB2IO-EXTENSION. 00027000
MOVE 'NNNNN' TO DB2IO-ZIP-CODE. 00027100
MOVE 0 TO DB2OUT-AIBRETRN. 00027200
MOVE 0 TO DB2OUT-AIBREASN. 00027300
MOVE 0 TO DC-ERROR-CALL. 00027400
MOVE ' ' TO DC-ERROR-CALL. 00027500
 00027600
PERFORM C31000-CALL-ODBA-SP. 00027700
 00027800
 00027900
IF OKAY THEN 00028000
 DISPLAY '*' Entry for:' 00028100
 DISPLAY '*' - Last Name ' DB2IO-LAST-NAME 00028200
 DISPLAY '*' - First Name ' DB2IO-FIRST-NAME 00028300
 DISPLAY '*' - Extension Number ... ' DB2IO-EXTENSION 00028400

```

```

DISPLAY '*' - Internal Zip Code .. ' DB2IO-ZIP-CODE 00028500
DISPLAY '*' retrieved successfully from DFSIVD1.' 00028600
DISPLAY '*'. 00028700
 00028800
 00028900
 00029000
C31000-CALL-ODBA-SP. 00029100
***** * Invoke the sample stored procedure for IMS/ODBA 00029200
***** * EXEC SQL CALL DSN8.DSN8EC1 (:DB2IO-TDBCTLID, 00029300
 :DB2IO-COMMAND, 00029400
 :DB2IO-LAST-NAME, 00029500
 :DB2IO-FIRST-NAME, 00029600
 :DB2IO-EXTENSION, 00029700
 :DB2IO-ZIP-CODE, 00029800
 :DB2OUT-AIBRETRN, 00029900
 :DB2OUT-AIBREASN, 00030000
 :DC-ERROR-CALL) 00030100
 00030200
END-EXEC. 00030300
 00030400
PERFORM D31100-CHECK-SQLCODE. 00030500
 00030600
IF OKAY THEN 00030700
 PERFORM D31200-CHECK-AIBCODE. 00030800
 00030900
 00031000
D31100-CHECK-SQLCODE. 00031100
***** * Verify that the prior SQL call completed successfully 00031200
***** * IF SQLCODE NOT = 0 THEN 00031300
 MOVE 'BAD' TO RUN-STATUS 00031400
 DISPLAY '*' Unexpected SQLCODE from DSN8.DSN8EC1 ' 00031500
 'during ' DB2IO-COMMAND ' request.' 00031600
 DISPLAY '*' 00031700
 PERFORM E31110-DETAIL-SQL-ERROR. 00031800
 00031900
 00032000
 00032100
 00032200
E31110-DETAIL-SQL-ERROR. 00032300
***** * Call DSNTIAR to return a text message for an unexpected 00032400
* SQLCODE. 00032500
***** * CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN. 00032600
***** * IF RETURN-CODE = ZERO 00032700
 PERFORM F31111-PRINT-SQL-ERROR-MSG VARYING ERROR-INDEX 00032800
 FROM 1 BY 1 UNTIL ERROR-INDEX GREATER THAN 10. 00032900
 00033000
 00033100
 00033200
* **MESSAGE FORMAT 00033300
* **ROUTINE ERROR 00033400
* **PRINT ERROR MESSAG 00033500
 00033600
 00033700
F31111-PRINT-SQL-ERROR-MSG. 00033800
***** * Print message text 00033900
***** * DISPLAY ERROR-TEXT (ERROR-INDEX). 00034000
 00034100
 00034200
 00034300
 00034400
D31200-CHECK-AIBCODE. 00034500
***** * Verify that the IMS operation via ODBA succeeded 00034600
***** * IF DB2OUT-AIBRETRN NOT = 0 OR DB2OUT-AIBREASN NOT = 0 THEN 00034700
 MOVE 'BAD' TO RUN-STATUS 00034800
 DISPLAY '*' Unexpected return code from ODBA:' 00034900
 DISPLAY '*' - Command' DB2IO-COMMAND 00035000
 DISPLAY '*' - AIB return code' DB2OUT-AIBRETRN 00035100
 DISPLAY '*' - AIB reason code' DB2OUT-AIBREASN 00035200
 DISPLAY '*' - DC error call' DC-ERROR-CALL 00035300
 DISPLAY '*' 00035400
 00035500
 00035600

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8ES1

Acepta un número de departamento del invocador y devuelve parámetros que contienen las ganancias totales (salarios y bonificaciones) para empleados de ese departamento, así como el número de empleados que obtienen una bonificación.

```
-- DSN8ES1: SOURCE MODULE FOR THE SAMPLE SQL PROCEDURE 00010000
-- 00020000
-- LICENSED MATERIALS - PROPERTY OF IBM 00030000
-- 5635-DB2 00040000
-- (C) COPYRIGHT 2000, 2006 IBM CORP. ALL RIGHTS RESERVED. 00050000
-- 00060000
-- STATUS = VERSION 9 00070000
-- 00080000
-- Function: Accepts a department number from the caller and returns 00090000
-- parameters containing the total earnings (salaries and 00100000
-- bonuses) for employees in that department, as well as the 00110000
-- number of employees who got a bonus. 00120000
-- 00130000
-- In addition, DSN8ES1 generates a result set that contains 00140000
-- the serial no, first and last name, salary, and bonus for 00150000
-- each employee in the department who got a bonus. The 00160000
-- result set also contains a sequence number so that it can 00170000
-- be read in the order it was generated. 00180000
-- 00190000
-- Notes: 00200000
-- Dependencies: 00210000
-- - Requires DB2 precompiler support for SQL procedures (DSNHPSM) 00220000
-- - Requires a global temporary table (created in sample job 00230000
-- DSNTEJ63) for returning the result. 00240000
-- 00250000
-- Restrictions: 00260000
-- 00270000
-- Module Type: SQL Procedure 00280000
-- Processor: DB2 for OS/390 precompiler and IBM C/C++ for OS/390 00290000
-- or a subsequent release 00300000
-- Attributes: Re-entrant and re-usable 00310000
-- 00320000
-- Entry Point: DSN8ES1 00330000
-- Purpose: See Function, above 00340000
-- 00350000
-- Parameters: 00360000
-- - Input: DEPTNO CHAR(3) 00370000
-- - Output: DEPTSAL DECIMAL(15,2) 00380000
-- BONUSCNT INTEGER 00390000
-- 00400000
-- Normal Exit: 00410000
-- Error Exit: 00420000
-- 00430000
-- 00440000
-- External References: 00450000
-- - EMP : DB2 Sample Employee Table 00460000
-- - DSN8.DSN8ES1_RS_TBL: Global Temporary Table for result set 00470000
-- 00480000
-- Pseudocode: 00490000
-- - Clear any residual from result set table 00500000
-- - Open cursor on EMP table for employees in department DEPTNO 00510000
-- - While more rows:
-- - Add current employee's salary and bonus to total department 00520000
-- earnings
-- - If current employee's bonus is greater than zero 00530000
-- - increment the department bonus counter 00540000
-- - add the employee's serial, first and last name, salary and 00550000
-- bonus to the result set table, using the bonus counter as 00560000
-- a result set sequence number 00570000
-- - If no errors, open the cursor to the result set 00580000
-- 00590000
-- CREATE PROCEDURE DSN8.DSN8ES1 00600000
-- (IN DEPTNO CHAR(3), 00610000
-- OUT DEPTSAL DECIMAL(15,2), 00620000
-- OUT BONUSCNT INT) 00630000
-- PARAMETER CCSID EBCDIC 00640000
-- FENCED
-- RESULT SET 1 00650000
-- LANGUAGE SQL 00660000
-- NOT DETERMINISTIC 00670000
-- MODIFIES SQL DATA 00680000
-- COLLID DSN8ES!! 00690000
-- 00700000
-- 00710000
-- 00720000
-- 00730000
-- 00740000
```

```

WLM ENVIRONMENT WLMENV 00750000
 ASUTIME NO LIMIT 00760000
 COMMIT ON RETURN NO 00800000
 00810000
P1: BEGIN NOT ATOMIC 00820000
 DECLARE EMPLOYEE_NUMBER CHAR(6) CCSID EBCDIC; 00830000
 DECLARE EMPLOYEE_FIRSTNAME CHAR(12) CCSID EBCDIC; 00840000
 DECLARE EMPLOYEE_LASTNAME CHAR(15) CCSID EBCDIC; 00850000
 DECLARE EMPLOYEE_SALARY DECIMAL(9,2) DEFAULT 0; 00860000
 DECLARE EMPLOYEE_BONUS DECIMAL(9,2) DEFAULT 0; 00870000
 DECLARE TOTAL_SALARY DECIMAL(15,2) DEFAULT 0; 00880000
 DECLARE BONUS_COUNTER INT DEFAULT 0; 00890000
 DECLARE END_TABLE INT DEFAULT 0; 00900000
 00910000
 -- Cursor for result set of employees who got a bonus 00920000
 DECLARE DSN8ES1_RS_CSR CURSOR WITH RETURN WITH HOLD FOR 00930000
 SELECT RS_SEQUENCE, 00940000
 RS_EMPNO, 00950000
 RS_FIRSTNAME, 00960000
 RS_LASTNAME, 00970000
 RS_SALARY, 00980000
 RS_BONUS 00990000
 FROM DSN8.DSN8ES1_RS_TBL 01000000
 ORDER BY RS_SEQUENCE; 01010000
 01020000
 -- Cursor to fetch department employees 01030000
 DECLARE C1 CURSOR FOR 01040000
 SELECT EMPNO, 01050000
 FIRSTNAME, 01060000
 LASTNAME, 01070000
 SALARY, 01080000
 BONUS 01090000
 FROM EMP 01100000
 WHERE WORKDEPT = DEPTNO; 01110000
 01120000
 DECLARE CONTINUE HANDLER FOR NOT FOUND 01130000
 SET END_TABLE = 1; 01140000
 01150000
 DECLARE EXIT HANDLER FOR SQLEXCEPTION 01160000
 SET DEPTSAL = NULL; 01170000
 01180000
 -- Clean residual from the result set table 01190000
 DELETE FROM DSN8.DSN8ES1_RS_TBL; 01200000
 01210000
 OPEN C1; 01220000
 01230000
 FETCH C1 01240000
 INTO EMPLOYEE_NUMBER,
 EMPLOYEE_FIRSTNAME,
 EMPLOYEE_LASTNAME,
 EMPLOYEE_SALARY,
 EMPLOYEE_BONUS; 01250000
 01260000
 -- Process each employee in the department 01270000
 WHILE END_TABLE = 0 DO 01280000
 -- Update department total salary 01290000
 SET TOTAL_SALARY = TOTAL_SALARY
 + EMPLOYEE_SALARY
 + EMPLOYEE_BONUS; 01300000
 01310000
 -- If the current employee received a bonus 01320000
 IF EMPLOYEE_BONUS > 0.00 THEN
 -- Update department bonus count 01330000
 SET BONUS_COUNTER = BONUS_COUNTER + 1; 01340000
 01350000
 -- Add the employee's data to the result set 01360000
 INSERT INTO DSN8.DSN8ES1_RS_TBL
 (RS_SEQUENCE,
 RS_EMPNO,
 RS_FIRSTNAME,
 RS_LASTNAME,
 RS_SALARY,
 RS_BONUS) 01370000
 VALUES(P1.BONUS_COUNTER,
 P1.EMPLOYEE_NUMBER,
 P1.EMPLOYEE_FIRSTNAME,
 P1.EMPLOYEE_LASTNAME,
 P1.EMPLOYEE_SALARY,
 P1.EMPLOYEE_BONUS); 01380000
 01390000
 END IF; 01400000
 01410000
 01420000
 01430000
 01440000
 01450000
 01460000
 01470000
 01480000
 01490000
 01500000
 01510000
 01520000
 01530000
 01540000
 01550000
 01560000
 01570000
 01580000
 01590000
 FETCH C1

```

```

 INTO EMPLOYEE_NUMBER, 01600000
 EMPLOYEE_FIRSTNAME, 01610000
 EMPLOYEE_LASTNAME, 01620000
 EMPLOYEE_SALARY, 01630000
 EMPLOYEE_BONUS; 01640000
 01650000
 END WHILE; 01660000
 01670000
CLOSE C1; 01680000
-- Set return parameters 01690000
SET DEPTSAL = TOTAL_SALARY; 01700000
SET BONUSCNT = BONUS_COUNTER; 01710000
 01720000
-- Open the cursor to the result set 01730000
OPEN DSN8ES1_RS_CSR; 01740000
END P1 01750000

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO" en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8ED3

Muestra cómo invocar el procedimiento almacenado PSM de ejemplo DSN8ES1 utilizando SQL estático.

```

/***** ****
* Module name = DSN8ED3 (DB2 sample program) * 00010000
* * 00020000
* DESCRIPTIVE NAME = Client for sample PSM Stored Procedure DSN8ES1 * 00030000
* * 00040000
* LICENSED MATERIALS - PROPERTY OF IBM * 00050000
* 5675-DB2 * 00070000
* (C) COPYRIGHT 2000 IBM CORP. ALL RIGHTS RESERVED. * 00080000
* * 00090000
* STATUS = VERSION 7 * 00100000
* * 00110000
* * 00120000
* Function: Demonstrates how to call the sample PSM stored procedure * 00130000
* DSN8ES1 using static SQL. * 00140000
* * 00150000
* Notes: * 00160000
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher * 00170000
* * 00180000
* Restrictions: * 00190000
* * 00200000
* Module type: C program * 00210000
* Processor: IBM C/C++ for OS/390 V1R3 or higher * 00220000
* Module size: See linkedit output * 00230000
* Attributes: Re-entrant and re-usable * 00240000
* * 00250000
* Entry Point: DSN8ED3 * 00260000
* Purpose: See Function * 00270000
* Linkage: DB2SQL * 00280000
* Invoked via SQL UDF call * 00290000
* * 00300000
* * 00310000
* Parameters: DSN8ED3 uses the C "main" argument convention of * 00320000
* argv (argument vector) and argc (argument count). * 00330000
* * 00340000
* - ARGV[0] = (input) pointer to a char[9], null- * 00350000
* terminated string having the name of * 00360000
* this program (DSN8ED3) * 00370000
* - ARGV[1] = (input) pointer to a char[4], null- * 00380000
* terminated string having the department * 00390000
* number to be passed to DSN8ES1. * 00400000
* - ARGV[2] = (input) pointer to a char[16], null- * 00410000
* terminated string having the location * 00420000
* name of a server to connect to process * 00430000
* the current request. This parameter is * 00440000
* optional. In its absence, the current * 00450000
* location is used. * 00460000
* * 00470000
* Normal Exit: Return Code: 0000 * 00480000
* - Message: none * 00490000
* * 00500000
* Error Exit: Return Code: 0008 * 00510000
* - Message: DSN8ED3 failed: Invalid parameter count * 00520000
* * 00530000
* - Message: <formatted SQL text from DSNTIAR> * 00540000

```

```

/*
 * External References:
 * - Routines/Services: DSNTIAR: DB2 msg text formatter
 * - Data areas : None
 * - Control blocks : None
 *
 * Pseudocode:
 * DSN8ED3:
 * - Verify that number of input parameters passed is either:
 * - 2 (program name and department number); or
 * - 3 (program name, department number, and (remote) server name)
 * - Other: issue diagnostic message and end with code 0008
 * - Connect to server location, if one was passed in.
 * - Call sample stored procedure DSN8ES1, passing the department
 * number as the argument of the first (input) parameter.
 * - if unsuccessful, call sql_error to issue a diagnostic mes-
 * sage, then end with code 0008.
 * - Report the following parameters, passed back from DSN8ES1:
 * - Total of salary and bonuses for department members
 * - Number of employees in the department who received a bonus
 * - If a result set was returned, call processResultSet to handle
 * it
 * End DSN8ED3
 *
 * processResultSet:
 * - Associate a locator with the result set passed from DSN8ES1,
 * which contains the serial number, first and last name, salary,
 * and bonus for each department member who got a bonus.
 * - if unsuccessful, call sql_error to issue a diagnostic mes-
 * sage, then end with code 0008.
 * - Allocate DSN8ES1_RS_CSR as a cursor for the locator
 * - if unsuccessful, call sql_error to issue a diagnostic mes-
 * sage, then end with code 0008.
 * - Do while not end of cursor
 * - Read the cursor
 * - If successful, print the row as a report line item
 * - else if not end of cursor, call sql_error to issue a diag-
 * nistic message, then end with code 0008.
 * - Close the cursor
 * - if unsuccessful, call sql_error to issue a diagnostic mes-
 * sage, then end with code 0008.
 * End processResultSet
 *
 * sql_error:
 * - call DSNTIAR to format the unexpected SQLCODE.
 * End sql_error
 */
***** C library definitions *****/
#include <stdio.h> 01030000
#include <stdlib.h> 01040000
#include <string.h> 01050000
#include <decimal.h> 01060000
#define NULLCHAR '\0' /* Null character */ 01070000
#define OUTLEN 80 /* Length of output line */ 01080000
#define DATA_DIM 10 /* Number of message lines */ 01090000
#define NOT_OK 0 /* Run status indicator: Error */ 01100000
#define OK 1 /* Run status indicator: Good */ 01110000
/* DB2 SQL Communication Area */
EXEC SQL INCLUDE SQLCA;
***** DB2 Host Variables *****/
EXEC SQL BEGIN DECLARE SECTION;
char hvDeptNo[4]; /* ID of department to query */ 01120000
short int niDeptNo = 0; /* Indic var for dept number */ 01130000
char hvServerName[17]; /* Location name of server */ 01140000
decimal(15,2) hvDeptEarnings = 0; /* Total dept salaries & bonus*/ 01150000
short int niDeptEarnings = 0; /* Indic var for dept salary */ 01160000
long int hvDeptBonusCount= 0; /* Total no. of bonuses in dpt*/ 01170000
short int niDeptBonusCount= 0; /* Indic var for dpt bonus cnt*/ 01180000

```

```

long int hvSequence; /* Result set row sequence no.*/ 01370000
char hvEmpno[7]; /* Employee number */ 01380000
char hvFirstName[13]; /* Employee first name */ 01390000
char hvLastName[16]; /* Employee last name */ 01400000
decimal(9,2) hvSalary = 0; /* Employee salary */ 01410000
decimal(9,2) hvBonus = 0; /* Employee bonus */ 01420000
 01430000
EXEC SQL END DECLARE SECTION; 01440000
 01450000
 01460000
/***** DB2 Result Set Locators *****/ 01470000
EXEC SQL BEGIN DECLARE SECTION; 01480000
 static volatile SQL TYPE IS RESULT_SET_LOCATOR *DSN8ES1_rs_loc; 01490000
EXEC SQL END DECLARE SECTION; 01500000
 01510000
 01520000
/***** DB2 Message Formatter *****/ 01530000
struct error_struct /* DSNTIAR message structure */ 01540000
{
 short int error_len;
 char error_text[DATA_DIM][OUTLEN];
} error_message = {DATA_DIM * (OUTLEN)}; 01550000
 01560000
#pragma linkage(dsntiar, OS)
 01570000
 01580000
 01590000
 01600000
 01610000
extern short int dsntiar(struct sqlca *sqlca,
 struct error_struct *msg,
 int *len); 01620000
 01630000
 01640000
 01650000
 01660000
/***** DSN8ED3 Global Variables *****/ 01670000
short int status = OK; /* DSN8ED3 run status */ 01680000
 01690000
long int completion_code = 0; /* DSN8ED3 return code */ 01700000
 01710000
 01720000
/***** DSN8ED3 Function Prototypes *****/ 01730000
int main(int argc, char *argv[]);
void processResultSet(void);
void sql_error(char locmsg[]);
 01740000
 01750000
 01760000
 01770000
 01780000
int main(int argc, char *argv[]) 01790000
/***** Get input parms, pass them to DSN8ES1, and process the results */
{* Get input parms, pass them to DSN8ES1, and process the results *} 01800000
{* Get input parms, pass them to DSN8ES1, and process the results *} 01810000
***** 01820000
{* printf("**** DSN8ED3: Sample client for DB2 PSM "
 "Stored Procedure Sample (DSN8ES1)\n\n");
 if(argc == 2) /* Only dept no. was passed */ 01830000
 {
 strcpy(hvDeptNo,argv[1]);
 }
 else if(argc == 3) /* Dept & server name passed */ 01840000
 {
 strcpy(hvDeptNo,argv[1]);
 strcpy(hvServerName,argv[2]);
 EXEC SQL CONNECT TO :hvServerName;
 if(SQLCODE != 0)
 sql_error(" *** Connect to server");
 }
 else
 {
 printf("DSN8ED3 failed: Invalid parameter count\n");
 status = NOT_OK;
 }
 if(status == OK)
 printf("Salary and Bonus Report for Department %s\n",hvDeptNo);
 if(status == OK)
 {
 EXEC SQL CALL DSN8.DSN8ES1(:hvDeptNo :niDeptNo,
 :hvDeptEarnings :niDeptEarnings,
 :hvDeptBonusCount:niDeptBonusCount);
 if(SQLCODE != 0 && SQLCODE != 466)
 sql_error(" *** Call DSN8ES1");
 else
 {
 printf("Total Department Salaries and Bonuses: %D(15,2)\n",
 hvDeptEarnings);
 }
 }
}
 01850000
 01860000
 01870000
 01880000
 01890000
 01900000
 01910000
 01920000
 01930000
 01940000
 01950000
 01960000
 01970000
 01980000
 01990000
 02000000
 02010000
 02020000
 02030000
 02040000
 02050000
 02060000
 02070000
 02080000
 02090000
 02100000
 02110000
 02120000
 02130000
 02140000
 02150000
 02160000
 02170000
 02180000

```

```

 printf("Total Number of Bonuses in Department: %i\n",
 hvDeptBonusCount);
 }

if(SQLCODE == 0 && status == OK)
if(hvDeptBonusCount != 0)
{
 printf("\n*** Error: Result set was expected from DSN8ES1 "
 "but was not received\n");
 status = NOT_OK;
}

if(SQLCODE == 466 && status == OK)
processResultSet();

if(status != OK)
completion_code = 8;

return(completion_code);
} /* end main */

void processResultSet(void)
/***
* If a result was returned by DSN8ES1, this function will process it *
**/
{
 printf("Bonus Earners are\n");
 printf("Serial First Name Last Name "
 "Salary Bonus\n");
 printf("----- ----- ----- "
 "-----\n");
 EXEC SQL ASSOCIATE LOCATOR(:DSN8ES1_rs_loc)
 WITH PROCEDURE DSN8.DSN8ES1;
 if(SQLCODE != 0)
 sql_error(" *** Associate locator DSN8ES1_rs_loc");
 if(SQLCODE == 0 && status == OK)
 {
 EXEC SQL ALLOCATE DSN8ES1_RS_CSR
 CURSOR FOR
 RESULT SET :DSN8ES1_rs_loc;
 if(SQLCODE != 0)
 sql_error(" *** Allocate cursor for DSN8ES1 result set");
 }
 while(SQLCODE == 0 && status == OK)
 {
 EXEC SQL FETCH DSN8ES1_RS_CSR
 INTO :hvSequence,
 :hvEmpno,
 :hvFirstName,
 :hvLastName,
 :hvSalary,
 :hvBonus;
 if(SQLCODE == 0)
 printf("%s %s %s %9D(9,2) %9D(9,2)\n",
 hvEmpno, hvFirstName, hvLastName, hvSalary, hvBonus);
 else if(SQLCODE != 100)
 sql_error(" *** Fetch from DSN8ES1 result set cursor");
 }
} /* end void processResultSet(void) */

/***
** SQL error handler
**/
void sql_error(char locmsg[]) /*proc*/
{
 short int rc; /* DSNTIAR Return code */
 int j,k; /* Loop control */
 static int lrecL = OUTLEN; /* Width of message lines */
}

```

```

/***** 03010000
* set status to prevent further processing * 03020000
***** 03030000
status = NOT_OK; 03040000
 03050000
/***** 03060000
* print the locator message * 03070000
***** 03080000
printf(".%80s\n", locmsg); 03090000
 03100000
/***** 03110000
* format and print the SQL message * 03120000
***** 03130000
rc = dsntiar(&sqlca, &error_message, &lrecl);
if(rc == 0)
 for(j=0; j<DATA_DIM; j++)
 {
 for(k=0; k<OUTLEN; k++)
 putchar(error_message.error_text[j][k]);
 putchar('\n');
 }
else
{
 printf(" *** ERROR: DSNTIAR could not format the message\n");
 printf(" *** SQLCODE is %d\n",SQLCODE);
 printf(" *** SQLERRM is \n");
 for(j=0; j<sqlca.sqlerrml; j++)
 printf("%c", sqlca.sqlerrmc[j]);
 printf("\n");
}
} /* end of sql_error */

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8ES2

Acepta la cantidad base de bonificación (BONUSBAS) que se va a adjudicar a empleados que son directores.

```

-- DSN8.DSN8ES2: SOURCE MODULE FOR SAMPLE SQL PROCEDURE 00010000
-- 00020000
-- LICENSED MATERIALS - PROPERTY OF IBM 00030000
-- 5635-DB2 00040000
-- (C) COPYRIGHT 2000, 2006 IBM CORP. ALL RIGHTS RESERVED. 00050000
-- 00060000
-- STATUS = VERSION 9 00070000
-- 00080000
-- Function: Accepts a bonus base amount (BONUSBAS) to be awarded to 00090000
-- employees who are managers. Determines a bonus premium 00100000
-- (BONUSPRM) for each manager, according to the number of 00110000
-- employees he or she manages. Updates the BONUS column of 00120000
-- the sample EMP table for each manager with the sum of the 00130000
-- bonus base and his or her bonus premium. Returns the 00140000
-- total (BONUSTOT) of all bonuses awarded to managers. 00150000
-- 00160000
-- SQLERRCD is null unless an SQL exception occurs, in which 00170000
-- case SQLERRCD is set to the current SQLCODE 00180000
-- 00190000
-- Notes: 00200000
-- Dependencies: 00210000
-- - Requires DB2 precompiler support for SQL procedures (DSNHPSM) 00220000
-- 00230000
-- Restrictions: 00240000
-- 00250000
-- Module Type: SQL Procedure 00260000
-- Processor: DB2 for OS/390 precompiler and IBM C/C++ for OS/390 00270000
-- or a subsequent release 00280000
-- Attributes: Re-entrant and re-usable 00290000
-- 00300000
-- Entry Point: DSN8ES2 00310000
-- Purpose: See Function, above 00320000
-- 00330000
-- Parameters:
-- - Input: BONUSBAS DECIMAL(15,2) 00340000
-- - Output: BONUSTOT DECIMAL(15,2) 00350000
-- 00360000

```

```

-- SQLERRCD INTEGER 00370000
-- 00380000
-- Normal Exit: 00390000
-- Error Exit: 00400000
-- 00410000
-- 00420000
-- External References: 00430000
-- - EMP : DB2 Sample Employee Table 00440000
-- 00450000
-- Pseudocode: 00460000
-- - Open a cursor on sample DEPT and EMP tables, that identifies 00470000
-- department managers and the number of persons in each department 00480000
-- 00490000
-- - For each manager found: 00500000
-- - Determine the bonus premium according to the number of 00510000
-- employees managed: $1000 for more than 10, $500 for 6 to 10, 00520000
-- $100 for 1 to 5 00530000
-- - Update the manager's bonus in the sample EMP table with the 00540000
-- sum of the bonus base and the bonus premium 00550000
-- - Add the manager's bonus to the total bonuses bucket. 00560000
-- - Return total amount of bonuses awarded 00570000
-- 00580000
CREATE PROCEDURE DSN8.DSN8ES2 00590000
 (IN BONUSBAS DECIMAL(15,2), 00600000
 OUT BONUSTOT DECIMAL(15,2), 00610000
 OUT SQLERRCD INTEGER) 00620000
PARAMETER CCSID EBCDIC 00630000
 FENCED 00640000
 RESULT SETS 0 00650000
 LANGUAGE SQL 00660000
NOT DETERMINISTIC 00670000
MODIFIES SQL DATA 00680000
 COLLID DSN8ES!! 00700000
 WLM ENVIRONMENT WLMENV 00710000
 ASUTIME NO LIMIT 00720000
 COMMIT ON RETURN NO 00760000
 00770000
P1: BEGIN NOT ATOMIC 00780000
 DECLARE MANAGER_ID CHAR(6) CCSID EBCDIC; 00790000
 DECLARE NUM_EMPLOYEES INT DEFAULT 0; 00800000
 DECLARE BONUSPRM DECIMAL(15,2) DEFAULT 0; 00810000
 DECLARE BONUSBKT DECIMAL(15,2) DEFAULT 0; 00820000
 DECLARE END_TABLE INT DEFAULT 0; 00830000
 DECLARE SQLCODE INT; 00831000
 00840000
 -- Cursor gets id and no. of direct reports for each manager 00850000
 DECLARE C1 CURSOR FOR 00860000
 SELECT DEPT.MGRNO,
 COUNT(DISTINCT EMP.EMPNO) 00870000
 FROM DEPT DEPT,
 EMP EMP
 WHERE EMP.WORKDEPT = DEPT.DEPTNO 00880000
 GROUP BY EMP.WORKDEPT, DEPT.MGRNO; 00890000
 00900000
 DECLARE CONTINUE HANDLER FOR NOT FOUND 00910000
 SET END_TABLE = 1; 00920000
 00930000
 DECLARE EXIT HANDLER FOR SQLEXCEPTION 00940000
 SET SQLERRCD = SQLCODE; 00950000
 00960000
 SET BONUSTOT = NULL; 00970000
 SET SQLERRCD = NULL; 00980000
 00990000
 OPEN C1; 01000000
 FETCH C1
 INTO MANAGER_ID,
 NUM_EMPLOYEES; 01010000
 01020000
 WHILE END_TABLE = 0 DO 01030000
 CASE
 WHEN(NUM_EMPLOYEES > 10) THEN 01040000
 SET BONUSPRM = 1000.00; 01050000
 WHEN(NUM_EMPLOYEES > 5) THEN 01060000
 SET BONUSPRM = 500.00; 01070000
 WHEN(NUM_EMPLOYEES > 0) THEN 01080000
 SET BONUSPRM = 100.00; 01090000
 ELSE
 SET BONUSPRM = 0.00; 01100000
 END CASE;
 UPDATE EMP 01110000
 01120000
 01130000
 01140000
 01150000
 01160000
 01170000
 01180000
 01190000
 01200000
 01210000

```

```

 SET BONUS = BONUSBAS + BONUSPRM 01220000
 WHERE EMPNO = MANAGER_ID; 01230000
 01240000
 SET BONUSBKT = BONUSBKT + BONUSBAS + BONUSPRM; 01250000
 01260000
 FETCH C1 01270000
 INTO MANAGER_ID, 01280000
 NUM_EMPLOYEES; 01290000
 01300000
 END WHILE; 01310000
 01320000
 CLOSE C1; 01330000
 01340000
 SET BONUSTOT = BONUSBKT; 01350000
 01360000
 END P1 01370000

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO" en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8ED6

Muestra cómo utilizar WLM\_REFRESH, el procedimiento almacenado de muestra para renovar un entorno WLM.

```

/***** ****
* Module name = DSN8ED6 (DB2 sample program) * 00010000
* * 00020000
* * 00030000
* DESCRIPTIVE NAME = Caller for sample WLM_REFRESH stored procedure * 00040000
* * 00050000
* LICENSED MATERIALS - PROPERTY OF IBM * 00060000
* 5675-DB2 * 00070000
* (C) COPYRIGHT 1999, 2000 IBM CORP. ALL RIGHTS RESERVED. * 00090000
* * 00120000
* STATUS = VERSION 7 * 00130000
* * 00160000
* Function: Demonstrates how to use WLM_REFRESH, the sample stored * 00220000
* procedure for refreshing a WLM environment * 00230000
* * 00240000
* Notes: * 00250000
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher * 00260000
* * 00270000
* Restrictions: * 00280000
* * 00290000
* Module type: C program * 00300000
* Processor: IBM C/C++ for OS/390 V1R3 or higher * 00310000
* Module size: See linkedit output * 00320000
* Attributes: Re-entrant and re-usable * 00330000
* * 00340000
* Entry Point: DSN8ED6 * 00350000
* Purpose: See Function * 00360000
* Linkage: Standard OS/390 linkage * 00370000
* * 00380000
* * 00390000
* Parameters: DSN8ED6 uses the C "main" argument convention of * 00400000
* argv (argument vector) and argc (argument count). * 00410000
* * 00420000
* - ARGV[0] = (input) pointer to a char[9], null- * 00430000
* terminated string having the name of * 00440000
* this program (DSN8ED6) * 00450000
* - ARGV[1] = (input) pointer to a char[32] null- * 00460000
* terminated string having the name of * 00470000
* the WLM environment to be refreshed. * 00480000
* - ARGV[2] = (input) pointer to a char[4], null- * 00490000
* terminated string having the DB2 sub- * 00500000
* system id associated with the WLM * 00510000
* environment to be refreshed * 00520000
* - ARGV[3] = (input) pointer to a char[8], null- * 00530000
* terminated string having the name of a * 00540000
* secondary authorization id that has * 00550000
* access to the resource profile <ssid>.- * 00560000
* WLM_REFRESH.<wlm-environment-name>. * 00570000
* in resource class DSNR. WLM_REFRESH * 00580000
* requires READ access on that profile * 00590000
* in order to fulfill a refresh request. * 00600000
* * 00610000
* * 00620000
* Normal Exit: Return Code: 0000

```

```

* - Message: none * 00630000
*
* Error Exit: Return Code: 1999
* - Message: Error: Invalid call parameter count * 00640000
* Specify either 2 or 3 call para- * 00650000
* meters for DSN8ED6, as follows: * 00660000
* 1. The name of a WLM environment * 00670000
* to be refreshed (1-32 characters) * 00680000
* 2. The DB2 subsystem id (1-4 char- * 00690000
* actors) * 00700000
* 3. A secondary authorization id for * 00710000
* submitting the refresh request * 00720000
* (Optional. 1-8 characters) * 00730000
* (Optional. 1-8 characters) * 00740000
* (Optional. 1-8 characters) * 00750000
* (Optional. 1-8 characters) * 00760000
*
* - Message: <formatted SQL text from DSNTIAR> * 00770000
*
*
* External References:
* - Routines/Services: DSNTIAR: DB2 msg text formatter * 00780000
* - Data areas : None * 00790000
* - Control blocks : None * 00800000
*
* Pseudocode:
* DSN8ED6:
* - Verify that number of input parameters passed is either: * 00810000
* - 2 (WLM environment name and DB2 ssid); or * 00820000
* - 3 (WLM environment name, DB2 ssid, and secondary auth id * 00830000
* - Other: issue diagnostic message and end with code 1999 * 00840000
* - Set current SQLID to secondary auth id, if one was passed in * 00850000
* - Call sample stored procedure WLM_REFRESH * 00860000
* - if unsuccessful, call sql_error to issue a diagnostic mes- * 00870000
* sage, then end with code 1999. * 00880000
* - Report the following parameters, passed back from WLM_REFRESH: * 00890000
* - Return code * 00900000
* - Return message * 00910000
* - Set DSN8ED6 return code from WLM_REFRESH return code * 00920000
* End DSN8ED6 * 00930000
* * 00940000
* * 00950000
* * 00960000
* * 00970000
* * 00980000
* * 00990000
* * 01000000
* * 01010000
* * 01020000
* * 01030000
* * 01032000
* Change log:
* 01/15/04 PQ79759 - Increase authID to 9 bytes * 01034000
*
***** C library definitions *****/
#include <stdio.h> 01050000
#include <stdlib.h> 01060000
#include <string.h> 01070000
01080000
01090000
01100000
01110000
01120000
01130000
01140000
01150000
01160000
01170000
01180000
01190000
01200000
01210000
01220000
01230000
01240000
01250000
01260000
01270000
01280000
01290000
01300000
01310000
01320000
01330000
01340000
01350000
01360000
01370000
01380000
01390000
01400000
01410000
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
 char wlmEnvName[33]; /* WLM environment name */ 01250000
 char ssID[5]; /* Subsystem name */ 01260000
 char authID[9]; /* Current authorization id */ 01270000
 char message[123]; /* WLM_REFRESH return message */ 01280000
 long int code; /* WLM_REFRESH return code */ 01290000
EXEC SQL END DECLARE SECTION;
EXEC SQL BEGIN DECLARE SECTION;
 struct error_struct /* DSNTIAR message structure */ 01300000
 {
 short int error_len;
 char error_text[DATA_DIM][OUTLEN];
 error_message = {DATA_DIM * (OUTLEN)};
 }
#pragma linkage(dsntiar, OS)

```

```

extern short int dsntiar(struct sqlca *sqlca,
 struct error_struct *msg,
 int *len);
 01420000
 01430000
 01440000
 01450000
 01460000
 01470000
 01480000
 01490000
 01500000
 01510000
 01520000
 01530000
 01540000
 01550000
 01560000
 01570000
 01580000
 01590000
 01600000
 01610000
 01620000
 01630000
 01640000
 01650000
 01660000
 01670000
 01680000
 01690000
 01700000
 01710000
 01720000
 01730000
 01740000
 01750000
 01760000
 01770000
 01780000
 01790000
 01800000
 01810000
 01820000
 01830000
 01840000
 01850000
 01860000
 01870000
 01880000
 01890000
 01900000
 01910000
 01920000
 01930000
 01940000
 01950000
 01960000
 01970000
 01980000
 01990000
 02000000
 02010000
 02020000
 02030000
 02040000
 02050000
 02060000
 02070000
 02080000
 02090000
 02100000
 02110000
 02120000
 02130000
 02140000
 02150000
 02160000
 02170000
 02180000
 02190000
 02200000
 02210000
 02220000
 02230000

extern short int dsntiar(struct sqlca *sqlca,
 struct error_struct *msg,
 int *len);
 01420000
 01430000
 01440000
 01450000
 01460000
 01470000
 01480000
 01490000
 01500000
 01510000
 01520000
 01530000
 01540000
 01550000
 01560000
 01570000
 01580000
 01590000
 01600000
 01610000
 01620000
 01630000
 01640000
 01650000
 01660000
 01670000
 01680000
 01690000
 01700000
 01710000
 01720000
 01730000
 01740000
 01750000
 01760000
 01770000
 01780000
 01790000
 01800000
 01810000
 01820000
 01830000
 01840000
 01850000
 01860000
 01870000
 01880000
 01890000
 01900000
 01910000
 01920000
 01930000
 01940000
 01950000
 01960000
 01970000
 01980000
 01990000
 02000000
 02010000
 02020000
 02030000
 02040000
 02050000
 02060000
 02070000
 02080000
 02090000
 02100000
 02110000
 02120000
 02130000
 02140000
 02150000
 02160000
 02170000
 02180000
 02190000
 02200000
 02210000
 02220000
 02230000

/****** DSN8ED6 Global Variables *****/
short int status = OK; /* DSN8ED6 run status */ 01470000
long int completion_code = 0; /* DSN8ED6 return code */ 01480000
 01490000
 01500000
 01510000
 01520000
 01530000
 01540000
 01550000
 01560000
 01570000
 01580000
 01590000
 01600000
 01610000
 01620000
 01630000
 01640000
 01650000
 01660000
 01670000
 01680000
 01690000
 01700000
 01710000
 01720000
 01730000
 01740000
 01750000
 01760000
 01770000
 01780000
 01790000
 01800000
 01810000
 01820000
 01830000
 01840000
 01850000
 01860000
 01870000
 01880000
 01890000
 01900000
 01910000
 01920000
 01930000
 01940000
 01950000
 01960000
 01970000
 01980000
 01990000
 02000000
 02010000
 02020000
 02030000
 02040000
 02050000
 02060000
 02070000
 02080000
 02090000
 02100000
 02110000
 02120000
 02130000
 02140000
 02150000
 02160000
 02170000
 02180000
 02190000
 02200000
 02210000
 02220000
 02230000

/****** DSN8ED6 Function Prototypes *****/
int main(int argc, char *argv[]);
void sql_error(char locmsg[]);

int main(int argc, char *argv[])
/****** DSN8ED6: Sample caller of WLM_REFRESH stored */
{ printf("**** DSN8ED6: Sample caller of WLM_REFRESH stored "
 "procedure (DSNTWR)\n");
 printf("\n");
 if(argc < 3 || argc > 6)
 { printf("* Error: Invalid call parameter count\n");
 printf("* Specify either 2 or 3 call parameters "
 "for DSN8ED6, as follows:\n");
 printf("* 1. The name of a WLM environment to be "
 "refreshed (1-32 characters)\n");
 printf("* 2. The DB2 subsystem id (1-4 characters)\n");
 printf("* 3. A secondary authorization id for "
 "submitting the refresh request\n");
 printf("* (Optional. 1-8 characters)\n");
 status = NOT_OK;
 }
 else if(strlen(argv[1]) < 1 || strlen(argv[1]) > 32)
 { printf("* Error: The WLM environment name must be 1-32 "
 "characters in length\n");
 status = NOT_OK;
 }
 else if(strlen(argv[2]) < 1 || strlen(argv[2]) > 4)
 { printf("* Error: The DB2 subsystem id must be 1-4 "
 "characters in length\n");
 status = NOT_OK;
 }
 else if(argc == 4 && (strlen(argv[3]) < 1 || strlen(argv[3]) > 8))
 { printf("* Error: The secondary authorization id must be 1-8 "
 "characters in length\n");
 status = NOT_OK;
 }
 else
 { strcpy(wlmEnvName,argv[1]);
 strcpy(ssID,argv[2]);
 }
}

/* Change authid if one was passed in */
if(status == OK && argc == 4)
{ strcpy(authID,argv[3]);
 EXEC SQL SET CURRENT SQLID = :authID;
 if(SQLCODE != 0)
 sql_error("Error setting SQLID");
}

/* Call WLM_REFRESH to refresh the specified WLM environment */
if(status == OK)
{ EXEC SQL CALL SYSPROC.WLM_REFRESH(
 :wlmEnvName,
 :ssID,
 :message,
 :code);
 if(SQLCODE != 0)
 sql_error("Error calling SYSPROC.WLM_REFRESH");
 else
 { printf("* Results: WLM_REFRESH returned code %i "
 "and the following message:\n",code);
 printf("* %s\n",message);
 }
}

```

```

if(status != OK)
 completion_code = 1999;
else
 completion_code = code;

return(completion_code);
} /* end main */

void sql_error(char locmsg[]) /* SQL message formatter */ 02240000
{ short int rc; /* DSNTIAR Return code */ 02250000
 int j,k; /* Loop control */ 02260000
 static int lrecl = OUTLEN; /* Width of message lines */ 02270000
 */ 02280000
 */ 02290000
 */ 02300000
 */ 02310000
 */ 02320000
 */ 02330000
 */ 02340000
 */ 02350000
 */ 02360000
 */ 02370000
 */ 02380000
 */ 02390000
 * set status to prevent further processing */ 02400000
 *****/ 02410000
status = NOT_OK;
 */ 02420000
 */ 02430000
 */ 02440000
 * print the locator message */ 02450000
 *****/ 02460000
printf("%.80s\n", locmsg); */ 02470000
 */ 02480000
 */ 02490000
 * format and print the SQL message */ 02500000
 *****/ 02510000
rc = dsntiar(&sqlca, &error_message, &lrecl);
 */ 02520000
if(rc == 0)
 for(j=0; j<DATA_DIM; j++)
{
 for(k=0; k<OUTLEN; k++)
 putchar(error_message.error_text[j][k]);
 putchar('\n');
}
else
{
 printf(" *** ERROR: DSNTIAR could not format the message\n");
 printf(" *** SQLCODE is %d\n",SQLCODE);
 printf(" *** SQLERRM is \n");
 for(j=0; j<sqlca.sqlerrml; j++)
 printf("%c", sqlca.sqlerrmc[j]);
 printf("\n");
}
} /* end of sql_error */
 */ 02530000
 */ 02540000
 */ 02550000
 */ 02560000
 */ 02570000
 */ 02580000
 */ 02590000
 */ 02600000
 */ 02610000
 */ 02620000
 */ 02630000
 */ 02640000
 */ 02650000
 */ 02660000
 */ 02670000
 */ 02680000
 */ 02690000
 */ 02700000

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8ED7

Invoca el procedimiento almacenado proporcionado por Db2 ADMIN\_INFO\_SYSPARM, que devuelve los valores actuales de los parámetros de subsistema Db2.

```

/*****
* Module name = DSN8ED7 (DB2 sample program) *
* DESCRIPTIVE NAME = Caller for SYSPROC.ADMIN_INFO_SYSPARM *
* (IFCID 106 formatter stored procedure) *
* Licensed Materials - Property of IBM *
* 5635-DB2 *
* (C) COPYRIGHT 1982, 2006 IBM Corp. All Rights Reserved. *
* STATUS = Version 11 *
* Function: Calls DB2-provided stored procedure ADMIN_INFO_SYSPARM, *
* which returns the current settings of the DB2 subsystem *
* parameters. These settings are then written in *
* report format to standard output. *
* Notes: *
* Dependencies: Requires IBM C/C++ for z/OS *
* Restrictions: *

```

```

/*
 * Module type: C program
 * Processor: IBM C/C++ for z/OS
 * Module size: See linkedit output
 * Attributes: Re-entrant and re-usable
 *
 * Entry Point: DSN8ED7
 * Purpose: See Function
 * Linkage: Standard z/OS linkage
 *
 *
 * Parameters: none
 *
 * Normal Exit: Return Code: 0000
 * - Message: report of DB2 subsystem parameter settings
 *
 * Error Exit: Return Code: 0012
 * - Message: <formatted SQL text from DSNTIAR>
 *
 *
 * External References:
 * - Routines/Services: DSNTIAR: DB2 msg text formatter
 * - Data areas : None
 * - Control blocks : None
 *
 * Pseudocode:
 * DSN8ED7:
 * - Call ADMIN_INFO_SYSPARM
 * - if unsuccessful, call sql_error to issue a diagnostic mes-
 * sage, then end with code 0012.
 * - Associate a locator variable with the result set
 * - Allocate the result set cursor
 * - Fetch first row from the result set
 * - Print headings
 * - Output the content of the result set's current row and fetch
 * the next row, until all rows have been fetched
 * - Check for successful processing of result set
 * End DSN8ED7
 *
 * sql_error:
 * - call DSNTIAR to format the unexpected SQLCODE.
 * End sql_error
 *
 * Change activity =
 * 11/07/2012 Convert from SYSPROC.DSNWZP dn1651_inst1 / dn1651
 * to SYSPROC.ADMIN_INFO_SYSPARM
 *

 **** Equates ****
#define NOT_OK 0 /* Run status indicator: Error */
#define OK 1 /* Run status indicator: Good */

#define OUTLEN 80 /* Length of DSNTIAR line */
#define DATA_DIM 10 /* Number of DSNTIAR lines */

 **** Includes ****
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

 **** DB2 SQL Communication Area ****
EXEC SQL INCLUDE SQLCA;

 **** DB2 Host Variables ****
EXEC SQL BEGIN DECLARE SECTION;

 **** Host variables for ADMIN_INFO_SYSPARM parameters ****
char hvDB2_MEMBER[9]; /* host var, target DB2 */
short int niDB2_MEMBER = -1; /* indic var for above parm */

long int hvRETURN_CODE = 0; /* host var, return code */
short int niRETURN_CODE = 0; /* indic var for above parm */

char hvMSG[1332]; /* host var, status message */
short int niMSG = 0; /* indic var for above parm */

 **** Result set locator for ADMIN_INFO_SYSPARM result set ****
static volatile SQL TYPE IS RESULT_SET_LOCATOR *DB2_SYSPARM_rs_loc;

 **** Host variables for ADMIN_INFO_SYSPARM result set ****
long int hvROWNUM = 0; /* host var, row number */

```

```

char hvMACRO[9]; /* host var, zparm macro name */
char hvPARAMETER[41]; /* host var, zparm name */
char hvINSTALL_PANEL[9]; /* host var, inst panel name */
short int niINSTALL_PANEL = 0; /* indic var for above parm */
char hvINSTALL_FIELD[41]; /* host var, inst field name */
short int niINSTALL_FIELD = 0; /* indic var for above parm */
char hvINSTALL_LOCATION[13];/*host var, inst field numb */
short int niINSTALL_LOCATION = 0; /*indic var for above parm */
char hvVALUE[2049]; /* host var, zparm setting */
short int niADDITIONAL_INFO[201];/*host var, zparm setting */
short int niADDITIONAL_INFO = 0; /*indic var for above parm */

EXEC SQL END DECLARE SECTION;

/***** DB2 Message Formatter *****/
struct error_struct /* DSNTIAR message structure */
{ short int error_len;
 char error_text[DATA_DIM][OUTLEN];
} error_message = {DATA_DIM * (OUTLEN)};

#pragma linkage(dsntiar, OS)

extern short int dsntiar(struct sqlca *sqlca,
 struct error_struct *msg,
 int *len);

/***** DSN8ED7 Global Variables *****/
short int status = OK; /* DSN8ED7 run status */

/***** DSN8ED7 Function Prototypes *****/
int main(int argc, char *argv[]);
void sql_error(char locmsg[]); /* Calls SQL text formatter */

/*****
 * Get DB2's current subsystem and DSNHDECP parameter settings
 *****/
int main(int argc, char *argv[])
{
 char msgBuf[1400]; /* message buffer */

 /*****
 * Call SYSPROC.ADMIN_INFO_SYSParm
 *****/
 EXEC SQL
 CALL SYSPROC.ADMIN_INFO_SYSParm
 (:hvDB2_MEMBER :nidB2_MEMBER
 , :hvRETURN_CODE :niRETURN_CODE
 , :hvMSG :nimSG
);

 if(SQLCODE != 466)
 { sprintf(msgBuf
 , "DSN8ED7: Error calling stored procedure "
 , "ADMIN_INFO_SYSParm: \n"
 , " Return code=%i,\n"
 , " Message=%s"
 , hvRETURN_CODE
 , hvMSG
);
 sql_error(msgBuf);
 }

 /*****
 * Associate a locator variable with the result set
 *****/
 if(status == OK)
 {
 EXEC SQL ASSOCIATE LOCATOR
 (:DB2_SYSParm_rs_loc)
 WITH PROCEDURE SYSPROC.ADMIN_INFO_SYSParm;

 if(SQLCODE != 0)
 { sql_error("*** Associate result set locator "
 , "call unsuccessful.");
 }
 }
}

```

```

* Allocate the result set cursor

if(status == OK)
{
 EXEC SQL ALLOCATE DB2_SYSPARM_RS_CSR
 CURSOR FOR
 RESULT SET :DB2_SYSPARM_rs_loc;

 if (SQLCODE != 0)
 { sql_error("*** Allocate result set cursor "
 "call unsuccessful.");
 }
}

* Fetch first row from the result set

if(status == OK)
{
 EXEC SQL FETCH DB2_SYSPARM_RS_CSR
 INTO :hvROWNUM
 , :hvMACRO
 , :hvPARAMETER
 , :hvINSTALL_PANEL :niINSTALL_PANEL
 , :hvINSTALL_FIELD :niINSTALL_FIELD
 , :hvINSTALL_LOCATION :niINSTALL_LOCATION
 , :hvVALUE
 , :hvADDITIONAL_INFO
 ;

 if (SQLCODE != 0)
 { sql_error("*** Priming fetch of result "
 "set cursor unsuccessful");
 }
}

* Write the report header

if(status == OK)
{
 printf("DSN8ED7: Sample DB2 for z/OS "
 "Configuration Setting Report Generator\n \n");

 printf("Macro Parameter "
 "Current "
 "Description/ "
 "Install Fld \n");
 printf("Name Name "
 "Setting "
 "Install Field Name"
 "Panel ID No. \n");
 printf("----- - - - - - "
 "----- - - - - - "
 "----- - - - - - "
 "----- - - - - \n");
}

* Output the contents of the result set

while(SQLCODE == 0 && status == OK)
{
 if(strcmp(hvMACRO,"DSN6SYSP") == 0
 || strcmp(hvMACRO,"DSN6LOGP") == 0
 || strcmp(hvMACRO,"DSN6ARVP") == 0
 || strcmp(hvMACRO,"DSN6SPRM") == 0
 || strcmp(hvMACRO,"DSN6FAC") == 0
 || strcmp(hvMACRO,"DSN6GRP") == 0
 || strcmp(hvMACRO,"DSNHDECP") == 0
)
 printf("%-9.8s"
 "%-26.25s"
 "%-40.39s"
 "%-40.39s"
 "%-9.8s"
 "%4.4s\n"
 , hvMACRO
 , hvPARAMETER
 , hvVALUE

```

```

 , hvINSTALL_FIELD
 , hvINSTALL_PANEL
 , hvINSTALL_LOCATION
);

EXEC SQL FETCH DB2_SYSPARM_RS_CSR
 INTO :hvROWNUM
 , :hvMACRO
 , :hvPARAMETER
 , :hvINSTALL_PANEL :niINSTALL_PANEL
 , :hvINSTALL_FIELD :niINSTALL_FIELD
 , :hvINSTALL_LOCATION :niINSTALL_LOCATION
 , :hvVALUE
 , :hvADDITIONAL_INFO
 ;
}

/***
* Check for successful processing of result set
***/
if (SQLCODE != 100 && status == OK)
{
 sql_error("*** Fetch of result set cursor "
 "unsuccessful.");
}

if(status == OK)
 return(0);
else
 return(12);
} /* end: main */

void sql_error(char locmsg[]) /* SQL message formatter */
{ short int rc; /* DSNTIAR Return code */
int j,k; /* Loop control */
static int lrecl = OUTLEN; /* Width of message lines */

/***
* Set status to prevent further processing
***/
status = NOT_OK;

/***
* Print the locator message
**/
printf("%s\n", locmsg);

/***
* Format and print the SQL message
**/
rc = dsntiar(&sqlca, &error_message, &lrecl);
if(rc == 0)
 for(j=0; j<DATA_DIM; j++)
 { for(k=0; k<OUTLEN; k++)
 putchar(error_message.error_text[j][k]);
 putchar('\n');
 }
else
 { printf(" *** ERROR: DSNTIAR could not format the message\n");
 printf(" *** SQLCODE is %d\n",SQLCODE);
 printf(" *** SQLERRM is \n");
 for(j=0; j<sqlca.sqlerrml; j++)
 printf("%c", sqlca.sqlerrmc[j]);
 printf("\n");
 }

} /* end of sql_error */

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8ED9

Muestra cómo utilizar un programa de aplicación para invocar DSN8ES3, un procedimiento de SQL nativo de ejemplo.

```

* Module name = DSN8ED9 (sample program)
*
* DESCRIPTIVE NAME: Sample client for:
* DSN8ES3 (DB2 sample native SQL procedure)
*
*
* LICENSED MATERIALS - PROPERTY OF IBM
* 5650-DB2
* (C) COPYRIGHT 2006, 2016 IBM CORP. ALL RIGHTS RESERVED.
*
* STATUS = VERSION 12
*
* Function: Demonstrates how to use an application program to call
* DSN8ES3, a sample native SQL procedure. DSN8ED9
* receives the schema and name of a stored procedure
* and passes it to DSN8ES3 to request the CREATE PROCEDURE
* statement.
*
* Notes:
* Dependencies: Requires DSN8.DSN8ES3
*
* Restrictions:
*
* Module type: C program
* Processor: DB2 Precompiler
* IBM C/C++ for z/OS
* Module size: See linkedit output
* Attributes: Reentrant and reusable
*
* Entry point: DSN8ED9
* Purpose: See Function
* Linkage: Standard MVS program invocation, three parameters.
*
* Parameters: DSN8ED9 uses the C "main" argument convention of
* argv (argument vector) and argc (argument count).
*
* - ARGV[0]: (input) pointer to a char[9],
* null-terminated string having the name of
* this program (DSN8ED9)
* - ARGV[1]: (input) pointer to a char[129],
* null-terminated string having the schema
* of a stored procedure
* - ARGV[2]: (input) pointer to a char[129],
* null-terminated string having the name of
* a stored procedure
* - ARGV[3]: (input) pointer to a char[17],
* null-terminated string having the name of
* the server where DSN8ES3 is to be run.
* This is an optional parameter; the local
* server is used if no argument is provided.
*
* Inputs: None
*
* Outputs: Standard output (SYSPRINT)
*
* Normal Exit: Return Code: 0
* - Message: CREATE PROCEDURE statement for specified
* stored procedure
*
* Normal with Warnings Exit: Return Code: 0004
* - Message: DSN8ES3 ran successfully but returned
* no output
*
* Error Exit: Return Code: 0012
* - Message: DSN8ES3 has completed with return code <n>
* - Message: The length of the argument specified for
* the <parameter-name> does not fall within
* the required bounds of <minimum-length>
* and <maximum-length>
* - Message: DSN8ED9 was invoked with <parameter-count>
* parameters. At least 2 parameters are
* required
* - Message: <formatted SQL text from DSNTIAR>
*
* External References:
* - Routines/Services: DSNTIAR: DB2 msg text formatter
* - Data areas : None
* - Control blocks : None
*

```

```

* Pseudocode: *
* DSN8ED9:
* - call getCallParms to receive and validate call parm arguments*
* - call connectToLocation
* - call callDSN8ES3 to invoke the sample native SQL procedure *
* - call processDSN8ES3resultSet to output results from DSN8ES3 *
* End DSN8ED9
*
*
* Change activity =
* 04/22/2015 Storage overlay stops output d176357 *
*

***** C library definitions *****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <decimal.h>

***** Equates *****/
#define NULLCHAR '\0' /* Null character */

#define RETNRM 0 /* Normal return code @04*/
#define RETWRN 4 /* Warning return code */
#define RETERR 8 /* Error return code */
#define RETSEV 12 /* Severe error return code */

enum flag {No, Yes}; /* Settings for flags */

***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

***** DB2 Host Variables *****/
EXEC SQL BEGIN DECLARE SECTION;
 long int hvSequence; /* Result set row sequence no.*/
 char hvLine[80]; /* line */
 char hvSpSchema[129]; /* Stored procedure schema */
 short int niSpSchema = 0; /* Indic var for schema */
 char hvSpName[129]; /* Stored procedure name */
 short int niSpName = 0; /* Indic var for name */
 char hvLocationName[17]; /* Server location name */

EXEC SQL END DECLARE SECTION;

***** DB2 Result Set Locators *****/
EXEC SQL BEGIN DECLARE SECTION;
 static volatile SQL TYPE IS RESULT_SET_LOCATOR *DSN8ES3_rs_loc;
EXEC SQL END DECLARE SECTION;

***** DSN8ED9 Global Variables *****/
unsigned short resultSetReturned = 0; /* DSN8ES3 result set status */
long int rc = 0; /* DSN8ED9 return code */

***** DSN8ED9 Function Prototypes *****/
int main() /* DSN8ED9 driver */
{
 (int argc, /* - Input argument count */
 char *argv[]) /* - Input argument vector */
};

void getCallParms() /* Process args to call parms */
{
 (int argc, /* - Input argument count */
 char *argv[]) /* - Input argument vector */
};

void connectToLocation(void); /* Connect to DB2 location */

void callDSN8ES3(void); /* Call DSN8ES3 */

void processDSN8ES3resultSet(void); /* Process DSN8ES3 result set */

void associateResultSetLocator(void); /* Assoc DSN8ES3 RS locator */

void allocateResultSetCursor(void); /* Alloc DSN8ES3 RS cursor */

void writeDSN8ES3results(void); /* Output DSN8ES3 results */

void fetchFromResultSetCursor(void); /* Read DSNTSPMP RS cursor */

void issueInvalidCallParmCountError /* Handler for parm count err */
(
 (int argc /* - in: no. parms received */
);

void issueInvalidParmLengthError /* Handler for parm len error */
(
 (char *parmName, /* - in: identify of parm */
 int minLength, /* - in: min valid length */
 int maxLength) /* - in: max valid length */
);

```

```

);
void issueSqlError /* Handler for SQL error */
(char *locMsg /* - in: Call location */
);

int main /* DSN8ED9 driver */
(int argc, /* - Input argument count */
 char *argv[] /* - Input argument vector */
)
/***
* Get input parms, pass them to DSN8ES3, and process the results *
***/
{ printf("**** DSN8ED9: Sample client for DB2 PSM "
 "Stored Procedure Sample (DSN8ES3)\n\n");

 /***
* Extract the following information from the call parms: *
* (1) The schema of the stored procedure *
* (2) The name of the stored procedure *
* (3) Optional: The name of the location where the stored proc *
* resides *
***/
getCallParms(argc,argv);

/***
* Connect to location where the stored procedure resides *
***/
if(rc < RETSEV && strlen(hvLocationName) > 0)
 connectToLocation();

if(rc < RETSEV)
 callDSN8ES3();

if(rc < RETSEV && resultSetReturned == Yes)
 processDSN8ES3resultSet();

return(rc);
} /* end main */

void getCallParms /* Process args to call parms */
(int argc, /* - Input argument count */
 char *argv[] /* - Input argument vector */
)
/***
* Verifies that correct call parms have been passed in: *
* - Two parameters (the schema and the name of a stored procedure) *
* require an argument *
* - The third parameter (location name) is optional *
***/
{ if(argc < 3 || argc > 4)
 { issueInvalidCallParmCountError(argc);
 }
 else if(strlen(argv[1]) < 1 || strlen(argv[1]) > 130)
 { issueInvalidParmLengthError("Stored procedure schema",
 1,130);
 }
 else if(strlen(argv[2]) < 1 || strlen(argv[1]) > 130)
 { issueInvalidParmLengthError("Stored procedure name",
 1,130);
 }
 else
 { strcpy(hvSpSchema, argv[1]);
 strcpy(hvSpName, argv[2]);
 }
 if(argc > 3)
 { if(strlen(argv[3]) < 1 || strlen(argv[3]) > 16)
 { issueInvalidParmLengthError("Server Location Name",1,16);
 }
 else
 { strcpy(hvLocationName,argv[3]);
 }
 }
 else
 { hvLocationName[0] = NULLCHAR;
 }
} /* end of getCallParms */

void connectToLocation(void) /* Connect to DB2 location */
/***

```

```

* Connects to the DB2 location specified in call parm number 3 *
*****{* *****
{ EXEC SQL
 CONNECT TO :hvLocationName;

 if(SQLCODE != 0)
 { issueSqlError("Connect to location failed");
 }
} /* end of connectToLocation */

void callDSN8ES3(void) /* Run sample native SQL proc */
*****{* *****
* Calls the DSN8ES3 (sample native SQL procedure) *
*****{* *****
{ printf("\n");
 printf("-> Now requesting CREATE PROCEDURE statement for %s.%s\n",
 hvSpSchema, hvSpName);

 EXEC SQL CALL DSN8.DSN8ES3(:hvSpSchema :niSpSchema,
 :hvSpName :niSpName);

 *****{* *****
* Analyze status codes from DSN8ES3 *
*****{* *****
if(SQLCODE == 466)
 { resultSetReturned = Yes;
 }
else if(SQLCODE == 0)
 { resultSetReturned = No;
 printf("\n");
 printf("-> Call to DSN8ES3 succeeded "
 "but returned no result\n");
 }
else
 { issueSqlError("Call to DSN8ES3 failed");
 }
} /* end of callDSN8ES3 */

void processDSN8ES3resultSet(void) /* Handle DSN8ES3 result set */
*****{* *****
* Outputs data from the result set returned by DSN8ES3 *
*****{* *****
{
 *****{* *****
* Associate a locator with the result set from DSN8ES3 *
*****{* *****
associateResultSetLocator();

 *****{* *****
* Allocate a cursor for the result set *
*****{* *****
if(rc < RETSEV)
 allocateResultSetCursor();

 *****{* *****
* Output data from the result set *
*****{* *****
if(rc < RETSEV)
 writeDSN8ES3results();

} /* end of processDSN8ES3resultSet */

void associateResultSetLocator(void) /* Associate DSN8ES3 RS locator*/
*****{* *****
* Associates the result set from DSN8ES3 with a result set locator *
*****{* *****
{ EXEC SQL
 ASSOCIATE
 LOCATORS(:DSN8ES3_rs_loc)
 WITH PROCEDURE DSN8.DSN8ES3;

 if(SQLCODE != 0)
 { issueSqlError("Associate locator call failed");
 }
} /* end of associateResultSetLocator */

```

```

void allocateResultSetCursor(void) /* Alloc DSN8ES3 RS cursor */
{
 /* ****
 * Allocates a cursor to the locator for the DSN8ES3 result set
 * ****
 { EXEC SQL
 ALLOCATE DSN8ES3_RS_CSR
 CURSOR FOR RESULT SET :DSN8ES3_rs_loc;

 if(SQLCODE != 0)
 { issueSqlError("Allocate result set cursor call failed");
 }

 } /* end of allocateResultSetCursor */

void writeDSN8ES3results(void) /* Print DSN8ES3 results */
{
 /* ****
 * Outputs the results returned in the result set from DSN8ES3
 * ****
 { /* ****
 * Get the first entry in the result set
 * ****
 fetchFromResultSetCursor();

 /* ****
 * Process all rows in the result set
 * ****
 while(SQLCODE == 0 && rc < RETSEV)
 { printf("%s\n", hvLine);

 if(rc < RETSEV)
 { fetchFromResultSetCursor();
 }

 } /* end of writeDSN8ES3results */

void fetchFromResultSetCursor(void) /* Read DSN8ES3 RS cursor */
{
 /* ****
 * Reads the cursor for the DSN8ES3 result set
 * ****
 { memset(hvLine, ' ', 80); /*d176357*/
 EXEC SQL
 FETCH DSN8ES3_RS_CSR
 INTO :hvSequence,
 :hvLine;

 if(SQLCODE != 0 && SQLCODE != 100 && rc < RETSEV)
 { issueSqlError("*** Fetch from result set cursor failed");
 }

 } /* end of fetchFromResultSetCursor */

void issueInvalidCallParmCountError /* Handler for parm count err */
(int argc /* - in: no. parms received */
)
{
 /* ****
 * Called when this program is invoked with an inappropriate number
 * of call parms.
 * ****
 { printf("ERROR: DSN8ED9 was invoked with %i parameters\n", --argc);
 printf(" - The first two parms (schema and name"
 "of a stored procedure) are required\n");
 printf(" - The third parm (location name)"
 "is optional\n");
 printf("-----> Processing halted\n");
 rc = RETSEV;
 } /* end of issueInvalidCallParmCountError */

void issueInvalidParmLengthError /* Handler for parm len error */
(char *parmName, /* - in: identify of parm */
 int minLength, /* - in: min valid length */
 int maxLength /* - in: max valid length */
)
{
 /* ****
 * Called when the length of an argument specified for a DSN8ES3
 * parameter (parmName) does not fall within the valid bounds for
 * size (minLength and maxLength) for that parameter
 * ****

```

```

{ printf("ERROR: The length of the argument specified for the %s "
 "parameter\n",parmName);
 printf(" does not fall within the required bounds of %i "
 "and %i\n",minLength,maxLength);
 printf("-----> Processing halted\n");
 rc = RETSEV;
} /* end of issueInvalidParmLengthError */

#pragma linkage(dsntiar, OS)
void issueSqlError /* Handler for SQL error */
(char *locMsg /* - in: Call location */
)
/***
* Called when an unexpected SQLCODE is returned from a DB2 call *
***/
{ struct error_struct { /* DSNTIAR message structure */
 short int error_len;
 char error_text[10][80];
} error_message = {10 * 80};

extern short int dsntiar(struct sqlca *sqlca,
 struct error_struct *msg,
 int *len);

short int DSNTIARrc; /* DSNTIAR Return code */
int j; /* Loop control */
static int lrecl = 80; /* Width of message lines */

/***
* print the locator message *
***/
printf("ERROR: %-80s\n", locMsg);
printf("-----> Processing halted\n");

/***
* format and print the SQL message *
***/
DSNTIARrc = dsntiar(&sqlca, &error_message, &lrecl);
if(DSNTIARrc == 0)
 for(j = 0; j <= 10; j++)
 printf("%.80s\n", error_message.error_text[j]);
else
{
 printf(" *** ERROR: DSNTIAR could not format the message\n");
 printf(" *** SQLCODE is %d\n",SQLCODE);
 printf(" *** SQLERRM is \n");
 for(j=0; j<sqlca.sqlerrml; j++)
 printf("%c", sqlca.sqlerrmc[j]);
 printf("\n");
}

/***
* set severe error code *
***/
rc = RETSEV;

} /* end of issueSqlError */

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8ES3

Acepta el esquema y nombre del procedimiento almacenado externo y devuelve un conjunto de resultados que contiene la sentencia CREATE PROCEDURE.

```

-- DSN8ES3: SOURCE MODULE FOR THE SAMPLE NATIVE SQL PROCEDURE 00010000
-- 00020000
-- LICENSED MATERIALS - PROPERTY OF IBM 00022000
-- 5635-DB2 00024000
-- (C) COPYRIGHT 2006 IBM CORP. ALL RIGHTS RESERVED. 00026000
-- 00028000
-- STATUS = VERSION 9 00030000
-- 00040000
-- Function: Accepts the schema and name of an external stored 00050000
-- procedure and returns a result set that contains the 00060000

```

```

-- CREATE PROCEDURE statement. 00070000
-- Notes: 00080000
-- Dependencies: 00090000
-- - Requires support for native SQL procedures 00100000
-- - Requires a global temporary table (created in sample job 00110000
-- DSNTEJ66) for returning the result. 00120000
-- Restrictions: 00130000
-- 00140000
-- Module Type: SQL Procedure 00150000
-- Processor: DB2 for z/OS Version 9 00160000
-- or a subsequent release 00170000
-- 00180000
-- 00190000
-- 00200000
-- Entry Point: DSN8ES3 00210000
-- Purpose: See Function, above 00220000
-- 00230000
-- Parameters: 00240000
-- - Input: spSCHEMA VARCHAR(128) 00250000
-- spNAME VARCHAR(128) 00260000
-- - Output: (None) 00270000
-- 00280000
-- Normal Exit: 00290000
-- Error Exit: 00300000
-- 00310000
-- 00320000
-- External References: 00330000
-- - SYSIBM.SYSROUTINES : DB2 catalog table for routines 00340000
-- - SYSIBM.SYSPARMS : DB2 catalog table for routine parameters 00350000
-- - DSN8.DSN8ES3_RS_TBL: Global Temporary Table for result set 00360000
-- 00370000
-- Pseudocode: 00380000
-- - Clear any residual from result set table 00390000
-- - Get the stored proc properties from SYSIBM.SYSROUTINES 00400000
-- - If not found, return SQLSTATE 38602 and the message: 00410000
-- 'Requested object not found' 00420000
-- - If not a stored proc, return SQLSTATE 38603 and the message: 00430000
-- 'Object is not a stored procedure' 00440000
-- - If not an external stored proc, return SQLSTATE 38604 and the 00450000
-- message: 'Object is not an external stored procedure' 00460000
-- - Open a cursor on the SYSPARMS table 00470000
-- - Fetch the first row 00480000
-- - If a row is found, insert the CREATE PROCEDURE clause in the 00490000
-- result set 00500000
-- - For each row in the SYSPARMS cursor, build a parameter clause: 00510000
-- - Start with the parameter type (IN, OUT, or INOUT) 00520000
-- - Append the parameter name 00530000
-- - Append the parameter data type 00540000
-- - For string data types, add the CCSID clause 00550000
-- - Insert the entry in the result set table 00560000
-- - Build the remaining clauses and insert each in the result set 00570000
-- - Build and insert the RESULTS SETS clause 00580000
-- - Build and insert the EXTERNAL NAME clause 00590000
-- - Build and insert the LANGUAGE clause 00600000
-- - Build and insert the SQL data access type clause 00610000
-- - Build and insert the PARAMETER STYLE clause 00620000
-- - Build and insert the DETERMINISTIC clause 00630000
-- - Build and insert the FENCED clause 00640000
-- - Build and insert the COLLID clause 00650000
-- - Build and insert the WLM ENVIRONMENT clause 00660000
-- - Build and insert the ASUTIME clause 00670000
-- - Build and insert the STAY RESIDENT clause 00680000
-- - Build and insert the PROGRAM TYPE clause 00690000
-- - Build and insert the EXTERNAL SECURITY clause 00700000
-- - Build and insert the AFTER FAILURE clause 00710000
-- - Build and insert the RUN OPTIONS clause 00720000
-- - Build and insert the COMMIT ON RETURN clause 00730000
-- - Build and insert the SPECIAL REGISTERS clause 00740000
-- - Build and insert the CALLED ON NULL INPUT clause 00750000
-- - Open the cursor to the result set 00760000
-- 00762000
-- CHANGE ACTIVITY 00764000
-- 10/31/2013 Ignore SYSPARMS rows where ORDINAL = 0 PM98341 00766000
-- 00768000
-- 00770000
-- CREATE PROCEDURE DSN8.DSN8ES3 00780000
-- (IN spSCHEMA VARCHAR(128), 00790000
-- IN spNAME VARCHAR(128)) 00800000
-- PARAMETER CCSID EBCDIC 00810000
-- RESULT SET 1 00820000
-- NOT DETERMINISTIC 00830000
-- MODIFIES SQL DATA 00840000

```

```

 ASUTIME NO LIMIT 00850000
 COMMIT ON RETURN NO 00860000
 00870000
P1: BEGIN NOT ATOMIC 00880000
DECLARE hvLANGUAGE VARCHAR(24) CCSID EBCDIC; 00890000
DECLARE hvCOLLID VARCHAR(128) CCSID EBCDIC; 00900000
DECLARE hvDETERMINISTIC VARCHAR(17) CCSID EBCDIC; 00910000
DECLARE hvNULL_CALL CHAR(1) CCSID EBCDIC; 00920000
DECLARE hvPARAMETER_STYLE VARCHAR(18) CCSID EBCDIC; 00930000
DECLARE hvFENCED CHAR(1) CCSID EBCDIC; 00940000
DECLARE hvASUTIME INTEGER DEFAULT 0; 00950000
DECLARE hvCOMMIT_ON_RETURN VARCHAR(3) CCSID EBCDIC; 00960000
DECLARE hvEXTERNAL_NAME VARCHAR(762) CCSID EBCDIC; 00970000
DECLARE hvEXTERNAL_SECURITY VARCHAR(7) CCSID EBCDIC; 00980000
DECLARE hvMAX_FAILURE SMALLINT DEFAULT 0; 00990000
DECLARE hvORIGIN CHAR(1) CCSID EBCDIC; 01000000
DECLARE hvPROGRAM_TYPE VARCHAR(4) CCSID EBCDIC; 01010000
DECLARE hvRESULT_SETS SMALLINT DEFAULT 0; 01020000
DECLARE hvROUTINETYPE CHAR(1) CCSID EBCDIC; 01030000
DECLARE hvRUNOPTS VARCHAR(762) CCSID EBCDIC; 01040000
DECLARE hvSPECIAL_REGS VARCHAR(25) CCSID EBCDIC; 01050000
DECLARE hvSQL_DATA_ACCESS VARCHAR(17) CCSID EBCDIC; 01060000
DECLARE hvSTAYRESIDENT VARCHAR(3) CCSID EBCDIC; 01070000
DECLARE hvWLM_ENVIRONMENT VARCHAR(54) CCSID EBCDIC; 01080000
 01090000
DECLARE hvENCODING_SCHEME VARCHAR(7) CCSID EBCDIC; 01100000
DECLARE hvLENGTH INTEGER DEFAULT 0; 01110000
DECLARE hvORDINAL SMALLINT DEFAULT 0; 01120000
DECLARE hvPARMNAME VARCHAR(128) CCSID EBCDIC; 01130000
DECLARE hvROWTYPE VARCHAR(6) CCSID EBCDIC; 01140000
DECLARE hvSCALE SMALLINT DEFAULT 0; 01150000
DECLARE hvSUBTYPE VARCHAR(15) CCSID EBCDIC; 01160000
DECLARE hvTYPENAME VARCHAR(128) CCSID EBCDIC; 01170000
 01180000
DECLARE RETURN_POINT CHAR(4) CCSID EBCDIC; 01190000
 01200000
DECLARE LINE VARCHAR(384) CCSID EBCDIC; 01210000
DECLARE LINE_LENGTH INT DEFAULT 0; 01220000
DECLARE END_TABLE INT DEFAULT 0; 01230000
 01240000
DECLARE OPERATION VARCHAR(12) CCSID EBCDIC; 01250000
 01260000
DECLARE ROW CHAR(80) CCSID EBCDIC; 01270000
DECLARE ROW_SEQUENCE SMALLINT DEFAULT 1; 01280000
 01290000
-- Cursor for result set (CREATE PROCEDURE statement) 01300000
DECLARE DSN8ES3_RS_CSR CURSOR WITH RETURN WITH HOLD FOR 01310000
 SELECT RS_SEQUENCE,
 RS_LINE
 FROM DSN8.DSN8ES3_RS_TBL
 ORDER BY RS_SEQUENCE; 01320000
 01330000
 01340000
 01350000
 01360000
-- Cursor to fetch proc parm properties from SYSIBM.SYSPARMS 01370000
DECLARE SYSPARMS_CURSOR CURSOR FOR 01380000
 SELECT PARMNAME
 ,CASE ROWTYPE
 WHEN 'P' THEN 'IN'
 WHEN 'O' THEN 'OUT'
 WHEN 'B' THEN 'INOUT'
 END
 ,ORDINAL
 ,TYPENAME
 ,LENGTH
 ,SCALE
 ,CASE SUBTYPE
 WHEN 'B' THEN 'FOR BIT DATA'
 WHEN 'M' THEN 'FOR MIXED DATA'
 WHEN 'S' THEN 'FOR SBCS DATA'
 WHEN '' THEN ''
 END
 ,CASE ENCODING_SCHEME
 WHEN 'A' THEN 'ASCII'
 WHEN 'E' THEN 'EBCDIC'
 WHEN 'U' THEN 'UNICODE'
 WHEN '' THEN ''
 END
 FROM SYSIBM.SYSPARMS
 WHERE SCHEMA = spSCHEMA
 AND SPECIFICNAME = spNAME
 AND ORDINAL <> 0
 ORDER BY ORDINAL
 FOR FETCH ONLY; 01590000
 01600000
 01610000
 01620000
 01630000
 01635000
 01640000
 01650000

```

```

DECLARE CONTINUE HANDLER FOR NOT FOUND 01660000
 SET END_TABLE = 1; 01670000
 01680000
 01690000
 01700000
DECLARE EXIT HANDLER FOR SQLEXCEPTION 01710000
 SIGNAL SQLSTATE '38601'
 SET MESSAGE_TEXT = 'Unexpected SQLCODE '
 || CHAR(SQLCODE)
 || ' from '
 || OPERATION;
 01720000
 01730000
 01740000
 01750000
 01760000
-- Clean residual from the result set table 01770000
DELETE FROM DSN8.DSN8ES3_RS_TBL; 01780000
 01790000
-- Fetch the stored proc properties from SYSIBM.SYSROUTINES 01800000
SET END_TABLE = 0; 01810000
SET OPERATION = 'SELECT INTO';
SELECT LANGUAGE 01820000
 ,COLLID 01830000
 ,CASE DETERMINISTIC 01840000
 WHEN 'N' THEN 'NOT DETERMINISTIC' 01850000
 WHEN 'Y' THEN 'DETERMINISTIC' 01860000
 WHEN ' ' THEN ' '
 END 01870000
 ,NULL_CALL 01880000
 ,CASE PARAMETER_STYLE 01890000
 WHEN 'D' THEN 'DB2SQL' 01900000
 WHEN 'G' THEN 'GENERAL' 01910000
 WHEN 'N' THEN 'GENERAL WITH NULLS' 01920000
 WHEN 'J' THEN 'JAVA' 01930000
 WHEN ' ' THEN ' '
 END 01940000
 ,FENCED 01950000
 ,CASE SQL_DATA_ACCESS 01960000
 WHEN 'C' THEN 'CONTAINS SQL' 01970000
 WHEN 'M' THEN 'MODIFIES SQL DATA' 01980000
 WHEN 'N' THEN 'NO SQL' 01990000
 WHEN 'R' THEN 'READS SQL DATA' 02000000
 WHEN ' ' THEN ' '
 END 02010000
 ,CASE STAYRESIDENT 02020000
 WHEN 'N' THEN 'NO' 02030000
 WHEN 'Y' THEN 'YES' 02040000
 WHEN ' ' THEN ' '
 END 02050000
 ,ASUTIME 02060000
 ,WLM_ENVIRONMENT 02070000
 ,CASE PROGRAM_TYPE 02080000
 WHEN 'M' THEN 'MAIN' 02090000
 WHEN 'S' THEN 'SUB' 02100000
 WHEN ' ' THEN ' '
 END 02110000
 ,CASE EXTERNAL_SECURITY 02120000
 WHEN 'D' THEN 'DB2' 02130000
 WHEN 'U' THEN 'USER' 02140000
 WHEN 'C' THEN 'DEFINER' 02150000
 WHEN ' ' THEN ' '
 END 02160000
 ,CASE COMMIT_ON_RETURN 02170000
 WHEN 'N' THEN 'NO' 02180000
 WHEN 'Y' THEN 'YES' 02190000
 WHEN ' ' THEN ' '
 END 02200000
 ,CASE RESULT_SETS 02210000
 ,EXTERNAL_NAME 02220000
 ,RUNOPTS 02230000
 ,CASE SPECIAL_REGS 02240000
 WHEN 'D' THEN 'DEFAULT SPECIAL REGISTERS' 02250000
 WHEN 'I' THEN 'INHERIT SPECIAL REGISTERS' 02260000
 WHEN ' ' THEN ' '
 END 02270000
 ,MAX_FAILURE 02280000
INTO hvLANGUAGE 02290000
 ,hvCOLLID 02300000
 ,hvDETERMINISTIC 02310000
 ,hvNULL_CALL 02320000
 ,hvPARAMETER_STYLE 02330000
 ,hvFENCED 02340000
 ,hvSQL_DATA_ACCESS 02350000
 ,hvSTAYRESIDENT 02360000
 ,hvASUTIME 02370000
 ,hvWLM_ENVIRONMENT 02380000
 02390000
 02400000
 02410000
 02420000
 02430000
 02440000
 02450000
 02460000
 02470000

```

```

, hvPROGRAM_TYPE 02480000
, hvEXTERNAL_SECURITY 02490000
, hvCOMMIT_ON_RETURN 02500000
, hvRESULT_SETS 02510000
, hvEXTERNAL_NAME 02520000
, hvRUNOPTS 02530000
, hvSPECIAL_REGS 02540000
, hvMAX_FAILURE 02550000
FROM SYSIBM.SYSROUTINES 02560000
WHERE SCHEMA = spSCHEMA 02570000
AND NAME = spNAME; 02580000
 02590000
 02600000
CASE
WHEN END_TABLE = 1 THEN 02610000
 SIGNAL SQLSTATE '38602'
 SET MESSAGE_TEXT = 'Requested object "' 02620000
 || spSCHEMA 02630000
 || '.'. '
 || spNAME 02640000
 || '" not found';
WHEN hvROUTINETYPE <> 'P' THEN 02650000
 SIGNAL SQLSTATE '38603'
 SET MESSAGE_TEXT = 'Object is not a stored procedure'; 02660000
WHEN hvORIGIN <> 'E' THEN 02670000
 SIGNAL SQLSTATE '38604'
 SET MESSAGE_TEXT = 'Object is not an external stored procedure'; 02680000
ELSE -- NOOP below provided to satisfy requirement for ELSE clause
 SET ROW_SEQUENCE = ROW_SEQUENCE; 02690000
END CASE; 02700000
SET END_TABLE = 0; 02710000
SET OPERATION = 'OPEN CURSOR'; 02720000
OPEN SYSPARMS_CURSOR; 02730000
 02740000
SET OPERATION = 'FIRST FETCH'; 02750000
FETCH SYSPARMS_CURSOR 02760000
 INTO hvPARMNAME
 ,hvROWTYPE
 ,hvORDINAL
 ,hvTYPENAME
 ,hvLENGTH
 ,hvSCALE
 ,hvSUBTYPE
 ,hvENCODING_SCHEME;
 02770000
-- Output the CREATE PROCEDURE clause
IF END_TABLE = 0 THEN 02780000
 SET LINE = 'CREATE PROCEDURE ' || spSCHEMA || '.' || spNAME; 02790000
 SET RETURN_POINT = 'A100';
 GOTO INSERTLINE;
END IF;
 02800000
A100: -- Build and output the parameter list
SET LINE = ' ('; 02810000
WHILE END_TABLE = 0 DO
 -- Output the parameter type (IN, OUT, or INOUT)
 SET LINE = LINE
 || hvROWTYPE || ' '
 || hvPARMNAME || ' '
 || RTRIM(hvTYPENAME); 02820000
CASE
WHEN hvTYPENAME = 'DECIMAL' 02830000
 OR hvTYPENAME = 'DEC' 02840000
 OR hvTYPENAME = 'NUMERIC' THEN 02850000
 SET LINE = LINE || '(' || VARCHAR(hvLENGTH)
 || ',' || VARCHAR(hvSCALE) || ')';
 02860000
WHEN hvTYPENAME = 'FLOAT' THEN 02870000
 SET LINE = LINE || '(' || VARCHAR(hvLENGTH) || ')';
 02880000
WHEN hvTYPENAME = 'CHARACTER' 02890000
 OR hvTYPENAME = 'CHAR' 02900000
 OR hvTYPENAME = 'CHARACTER VARYING' 02910000
 OR hvTYPENAME = 'CHAR VARYING' 02920000
 OR hvTYPENAME = 'VARCHAR' 02930000
 OR hvTYPENAME = 'CHARACTER LARGE OBJECT' 02940000
 OR hvTYPENAME = 'CHAR LARGE OBJECT' 02950000
 OR hvTYPENAME = 'CLOB' 02960000
 OR hvTYPENAME = 'GRAPHIC' 02970000
 OR hvTYPENAME = 'VARGRAPHIC' 02980000
 OR hvTYPENAME = 'DBCLOB' 02990000
 03000000
 03010000
SET LINE = ''; 03020000
WHILE END_TABLE = 0 DO
 -- Output the parameter type (IN, OUT, or INOUT)
 SET LINE = LINE
 || hvROWTYPE || ' '
 || hvPARMNAME || ' '
 || RTRIM(hvTYPENAME); 03030000
CASE
WHEN hvTYPENAME = 'DECIMAL' 03040000
 OR hvTYPENAME = 'DEC' 03050000
 OR hvTYPENAME = 'NUMERIC' THEN 03060000
 SET LINE = LINE || '(' || VARCHAR(hvLENGTH)
 || ',' || VARCHAR(hvSCALE) || ')';
 03070000
WHEN hvTYPENAME = 'FLOAT' THEN 03080000
 SET LINE = LINE || '(' || VARCHAR(hvLENGTH) || ')';
 03090000
WHEN hvTYPENAME = 'CHARACTER' 03100000
 OR hvTYPENAME = 'CHAR' 03110000
 OR hvTYPENAME = 'CHARACTER VARYING' 03120000
 OR hvTYPENAME = 'CHAR VARYING' 03130000
 OR hvTYPENAME = 'VARCHAR' 03140000
 OR hvTYPENAME = 'CHARACTER LARGE OBJECT' 03150000
 OR hvTYPENAME = 'CHAR LARGE OBJECT' 03160000
 OR hvTYPENAME = 'CLOB' 03170000
 OR hvTYPENAME = 'GRAPHIC' 03180000
 OR hvTYPENAME = 'VARGRAPHIC' 03190000
 OR hvTYPENAME = 'DBCLOB' 03200000
 03210000
SET LINE = ''; 03220000
WHILE END_TABLE = 0 DO
 -- Output the parameter type (IN, OUT, or INOUT)
 SET LINE = LINE
 || hvROWTYPE || ' '
 || hvPARMNAME || ' '
 || RTRIM(hvTYPENAME); 03230000
CASE
WHEN hvTYPENAME = 'DECIMAL' 03240000
 OR hvTYPENAME = 'DEC' 03250000
 OR hvTYPENAME = 'NUMERIC' THEN 03260000
 SET LINE = LINE || '(' || VARCHAR(hvLENGTH)
 || ',' || VARCHAR(hvSCALE) || ')';
 03270000
WHEN hvTYPENAME = 'FLOAT' THEN 03280000
 SET LINE = LINE || '(' || VARCHAR(hvLENGTH) || ')';
 03290000

```

```

 OR hvTYPENAME = 'BINARY LARGE OBJECT' 03300000
 OR hvTYPENAME = 'BLOB' THEN 03310000
 SET LINE = LINE || '(' || VARCHAR(hvLENGTH) || ')';
 ELSE -- busy statement below required to handle ELSE case 03320000
 SET ROW_SEQUENCE = ROW_SEQUENCE; 03330000
 END CASE; 03340000
 03350000
 03360000
 03370000
 IF hvSUBTYPE <> '' THEN 03380000
 SET LINE = LINE || hvSUBTYPE; 03390000
 END IF; 03400000
 IF hvENCODING_SCHEME <> '' THEN 03410000
 SET LINE = LINE || ' CCSID' || RTRIM(hvENCODING_SCHEME); 03420000
 END IF; 03430000
 SET RETURN_POINT = 'B100'; 03440000
 GOTO INSERTLINE; 03450000
 03460000
B100: -- Fetch the next parameter 03470000
SET OPERATION = 'FETCH'; 03480000
FETCH SYSPARMS_CURSOR 03490000
 INTO hvPARMNAME 03500000
 ,hvROWTYPE 03510000
 ,hvORDINAL 03520000
 ,hvTYPENAME 03530000
 ,hvLENGTH 03540000
 ,hvSCALE 03550000
 ,hvSUBTYPE 03560000
 ,hvENCODING_SCHEME; 03570000
 03580000
 SET LINE = ' ,';
END WHILE; 03590000
 03600000
 03610000
SET OPERATION = 'CLOSE CURSOR'; 03620000
CLOSE SYSPARMS_CURSOR; 03630000
-- Close the parameter list 03640000
SET LINE = ')';
SET RETURN_POINT = 'C100'; 03650000
GOTO INSERTLINE; 03660000
 03670000
 03680000
C100: -- Build remaining clauses for the CREATE PROCEDURE statement 03690000
03700000
-- Output the RESULTS SETS clause 03710000
IF hvRESULT_SETS > 0 THEN 03720000
 SET LINE = 'DYNAMIC RESULT SETS ' || VARCHAR(hvRESULT_SETS); 03730000
 SET RETURN_POINT = 'D100';
 GOTO INSERTLINE;
END IF; 03740000
 03750000
 03760000
 03770000
D100: -- Output the EXTERNAL NAME clause 03780000
SET LINE = 'EXTERNAL NAME ' || RTRIM(hvEXTERNAL_NAME); 03790000
SET RETURN_POINT = 'E100';
GOTO INSERTLINE; 03800000
 03810000
 03820000
E100: -- Output the LANGUAGE clause 03830000
SET LINE = 'LANGUAGE ' || RTRIM(hvLANGUAGE);
SET RETURN_POINT = 'F100';
GOTO INSERTLINE; 03840000
 03850000
 03860000
 03870000
F100: -- Output the SQL data access type clause 03880000
IF hvSQL_DATA_ACCESS <> '' THEN 03890000
 SET LINE = hvSQL_DATA_ACCESS;
 SET RETURN_POINT = 'G100';
 GOTO INSERTLINE;
END IF; 03900000
 03910000
 03920000
 03930000
 03940000
G100: -- Output the PARAMETER STYLE clause 03950000
IF hvPARAMETER_STYLE <> '' THEN 03960000
 SET LINE = 'PARAMETER STYLE ' || hvPARAMETER_STYLE;
 SET RETURN_POINT = 'H100';
 GOTO INSERTLINE;
END IF; 03970000
 03980000
 03990000
 04000000
 04010000
H100: -- Output the DETERMINISTIC clause 04020000
IF hvDETERMINISTIC <> '' THEN 04030000
 SET LINE = hvDETERMINISTIC;
 SET RETURN_POINT = 'I100';
 GOTO INSERTLINE;
END IF; 04040000
 04050000
 04060000
 04070000
 04080000
I100: -- Output the FENCED clause 04090000
IF hvFENCED <> '' THEN 04100000
 SET LINE = 'FENCED';

```

```

 SET RETURN_POINT = 'J100';
 GOTO INSERTLINE;
END IF;

J100: -- Output the COLLID clause
IF hvCOLLID <> '' THEN
 SET LINE = 'COLLID ' || RTRIM(hvCOLLID);
ELSE
 SET LINE = 'NO COLLID';
END IF;
SET RETURN_POINT = 'K100';
GOTO INSERTLINE;

K100: -- Output the WLM ENVIRONMENT clause
SET LINE = 'WLM ENVIRONMENT ' || RTRIM(hvWLM_ENVIRONMENT);
SET RETURN_POINT = 'L100';
GOTO INSERTLINE;

L100: -- Output the ASUTIME clause
IF hvASUTIME <> 0 THEN
 SET LINE = 'ASUTIME ' || VARCHAR(hvASUTIME);
ELSE
 SET LINE = 'ASUTIME NO LIMIT';
END IF;
SET RETURN_POINT = 'M100';
GOTO INSERTLINE;

M100: -- Output the STAY RESIDENT clause
IF hvSTAYRESIDENT <> '' THEN
 SET LINE = 'STAY RESIDENT ' || hvSTAYRESIDENT;
 SET RETURN_POINT = 'N100';
 GOTO INSERTLINE;
END IF;

N100: -- Output the PROGRAM TYPE clause
IF hvPROGRAM_TYPE <> '' THEN
 SET LINE = 'PROGRAM TYPE ' || hvPROGRAM_TYPE;
 SET RETURN_POINT = 'O100';
 GOTO INSERTLINE;
END IF;

O100: -- Output the EXTERNAL SECURITY clause
IF hvEXTERNAL_SECURITY <> '' THEN
 SET LINE = 'SECURITY ' || hvEXTERNAL_SECURITY;
 SET RETURN_POINT = 'P100';
 GOTO INSERTLINE;
END IF;

P100: -- Output the AFTER FAILURE clause
IF hvMAX_FAILURE = -1 THEN
 SET LINE = 'STOP AFTER SYSTEM DEFAULT FAILURES';
ELSEIF hvMAX_FAILURE = 0 THEN
 SET LINE = 'CONTINUE AFTER FAILURE';
ELSE
 SET LINE = 'STOP AFTER ' || VARCHAR(hvMAX_FAILURE) || ' FAILURES';
END IF;
SET RETURN_POINT = 'Q100';
GOTO INSERTLINE;

Q100: -- Output the RUN OPTIONS clause
IF hvRUNOPTS <> '' THEN
 SET LINE = 'RUN OPTIONS ''' || hvRUNOPTS || '';
 SET RETURN_POINT = 'R100';
 GOTO INSERTLINE;
END IF;

R100: -- Output the COMMIT ON RETURN clause
IF hvCOMMIT_ON_RETURN <> '' THEN
 SET LINE = 'COMMIT ON RETURN ' || hvCOMMIT_ON_RETURN;
 SET RETURN_POINT = 'S100';
 GOTO INSERTLINE;
END IF;

S100: -- Output the SPECIAL REGISTERS clause
IF hvSPECIAL_REGS <> '' THEN
 SET LINE = hvSPECIAL_REGS;
 SET RETURN_POINT = 'T100';
 GOTO INSERTLINE;
END IF;

T100: -- Output the CALLED ON NULL INPUT clause
IF hvNULL_CALL = 'Y' THEN

```

```

 SET LINE = 'CALLED ON NULL INPUT';
 SET RETURN_POINT = 'U100';
 GOTO INSERTLINE;
 END IF;

 U100: -- Finish up
 GOTO DONE;

 INSERTLINE:
 SET LINE_LENGTH = LENGTH(LINE);
 WHILE LINE_LENGTH > 72 DO
 SET ROW = SUBSTR(LINE, 1, 72) || REPEAT(' ', 8);
 SET LINE = SUBSTR(LINE, 73, LINE_LENGTH-72);
 SET LINE_LENGTH = LENGTH(LINE);

 SET ROW_SEQUENCE = ROW_SEQUENCE + 1;
 INSERT INTO DSN8.DSN8ES3_RS_TBL
 (RS_SEQUENCE,
 RS_LINE)
 VALUES(P1.ROW_SEQUENCE,
 P1.ROW);
 END WHILE;

 SET ROW = SUBSTR((LINE || REPEAT(' ', 80)), 1, 80);
 SET ROW_SEQUENCE = ROW_SEQUENCE + 1;
 SET OPERATION = 'INSERT';
 INSERT INTO DSN8.DSN8ES3_RS_TBL
 (RS_SEQUENCE,
 RS_LINE)
 VALUES(P1.ROW_SEQUENCE,
 P1.ROW);

 CASE RETURN_POINT
 WHEN 'A100' THEN GOTO A100;
 WHEN 'B100' THEN GOTO B100;
 WHEN 'C100' THEN GOTO C100;
 WHEN 'D100' THEN GOTO D100;
 WHEN 'E100' THEN GOTO E100;
 WHEN 'F100' THEN GOTO F100;
 WHEN 'G100' THEN GOTO G100;
 WHEN 'H100' THEN GOTO H100;
 WHEN 'I100' THEN GOTO I100;
 WHEN 'J100' THEN GOTO J100;
 WHEN 'K100' THEN GOTO K100;
 WHEN 'L100' THEN GOTO L100;
 WHEN 'M100' THEN GOTO M100;
 WHEN 'N100' THEN GOTO N100;
 WHEN 'O100' THEN GOTO O100;
 WHEN 'P100' THEN GOTO P100;
 WHEN 'Q100' THEN GOTO Q100;
 WHEN 'R100' THEN GOTO R100;
 WHEN 'S100' THEN GOTO S100;
 WHEN 'T100' THEN GOTO T100;
 WHEN 'U100' THEN GOTO U100;
 ELSE GOTO DONE;
 END CASE;

 DONE:
 -- Open the cursor to the result set
 SET OPERATION = 'RS CURSOR';
 OPEN DSN8ES3_RS_CSR;
 END P1

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8DUAD

Devuelve la fecha actual en uno de estos 34 formatos.

```

/***** ****
* Module name = DSN8DUAD (DB2 sample program)
* DESCRIPTIVE NAME = Current date reformatter (UDF)
*
* LICENSED MATERIALS - PROPERTY OF IBM
* 5625-DB2
***** ****
* 00010000
* 00020000
* 00030000
* 00040000
* 00050000
* 00060000
* 00070000
* 00080000

```

```

* (C) COPYRIGHT 1998, 2003 IBM CORP. ALL RIGHTS RESERVED. * 00090000
* * 00100000
* STATUS = VERSION 8 * 00110000
* * 00120000
* * 00130000
* Function: Returns the current date in one these 34 formats: * 00140000
* * 00150000
* D MONTH YY D MONTH YYYY DD MONTH YY DD MONTH YYYY * 00160000
* D.M.YY D.M.YYYY DD.MM.YY DD.MM.YYYY * 00170000
* D-M-YY D-M-YYYY DD-MM-YY DD-MM-YYYY * 00180000
* D/M/YY D/M/YYYY DD/MM/YY DD/MM/YYYY * 00190000
* M/D/YY M/D/YYYY MM/DD/YY MM/DD/YYYY * 00200000
* YY/M/D YYYY/M/D YY/MM/DD YYYY/MM/DD * 00210000
* YY.M.D YYYY.M.D YY.MM.DD YYYY.MM.DD * 00220000
* YYYY-YY YYYY-M-D YYYY-MM-DD YYYY-MM-DD * 00230000
* YYYY-XX YYYY-DD-XX YYYY-DD-XX YYYY-DD-XX * 00240000
* YYYY-XX-D YYYY-XX-DD YYYY-XX-DD YYYY-XX-DD * 00250000
* * 00260000
* where: * 00270000
* * 00280000
* D: Suppress leading zero if the day is less than 10 * 00290000
* DD: Retain leading zero if the day is less than 10 * 00300000
* M: Suppress leading zero if the month is less than 10 * 00310000
* MM: Retain leading zero if the month is less than 10 * 00320000
* MONTH: Use English-language name of month * 00330000
* XX: Use a capital Roman numeral for month * 00340000
* XX: Use a capital Roman numeral for month * 00350000
* YY: Use non-century year format * 00360000
* YYYY: Use century year format * 00370000
* * 00380000
* Example invocation: * 00390000
* EXEC SQL SET :today = ALTDATE("DD MONTH YY"); * 00400000
* * 00410000
* Notes: * 00420000
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher * 00430000
* * 00440000
* Restrictions: * 00450000
* * 00460000
* Module type: C program * 00470000
* Processor: IBM C/C++ for OS/390 V1R3 or higher * 00480000
* Module size: See linkedit output * 00490000
* Attributes: Re-entrant and re-usable * 00500000
* * 00510000
* Entry Point: DSN8DUAD * 00520000
* Purpose: See Function * 00530000
* Linkage: DB2SQL * 00540000
* Invoked via SQL UDF call * 00550000
* * 00560000
* Input: Parameters explicitly passed to this function: * 00570000
* - *format : pointer to a char[14], null-termi- * 00580000
* nated string having the desired * 00590000
* format for the current date (see * 00600000
* "Function", above, for valid formats) * 00610000
* - *niFormat : pointer to a short integer having * 00620000
* the null indicator variable for * 00630000
* *format. * 00640000
* - *fnName : pointer to a char[138], null-termi- * 00650000
* nated string having the UDF family * 00660000
* name of this function. * 00670000
* - *specificName: pointer to a char[129], null-termi- * 00680000
* nated string having the UDF specific * 00690000
* name of this function. * 00700000
* * 00710000
* * 00720000
* Output: Parameters explicitly passed by this function: * 00730000
* - *dateOut : pointer to a char[18], null-termi- * 00740000
* nated string to receive the current * 00750000
* date in the formatted indicated by * 00760000
* *format. * 00770000
* - *niDateOut : pointer to a short integer to re- * 00780000
* ceive the null indicator variable * 00790000
* for *dateOut. * 00800000
* - *sqlstate : pointer to a char[06], null-termi- * 00810000
* nated string to receive the SQLSTATE.* 00820000
* - *message : pointer to a char[70], null-termi- * 00830000
* nated string to receive a diagnostic * 00840000
* message if one is generated by this * 00850000
* function. * 00860000
* * 00870000
* Normal Exit: Return Code: SQLSTATE = 00000 * 00880000
* - Message: none * 00890000
* * 00900000

```

```

* Error Exit: Return Code: SQLSTATE = 38601 * 00910000
* - Message: DSN8DUAD Error: No output format entered * 00920000
* - Message: DSN8DUAD Error: Unknown format specified * 00930000
*
* External References: * 00940000
* - Routines/Services: None * 00950000
* - Data areas : None * 00960000
* - Control blocks : None * 00970000
* * 00980000
* * 00990000
* * 01000000
* * 01010000
* * 01020000
* Pseudocode: * 01030000
* DSN8DUAD: * 01040000
* - Verify that a valid format for the current date was received: * 01050000
* - if *format is blank or niFormat is not 0, no format passed: * 01060000
* - issue SQLSTATE 38601 and a diagnostic message * 01070000
* Verify that a valid format for the current date was received: * 01080000
* Call formatDate to convert the current date in the indicated
* format. * 01090000
* If no errors, unset null indicators, and return SQLSTATE 00000 * 01100000
* else set null indicator and return null date out. * 01110000
* End DSN8DUAD * 01120000
*
* formatDate: * 01130000
* - Use the date format to generate a specification string for
* the getDate function * 01140000
* - Call getDate * 01150000
* - Perform edits on the result as appropriate:
* - call Remove0 to strip leading zeroes from the day and/or
* month * 01160000
* - call romanMonth to convert the month to a roman numeral * 01170000
* - If *format is not one of the 34 supported formats: * 01180000
* - issue SQLSTATE 38601 and a diagnostic message * 01190000
* End formatDate * 01200000
*
* getDate: * 01210000
* - invoke the time() library function to query calendar time. * 01220000
* - invoke the localtime() library function to convert and correct
* for local time * 01230000
* - invoke the strftime() library function to format the date from
* from the time vector according to specification generated by
* the local formatDate() function. * 01240000
* End getDate * 01250000
*
* Remove0: * 01260000
* - check the passed string for a character zero in the passed
* location. * 01270000
* - if a zero is found, eliminate it by shifting all bytes to its
* right 1 byte leftward. * 01280000
* End Remove0 * 01290000
*
* romanMonth * 01300000
* - convert the month (01-12) to a roman numeral (I-XII). * 01310000
* End romanMonth * 01320000
*
* Change Log: * 01330000
* 2002/10/17 PQ66488 Fix date truncation error @01* 01340000
* * 01350000
* * 01360000
* * 01370000
* * 01380000
* * 01390000
* * 01400000
* * 01410000
* * 01420000
* * 01430000
* * 01440000
* * 01450000
* @01* 01460000
* * 01470000
* * 01480000
******/ 01490000
01500000
#pragma linkage(DSN8DUAD,fetchable) 01510000
01520000
***** C library definitions *****/
#include <stdio.h> 01530000
#include <string.h> 01540000
#include <time.h> 01550000
01560000
01570000
***** Equates *****/
#define NULLCHAR '\0' /* Null character */ 01580000
01590000
01600000
#define MATCH 0 /* Comparison status: Equal */ 01610000
#define NOT_OK 0 /* Run status indicator: Error */ 01620000
#define OK 1 /* Run status indicator: Good */ 01630000
01640000
01650000
***** DSN8DUAD functions *****/
void DSN8DUAD /* main routine */ 01660000
(char *format, /* in: format for dateOut */ 01670000
 char *dateOut, /* out: formatted current date */ 01680000
 short int *niFormat, /* in: indic var, format */ 01690000
 short int *niDateOut, /* out: indic var for dateOut */ 01700000
 char *sqlstate, /* out: SQLSTATE */ 01710000
 * 01720000

```

```

char *fnName, /* in: family name of function*/ 01730000
char *specificName, /* in: specific name of func */ 01740000
char *message, /* out: diagnostic message */ 01750000
);
 01760000
 01770000
int formatDate /* format the current date */ 01780000
(char *dateOut, /* out: formatted curr date */ 01790000
 char *message, /* out: diagnostic message */ 01800000
 char *sqlstate, /* out: SQLSTATE */ 01810000
 char *format, /* in: desired format */ 01820000
);
 01830000
 01840000
void getDate /* gets curr date, formatted */ 01850000
(char *dateOut, /* out: formatted current date*/ 01860000
 char *dateFmt, /* in: desired date format */ 01870000
);
 01880000
 01890000
void Remove0 /* remove 0 from indic. byte */ 01900000
(char *string, /* in/out: character string */ 01910000
 short int loc, /* in: loc'n of zero to remove*/ 01920000
);
 01930000
 01940000
char *romanMonth(); /* get roman# of curr month# */ 01950000
 01960000
 01970000
/*** 01980000
/** main routine **** 01990000
/*** 02000000
void DSN8DUAD /* main routine */ 02010000
(char *format, /* in: format for dateOut */ 02020000
 char *dateOut, /* out: formatted current date*/ 02030000
 short int *niFormat, /* in: indic var, format */ 02040000
 short int *niDateOut, /* out: indic var for dateOut */ 02050000
 char *sqlstate, /* out: SQLSTATE */ 02060000
 char *fnName, /* in: family name of function*/ 02070000
 char *specificName, /* in: specific name of func */ 02080000
 char *message, /* out: diagnostic message */ 02090000
)
 02100000
/*** 02110000
*
* Assumptions:
* - *format points to a char[14], null-terminated string 02120000
* - *dateOut points to a char[18], null-terminated string 02130000
* - *niFormat points to a short integer 02140000
* - *niDateOut points to a short integer 02150000
* - *sqlstate points to a char[06], null-terminated string 02160000
* - *fnName points to a char[138], null-terminated string 02170000
* - *specificName points to a char[129], null-terminated string 02180000
* - *message points to a char[70], null-terminated string 02190000
***** 02200000
{* 02210000
 02220000
 02230000
 02240000
***** local variables ***** 02250000
short int status = OK; /* DSN8DUAD run status */ 02260000
 02270000
 02280000
***** 02290000
* Verify that a format has been passed in 02300000
***** 02310000
if(*niFormat || (strlen(format) == 0)) 02320000
{
 status = NOT_OK;
 strcpy(message,
 "DSN8DUAD Error: No output format entered");
 strcpy(sqlstate, "38601");
}
 02330000
 02340000
 02350000
 02360000
 02370000
 02380000
 02390000
***** 02400000
* Call formatDate to format the current date according to format * 02410000
***** 02420000
if(status == OK) 02430000
 status = formatDate(dateOut, message, sqlstate, format);
 02440000
 02450000
 02460000
***** 02470000
* If formatting was successful, clear the message buffer and sql- * 02480000
* state, and unset the SQL null indicator for dateOut. * 02490000
***** 02500000
if(status == OK)
{
 *niDateOut = 0;
 message[0] = NULLCHAR;
}
 02510000
 02520000
 02530000
 02540000

```

```

 strcpy(sqlstate,"00000");
 }
/***** If errors occurred, clear the dateOut buffer and set the SQL null indicator. A diagnostic message and the SQLSTATE have been set where the error was detected. *****/
else
{
 dateOut[0] = NULLCHAR;
 *niDateOut = -1;
}

return;
} /* end of DSN8DUAD */

/***** functions *****/
int formatDate /* format the current date */ 02760000
(char *dateOut, /* out: formatted curr date */ 02770000
 char *message, /* out: diagnostic message */ 02780000
 char *sqlstate, /* out: SQLSTATE */ 02790000
 char *format /* in: desired format */ 02800000
)
/***** Places the current date formatted according to format in dateOut. */
* If processing is successful, formatDate returns OK. Otherwise, a diagnostic message is placed in message, the SQLSTATE is 38601, and formatDate returns NOT_OK.
***** */
{
 short int func_status = OK; /* function status indicator */ 02890000
 char dateFmt[14]; /* date format work buffer */ 02900000
 02910000
/***** get the current date and format it according to format */
if(strcmp(format,"D MONTH YY") == MATCH)
{
 getDate(dateOut,"%d %B %y"); /* format date as DD MONTH YY */ 02970000
 Remove0(dateOut,0); /* strip leading 0 if day < 10*/ 02980000
}
else if(strcmp(format,"D MONTH YYYY") == MATCH)
{
 getDate(dateOut,"%d %B %Y"); /* formt date as DD MONTH YYYY*/ 03020000
 Remove0(dateOut,0); /* strip leading 0 if day < 10*/ 03030000
}
else if(strcmp(format,"DD MONTH YY") == MATCH)
{
 getDate(dateOut,"%d %B %y"); /* format date as DD MONTH YY */ 03070000
}
else if(strcmp(format,"DD MONTH YYYY") == MATCH)
{
 getDate(dateOut,"%d %B %Y"); /* formt date as DD MONTH YYYY*/ 03090000
}
else if(strcmp(format,"D.M.YY") == MATCH)
{
 getDate(dateOut,"%d.%m.%y"); /* format date as DD.MM.YY */ 03150000
 Remove0(dateOut,3); /* strip leading 0 if month<10*/ 03160000
 Remove0(dateOut,0); /* strip leading 0 if day < 10*/ 03170000
}
else if(strcmp(format,"D.M.YYYY") == MATCH)
{
 getDate(dateOut,"%d.%m.%Y"); /* format date as DD.MM.YYYY */ 03210000
 Remove0(dateOut,3); /* strip leading 0 if month<10*/ 03220000
 Remove0(dateOut,0); /* strip leading 0 if day < 10*/ 03230000
}
else if(strcmp(format,"DD.MM.YY") == MATCH)
{
 getDate(dateOut,"%d.%m.%y"); /* format date as DD.MM.YY */ 03270000
}
else if(strcmp(format,"DD.MM.YYYY") == MATCH)
{
 getDate(dateOut,"%d.%m.%Y"); /* format date as DD.MM.YYYY */ 03310000
}
else if(strcmp(format,"D-M-YY") == MATCH)
{
 getDate(dateOut,"%d-%m-%y"); /* format date as DD-MM-YY */ 03350000
 Remove0(dateOut,3); /* strip leading 0 if month<10*/ 03360000
}

```

```

 Remove0(dateOut,0); /* strip leading 0 if day < 10*/ 03370000
 }
 else if(strcmp(format,"D-M-YYYY") == MATCH)
 {
 getDate(dateOut,"%d-%m-%Y"); /* format date as DD-MM-YYYY */03410000
 Remove0(dateOut,3); /* strip leading 0 if month<10*/ 03420000
 Remove0(dateOut,0); /* strip leading 0 if day < 10*/ 03430000
 }
 else if(strcmp(format,"DD-MM-YY") == MATCH) 03440000
 {
 getDate(dateOut,"%d-%m-%y"); /* format date as DD-MM-YY */ 03450000
 }
 else if(strcmp(format,"DD-MM-YYYY") == MATCH) 03460000
 {
 getDate(dateOut,"%d-%m-%Y"); /* format date as DD-MM-YYYY */03470000
 }
 else if(strcmp(format,"DD/MM-YYYY") == MATCH) 03480000
 {
 getDate(dateOut,"%d/%m/%Y"); /* format date as DD/MM/YY */ 03490000
 }
 else if(strcmp(format,"D/M/YY") == MATCH) 03500000
 {
 getDate(dateOut,"%d/%m/%y"); /* format date as DD/MM/YY */ 03510000
 Remove0(dateOut,3); /* strip leading 0 if month<10*/ 03520000
 Remove0(dateOut,0); /* strip leading 0 if day < 10*/ 03530000
 }
 else if(strcmp(format,"D/M/YYYY") == MATCH) 03540000
 {
 getDate(dateOut,"%d/%m/%Y"); /* format date as DD/MM/YYYY */03550000
 Remove0(dateOut,3); /* strip leading 0 if month<10*/ 03560000
 Remove0(dateOut,0); /* strip leading 0 if day < 10*/ 03570000
 }
 else if(strcmp(format,"D/M/YYY") == MATCH) 03580000
 {
 getDate(dateOut,"%d/%m/%Y"); /* format date as DD/MM/Y */ 03590000
 }
 else if(strcmp(format,"D/M/YY") == MATCH) 03600000
 {
 getDate(dateOut,"%d/%m/%Y"); /* format date as DD/MM/Y */ 03610000
 Remove0(dateOut,3); /* strip leading 0 if month<10*/ 03620000
 Remove0(dateOut,0); /* strip leading 0 if day < 10*/ 03630000
 }
 else if(strcmp(format,"DD/MM/YY") == MATCH) 03640000
 {
 getDate(dateOut,"%d/%m/%Y"); /* format date as DD/MM/YY */ 03650000
 }
 else if(strcmp(format,"DD/MM/YYYY") == MATCH) 03660000
 {
 getDate(dateOut,"%d/%m/%Y"); /* format date as DD/MM/Y */ 03670000
 }
 else if(strcmp(format,"DD/MM/YYY") == MATCH) 03680000
 {
 getDate(dateOut,"%d/%m/%Y"); /* format date as DD/MM/Y */ 03690000
 }
 else if(strcmp(format,"DD/MM/YY") == MATCH) 03700000
 {
 getDate(dateOut,"%d/%m/%Y"); /* format date as DD/MM/Y */ 03710000
 }
 else if(strcmp(format,"M/D/YY") == MATCH) 03720000
 {
 getDate(dateOut,"%m/%d/%y"); /* format date as MM/DD/YY */ 03730000
 }
 else if(strcmp(format,"M/D/YYYY") == MATCH) 03740000
 {
 getDate(dateOut,"%m/%d/%Y"); /* format date as MM/DD/YY */ 03750000
 Remove0(dateOut,3); /* strip leading 0 if day < 10*/ 03760000
 Remove0(dateOut,0); /* strip leading 0 if month<10*/ 03770000
 }
 else if(strcmp(format,"M/D/YYY") == MATCH) 03780000
 {
 getDate(dateOut,"%m/%d/%Y"); /* format date as MM/DD/Y */ 03790000
 }
 else if(strcmp(format,"M/D/YY") == MATCH) 03800000
 {
 getDate(dateOut,"%m/%d/%Y"); /* format date as MM/DD/Y */ 03810000
 Remove0(dateOut,3); /* strip leading 0 if day < 10*/ 03820000
 Remove0(dateOut,0); /* strip leading 0 if month<10*/ 03830000
 }
 else if(strcmp(format,"MM/DD/YY") == MATCH) 03840000
 {
 getDate(dateOut,"%m/%d/%Y"); /* format date as MM/DD/YY */ 03850000
 }
 else if(strcmp(format,"MM/DD/YYYY") == MATCH) 03860000
 {
 getDate(dateOut,"%m/%d/%Y"); /* format date as MM/DD/Y */ 03870000
 }
 else if(strcmp(format,"MM/DD/YYY") == MATCH) 03880000
 {
 getDate(dateOut,"%m/%d/%Y"); /* format date as MM/DD/Y */ 03890000
 }
 else if(strcmp(format,"MM/DD/YY") == MATCH) 03900000
 {
 getDate(dateOut,"%m/%d/%Y"); /* format date as MM/DD/Y */ 03910000
 }
 else if(strcmp(format,"YY/M/D") == MATCH) 03920000
 {
 getDate(dateOut,"%y/%m/%d"); /* format date as YY/MM/DD */ 03930000
 }
 else if(strcmp(format,"YY/MM/DD") == MATCH) 03940000
 {
 getDate(dateOut,"%y/%m/%d"); /* format date as YY/MM/DD */ 03950000
 Remove0(dateOut,6); /* strip leading 0 if day < 10*/ 03960000
 Remove0(dateOut,3); /* strip leading 0 if month<10*/ 03970000
 }
 else if(strcmp(format,"YY/MM/DD") == MATCH) 03980000
 {
 getDate(dateOut,"%y/%m/%d"); /* format date as YY/MM/DD */ 03990000
 }
 else if(strcmp(format,"YY/MM/DD") == MATCH) 04000000
 {
 getDate(dateOut,"%y/%m/%d"); /* format date as YY/MM/DD */ 04010000
 }
 else if(strcmp(format,"YYYY/M/D") == MATCH) 04020000
 {
 getDate(dateOut,"%Y/%m/%d"); /* format date as YYYY/MM/DD */04030000
 }
 else if(strcmp(format,"YYYY/MM/D") == MATCH) 04040000
 {
 getDate(dateOut,"%Y/%m/%d"); /* format date as YYYY/MM/D */ 04050000
 Remove0(dateOut,8); /* strip leading 0 if day < 10*/ 04060000
 Remove0(dateOut,5); /* strip leading 0 if month<10*/ 04070000
 }
 else if(strcmp(format,"YYYY/MM/DD") == MATCH) 04080000
 {
 getDate(dateOut,"%Y/%m/%d"); /* format date as YYYY/MM/DD */04090000
 }
 else if(strcmp(format,"%Y/%m/%d") == MATCH) 04100000
 {
 getDate(dateOut,"%Y/%m/%d"); /* format date as YYYY/MM/DD */04110000
 }
 else if(strcmp(format,"YY.M.D") == MATCH) 04120000
 {
 getDate(dateOut,"%y.%m.%d"); /* format date as YY.MM.DD */ 04130000
 }
 else if(strcmp(format,"%y.%m.%d") == MATCH) 04140000
 {
 getDate(dateOut,"%y.%m.%d"); /* format date as YY.MM.DD */ 04150000
 Remove0(dateOut,6); /* strip leading 0 if day < 10*/ 04160000
 Remove0(dateOut,3); /* strip leading 0 if month<10*/ 04170000
 }
}

```

```

else if(strcmp(format,"YY.MM.DD") == MATCH)
{
 getDate(dateOut,"%y.%m.%d"); /* format date as YY.MM.DD */ 04190000
 04200000
}
else if(strcmp(format,"YYYY.M.D") == MATCH)
{
 getDate(dateOut,"%Y.%m.%d"); /* format date as YYYY.MM.DD */ 04210000
 Remove0(dateOut,8); /* strip leading 0 if day < 10*/ 04220000
 Remove0(dateOut,5); /* strip leading 0 if month<10*/ 04230000
}
else if(strcmp(format,"YYYY.MM.DD") == MATCH)
{
 getDate(dateOut,"%Y.%m.%d"); /* format date as YYYY.MM.DD */ 04240000
 04250000
}
else if(strcmp(format,"YYYY-MM.DD") == MATCH)
{
 getDate(dateOut,"%Y-%m-%d"); /* format date as YYYY-MM-DD */ 04260000
 Remove0(dateOut,8); /* strip leading 0 if day < 10*/ 04270000
 Remove0(dateOut,5); /* strip leading 0 if month<10*/ 04280000
}
else if(strcmp(format,"YYYY-MM-DD") == MATCH)
{
 getDate(dateOut,"%Y-%m-%d"); /* format date as YYYY-MM-DD */ 04290000
 04300000
}
else if(strcmp(format,"YYYY-M-D") == MATCH)
{
 getDate(dateOut,"%Y-%m-%d"); /* format date as YYYY-M-D */ 04310000
 04320000
}
else if(strcmp(format,"YYYY-MM-XX") == MATCH)
{
 getDate(dateOut,"%Y-%m-%d"); /* format date as YYYY-MM-XX */ 04330000
 04340000
}
else if(strcmp(format,"YYYY-XX-D") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-"); /* start format as YYYY-XX-D */ 04350000
 strcat(dateFmt, romanMonth());/* append roman# for curr mo. */ 04360000
 getDate(dateOut,dateFmt); /* format date as YYYY-XX-D */ 04370000
 Remove0(dateOut,5); /* strip leading 0 if day < 10*/ 04380000
}
else if(strcmp(format,"YYYY-XX-XX") == MATCH)
{
 getDate(dateOut,"%Y-%m-%d"); /* format date as YYYY-XX-XX */ 04390000
 04400000
}
else if(strcmp(format,"YYYY-XX-XX-D") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-"); /* start format as YYYY-XX-D */ 04410000
 04420000
}
else if(strcmp(format,"YYYY-XX-XX-XX") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d"); /* start format as YYYY-XX-XX */ 04430000
 04440000
}
else if(strcmp(format,"YYYY-XX-XX-XX-D") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d"); /* start format as YYYY-XX-XX-D */ 04450000
 strcat(dateFmt, romanMonth());/* append roman# for curr mo. */ 04460000
 getDate(dateOut,dateFmt); /* format date as YYYY-XX-XX-D */ 04470000
 Remove0(dateOut,5); /* strip leading 0 if day < 10*/ 04480000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d-%d"); /* start format as YYYY-XX-XX-XX */ 04490000
 04500000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX-D") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d-%d-%d"); /* start format as YYYY-XX-XX-XX-D */ 04510000
 04520000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX-XX") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d-%d-%d-%d"); /* start format as YYYY-XX-XX-XX-XX */ 04530000
 04540000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX-XX-D") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d-%d-%d-%d-%d"); /* start format as YYYY-XX-XX-XX-XX-D */ 04550000
 04560000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX-XX-XX") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d-%d-%d-%d-%d-%d"); /* start format as YYYY-XX-XX-XX-XX-XX */ 04570000
 04580000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX-XX-XX-D") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d-%d-%d-%d-%d-%d-%d"); /* start format as YYYY-XX-XX-XX-XX-XX-D */ 04590000
 04600000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX-XX-XX-XX") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d"); /* start format as YYYY-XX-XX-XX-XX-XX-XX */ 04610000
 04620000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX-XX-XX-XX-D") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d"); /* start format as YYYY-XX-XX-XX-XX-XX-XX-D */ 04630000
 04640000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX-XX-XX-XX-XX") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d"); /* start format as YYYY-XX-XX-XX-XX-XX-XX-XX */ 04650000
 04660000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX-XX-XX-XX-XX-D") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d"); /* start format as YYYY-XX-XX-XX-XX-XX-XX-XX-D */ 04670000
 04680000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX-XX-XX-XX-XX-XX") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d"); /* start format as YYYY-XX-XX-XX-XX-XX-XX-XX-XX */ 04690000
 04700000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX-XX-XX-XX-XX-XX-D") == MATCH)
{
 strcpy(dateFmt, "%Y-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d-%d"); /* start format as YYYY-XX-XX-XX-XX-XX-XX-XX-XX-D */ 04710000
 04720000
}
else if(strcmp(format,"YYYY-XX-XX-XX-XX-XX-XX-XX-XX-XX-XX") == MATCH)
{
 func_status = NOT_OK;
 strcpy(message,
 "DSN8DUAD Error: Unknown format specified");
 strcpy(sqlstate, "38601");
}
else if(strcmp(format,"%B %d %Y") == MATCH)
{
 getDate(dateOut,dateFmt); /* gets curr date, formatted */ 04730000
 04740000
}
else if(strcmp(format,"%B %d %Y %I:%M:%S %p") == MATCH)
{
 getDate(dateOut,dateFmt); /* gets curr date, formatted */ 04750000
 04760000
}
else if(strcmp(format,"%B %d %Y %I:%M:%S %p %z") == MATCH)
{
 getDate(dateOut,dateFmt); /* gets curr date, formatted */ 04770000
 04780000
}
else if(strcmp(format,"%B %d %Y %I:%M:%S %p %z %f") == MATCH)
{
 getDate(dateOut,dateFmt); /* gets curr date, formatted */ 04790000
 04800000
}
else if(strcmp(format,"%B %d %Y %I:%M:%S %p %z %f %t") == MATCH)
{
 getDate(dateOut,dateFmt); /* gets curr date, formatted */ 04810000
 04820000
}

***** functions *****
void getDate /* gets curr date, formatted */ 04830000
(char *dateOut, /* out: formatted current date*/ 04840000
 char *dateFmt /* in: desired date format */ 04850000
)
***** *
* Obtains the current date from the system and formats it according * 04910000
* to the format string in *dateFmt. The result is placed in dateOut.* 04920000
* *
* This function uses the C function localtime to obtain the system * 04930000
* time and date and the C function strftime to format it according * 04940000
* to *dateFmt. For this program, the following format tokens were * 04950000
* used. See the C/C++ library reference manuals for more info. * 04960000
* *
* %B = full month name * 04970000
* %d = day of the month * 04980000
* * 04990000
* 05000000

```

```

* %m = month (01-12) * 05010000
* %y = year with century * 05020000
* %Y = year with century * 05030000
******/ 05040000
{
 time_t t; /* buff for system time macro */ 05060000
 struct tm *tmPtr; /* ->buff for time.h tm struct*/ 05070000
 short int i; /* len of str rtnd by strftime*/ 05080000
 char dateBuff[19]; /* gets formatted date @01*/ 05090000
 05100000
/***** 05110000
* Use the C function localtime to get the current date from the * 05120000
* system, then use the C function strftime to format it according * 05130000
* to *dateFmt. * 05140000
***** 05150000
t = time(NULL); 05160000
tmPtr = localtime(&t); 05170000
i = strftime(dateBuff, 05180000
 sizeof(dateBuff)-1, 05190000
 dateFmt, 05200000
 tmPtr); 05210000
 05220000
if(i > 0) 05230000
 strcpy(dateOut,dateBuff);
 05240000
 05250000
return; 05260000
} /* end getDate */
 05270000
 05280000
 05290000
void Remove0 05300000
(char *string, /* in/out: character string */ 05310000
 short int loc; /* in: loc'n of byte to remove*/ 05320000
)
***** 05340000
* Checks *string at the location indicated by loc and, if a character* 05350000
* zero resides there, removes it from *string by shifting all bytes * 05360000
* to the right of it leftward by 1 byte. * 05370000
***** 05380000
{
 short int i; 05390000
 05400000
if(string[loc] == '0')
{
 for(i=loc; i<(strlen(string)-1); i++)
 string[i] = string[i+1];
 string[i] = NULLCHAR;
}
 05410000
 05420000
 05430000
 05440000
 05450000
 05460000
 05470000
 05480000
return; 05490000
} /* end Remove0 */
 05500000
 05510000
 05520000
 05530000
char *romanMonth() 05540000
***** 05550000
* Returns the roman numeral that corresponds to the month number of * 05550000
* the current month. * 05560000
***** 05570000
{
 char romNum[5]; /* gets roman numeral */ 05590000
 char monthNo[18]; /* gets current month number */ 05600000
 05610000
/***** 05620000
* call getDate to get the month number of the current month * 05630000
***** 05640000
getDate(monthNo,"%m"); /* format date as MM */ 05650000
 05660000
/***** 05670000
* look up the roman numeral that corresponds to the current month * 05680000
***** 05690000
if(strcmp(monthNo,"01") == MATCH) strcpy(romNum, "I");
else if(strcmp(monthNo,"02") == MATCH) strcpy(romNum, "II");
else if(strcmp(monthNo,"03") == MATCH) strcpy(romNum, "III");
else if(strcmp(monthNo,"04") == MATCH) strcpy(romNum, "IV");
else if(strcmp(monthNo,"05") == MATCH) strcpy(romNum, "V");
else if(strcmp(monthNo,"06") == MATCH) strcpy(romNum, "VI");
else if(strcmp(monthNo,"07") == MATCH) strcpy(romNum, "VII");
else if(strcmp(monthNo,"08") == MATCH) strcpy(romNum, "VIII");
else if(strcmp(monthNo,"09") == MATCH) strcpy(romNum, "IX");
else if(strcmp(monthNo,"10") == MATCH) strcpy(romNum, "X");
else if(strcmp(monthNo,"11") == MATCH) strcpy(romNum, "XI");
else strcpy(romNum, "XII");
 05700000
 05710000
 05720000
 05730000
 05740000
 05750000
 05760000
 05770000
 05780000
 05790000
 05800000
 05810000
 05820000

```

```

/*****+
* return the result
*****+/ 05850000
return(romNum);
05860000
05870000
05880000

} /* end romanMonth */

```

## Referencia relacionada

["Ejemplos de aplicaciones en TSO" en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8DUAT

Devuelve la hora actual en uno de estos 8 formatos.

```

/*****+
* Module name = DSN8DUAT (DB2 sample program)
*
* DESCRIPTIVE NAME = Current time reformatter (UDF)
*
* LICENSED MATERIALS - PROPERTY OF IBM
* 5675-DB2
* (C) COPYRIGHT 1998, 2000 IBM CORP. ALL RIGHTS RESERVED.
*
* STATUS = VERSION 7
*
* Function: Returns the current time in one these 8 formats:
* formats:
*
* H:MM AM/PM HH:MM AM/PM HH:MM:SS AM/PM HH:MM:SS
* H.MM HH.MM H.MM.SS HH.MM.SS
*
* where:
*
* H: Suppress leading zero if the hour is less than 10
* HH: Retain leading zero if the hour is less than 10
* M: Suppress leading zero if the minute is less than 10
* MM: Retain leading zero if the minute is less than 10
* AM/PM: Return time in 12-hour clock format, else 24-hour
*
* Example invocation:
* EXEC SQL SET :now = ALTTIME("HH:MM:SS AM/PM");
*
* Notes:
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher
*
* Restrictions:
*
* Module type: C program
* Processor: IBM C/C++ for OS/390 V1R3 or higher
* Module size: See linkedit output
* Attributes: Re-entrant and re-usable
*
* Entry Point: DSN8DUAT
* Purpose: See Function
* Linkage: DB2SQL
* Invoked via SQL UDF call
*
* Input: Parameters explicitly passed to this function:
* - *format : pointer to a char[15], null-terminated
* string having the desired
* format for the current time (see
* "Function", above, for valid formats)
* - *niFormat : pointer to a short integer having
* the null indicator variable for
* *format.
* - *fnName : pointer to a char[138], null-terminated
* string having the UDF family
* name of this function.
* - *specificName: pointer to a char[129], null-terminated
* string having the UDF specific
* name of this function.
*
* Output: Parameters explicitly passed by this function:
* - *timeOut : pointer to a char[12], null-terminated
* string to receive the current
* time in the formatted indicated by

```

```

* *format. * 00740000
* - *niTimeOut : pointer to a short integer to re- * 00750000
* ceive the null indicator variable * 00760000
* for *timeOut. * 00770000
* - *sqlstate : pointer to a char[06], null-termi- * 00780000
* nated string to receive the SQLSTATE.* 00790000
* - *message : pointer to a char[70], null-termi- * 00800000
* nated string to receive a diagnostic * 00810000
* message if one is generated by this * 00820000
* function. * 00830000
* * 00840000
* Normal Exit: Return Code: SQLSTATE = 00000 * 00850000
* - Message: none * 00860000
* * 00870000
* Error Exit: Return Code: SQLSTATE = 38601 * 00880000
* - Message: DSN8DUAT Error: No format entered * 00890000
* - Message: DSN8DUAT Error: Unknown format specified * 00900000
* * 00910000
* External References: * 00920000
* - Routines/Services: None * 00930000
* - Data areas : None * 00940000
* - Control blocks : None * 00950000
* * 00960000
* * 00970000
* Pseudocode: * 00980000
* DSN8DUAT:
* - Verify that a valid format for the current time was received: * 01000000
* - if *format is blank or niFormat is not 0, no format passed: * 01010000
* - issue SQLSTATE 38601 and a diagnostic message * 01020000
* - Call getTime to obtain: * 01030000
* - the current 12-hour clock hour * 01040000
* - the current 24-hour clock hour * 01050000
* - the current minute * 01060000
* - the current second * 01070000
* - Set AM/PM indicator to PM if 24-hour clock hour > 11, else AM * 01080000
* - Call remove0prefix to remove leading zeroes from the hour * 01090000
* component, if appropriate * 01100000
* - Call buildTime to generate the output time using the format to * 01110000
* determine which of the time components, delimiters, and AM/PM * 01120000
* indicator (if any) to pass. * 01130000
* - If no errors, unset null indicators, and return SQLSTATE 00000 * 01140000
* else set null indicator and return null time out. * 01150000
* End DSN8DUAT * 01160000
* * 01170000
* getTime: * 01180000
* - invoke the time() library function to query calendar time. * 01190000
* - invoke the localtime() library function to convert and correct * 01200000
* for local time. * 01210000
* - invoke the strftime() library function to format 12-hour, 24- * 01220000
* hour, minute, and second components from the time vector. * 01230000
* End getTime * 01240000
* * 01250000
* buildTime: * 01260000
* - Concatenate the minutes component to the hours component with * 01270000
* an intervening delimiter (: or .). * 01280000
* - If the seconds component is not null, concatenate it to the * 01290000
* timestamping with an intervening delimiter (: or .) * 01300000
* - If the AM/PM indicator is not null, concatenate it to the * 01310000
* timestamping * 01320000
* End buildTime * 01330000
* * 01340000
* remove0prefix: * 01350000
* - eliminate leading zeroes from a hour component * 01360000
* End remove0prefix * 01370000
* * 01380000
* * 01390000
*****/ 01400000
01410000
#pragma linkage(DSN8DUAT,fetchable) 01414990
01420000
***** C library definitions *****/ 01430000
#include <stdio.h> 01440000
#include <string.h> 01450000
#include <time.h> 01460000
01470000
***** Equates *****/ 01480000
#define NULLCHAR '\0' /* Null character */ 01490000
01500000
#define MATCH 0 /* Comparison status: Equal */ 01510000
#define NOT_OK 0 /* Run status indicator: Error */ 01520000
#define OK 1 /* Run status indicator: Good */ 01530000
01540000

```

```

***** DSN8DUAT functions *****
void DSN8DUAT
(char *format, /* main routine */ 01560000
 char *timeOut, /* in: format for timeOut */ 01570000
 short int *niFormat, /* out: formatted current time*/ 01580000
 short int *niTimeOut, /* in: indic var, format */ 01590000
 char *sqlstate, /* out: indic var, timeOut */ 01600000
 char *fnName, /* in: SQLSTATE */ 01610000
 char *specificName, /* in: family name of function*/ 01620000
 char *message, /* in: specific name of func */ 01630000
); /* out: diagnostic message */ 01640000
 /* */ 01650000
 /* */ 01660000
 /* */ 01670000
 /* */ 01680000
void getTime /* Get current time */ 01690000
(char *hh12, /* out: hours (12 hour clock) */ 01700000
 char *hh24, /* out: hours (24 hour clock) */ 01710000
 char *mm, /* out: minutes */ 01720000
 char *ss, /* out: seconds */ 01730000
); /* */ 01740000
 /* */ 01750000
void buildTime /* Format time as specified */ 01760000
(char *timeStr, /* out: reformatted time */ 01770000
 char *hh, /* in: hours component */ 01780000
 char *mm, /* in: minutes component */ 01790000
 char *ss, /* in: seconds component */ 01800000
 char *delim, /* in: delimiter of choice */ 01810000
 char *suffix /* in: AM/PM suffix (if any) */ 01820000
); /* */ 01830000
 /* */ 01840000
void remove0prefix /* Remove leading zeroes */ 01850000
(char *string /* in/out: character string */ 01860000
); /* */ 01870000
***** main routine *****
void DSN8DUAT
(char *format, /* main routine */ 01880000
 char *timeOut, /* in: format for timeOut */ 01890000
 short int *niFormat, /* out: formatted current time*/ 01900000
 short int *niTimeOut, /* in: indic var, format */ 01910000
 char *sqlstate, /* out: indic var, timeOut */ 01920000
 char *fnName, /* in: SQLSTATE */ 01930000
 char *specificName, /* in: family name of function*/ 01940000
 char *message, /* in: specific name of func */ 01950000
); /* out: diagnostic message */ 01960000
 /* */ 01970000
 /* */ 01980000
 /* */ 01990000
 /* */ 02000000
 /* */ 02010000
*
* * 02020000
*
* * 02030000
* Assumptions: * 02040000
* - *format points to a char[15], null-terminated string * 02050000
* - *timeOut points to a char[12], null-terminated string * 02060000
* - *niFormat points to a short integer * 02070000
* - *niTimeOut points to a short integer * 02080000
* - *sqlstate points to a char[06], null-terminated string * 02090000
* - *fnName points to a char[138], null-terminated string * 02105990
* - *specificName points to a char[129], null-terminated string * 02111980
* - *message points to a char[70], null-terminated string * 02120000
***** local variables *****
short int status = OK; /* DSN8DUMN run status */ 02130000
 /* */ 02140000
 /* */ 02150000
 /* */ 02160000
 /* */ 02170000
char hh12[3]; /* current time - hours */ 02180000
char hh24[3]; /* current time - hours */ 02190000
char mm[3]; /* current time - minutes */ 02200000
char ss[3]; /* current time - seconds */ 02210000
char suffix[4]; /* AM/PM indicator */ 02220000
 /* */ 02230000
 /* */ 02240000
***** Verify that a format has been passed in *****
* Verify that a format has been passed in * 02250000
* * 02260000
***** if(*niFormat || (strlen(format) == 0)) *****
if(*niFormat || (strlen(format) == 0))
{
 status = NOT_OK; /* */ 02270000
 strcpy(message, /* */ 02280000
 "DSN8DUAT Error: No format entered"); /* */ 02290000
 strcpy(sqlstate, "38601"); /* */ 02300000
}
 /* */ 02310000
 /* */ 02320000
 /* */ 02330000
 /* */ 02340000
 /* */ 02350000
***** 02360000

```

```

* Get current time in the specified format * 02370000
***** / 02380000
if(status == OK)
{
 /****** * 02390000
 * Get the current hour (hh), minute (mm), and second (ss) * 02400000
 ***** / 02410000
 * Suffix is AM unless it's noon or later on the 24-hour clock * 02420000
 ***** / 02430000
 getTime(hh12, hh24, mm, ss); 02440000
 02450000
 /****** * 02460000
 * Format the current time according to the input format * 02470000
 ***** / 02480000
 strcpy(suffix, " AM");
 if(strcmp(hh24, "11") > 0) 02490000
 strcpy(suffix, " PM");
 02500000
 /****** * 02510000
 * Build the time string according to the specified format * 02520000
 ***** / 02530000
 if(strcmp(format, "H:MM AM/PM") == MATCH) 02540000
 {
 remove0prefix(hh12);
 buildTime(timeOut, hh12, mm, "", ":" , suffix);
 }
 else if(strcmp(format, "HH:MM AM/PM") == MATCH) 02550000
 {
 buildTime(timeOut, hh12, mm, "", ":" , suffix);
 }
 else if(strcmp(format, "HH:MM:SS AM/PM") == MATCH) 02560000
 {
 buildTime(timeOut, hh12, mm, ss, ":" , suffix);
 }
 else if(strcmp(format, "HH:MM:SS") == MATCH) 02570000
 {
 buildTime(timeOut, hh24, mm, ss, ":" , "");
 }
 else if(strcmp(format, "H.MM") == MATCH) 02580000
 {
 remove0prefix(hh24);
 buildTime(timeOut, hh24, mm, "", ".", "");
 }
 else if(strcmp(format, "HH.MM") == MATCH) 02590000
 {
 buildTime(timeOut, hh24, mm, "", ".", "");
 }
 else if(strcmp(format, "H.MM.SS") == MATCH) 02600000
 {
 remove0prefix(hh24);
 buildTime(timeOut, hh24, mm, ss, ".", "");
 }
 else if(strcmp(format, "HH.MM.SS") == MATCH) 02610000
 {
 buildTime(timeOut, hh24, mm, ss, ".", "");
 }
 else if(strcmp(format, "H.MM") == MATCH) 02620000
 {
 buildTime(timeOut, hh24, mm, "", ".", "");
 }
 else if(strcmp(format, "H") == MATCH) 02630000
 {
 buildTime(timeOut, hh24, mm, "", "", "");
 }
 else if(strcmp(format, "MM") == MATCH) 02640000
 {
 buildTime(timeOut, hh24, mm, "", "", ":");
 }
 else if(strcmp(format, "SS") == MATCH) 02650000
 {
 buildTime(timeOut, hh24, mm, "", "", ":");
 }
 else if(strcmp(format, "AM/PM") == MATCH) 02660000
 {
 buildTime(timeOut, hh24, mm, "", "", ":");
 }
 else if(strcmp(format, "MM:SS") == MATCH) 02670000
 {
 buildTime(timeOut, hh24, mm, "", "", ":");
 }
 else if(strcmp(format, "MM") == MATCH) 02680000
 {
 buildTime(timeOut, hh24, mm, "", "", ":");
 }
 else if(strcmp(format, "SS") == MATCH) 02690000
 {
 buildTime(timeOut, hh24, mm, "", "", ":");
 }
 else if(strcmp(format, "H") == MATCH) 02700000
 {
 buildTime(timeOut, hh24, mm, "", "", ":");
 }
 else if(strcmp(format, "MM") == MATCH) 02710000
 {
 buildTime(timeOut, hh24, mm, "", "", ":");
 }
 else if(strcmp(format, "SS") == MATCH) 02720000
 {
 buildTime(timeOut, hh24, mm, "", "", ":");
 }
 else if(strcmp(format, "AM") == MATCH) 02730000
 {
 buildTime(timeOut, hh24, mm, "", "", ":");
 }
 else if(strcmp(format, "PM") == MATCH) 02740000
 {
 buildTime(timeOut, hh24, mm, "", "", ":");
 }
 else
 {
 status = NOT_OK;
 strcpy(message,
 "DSN8DUAT Error: Unknown format specified");
 strcpy(sqlstate, "38601");
 }
} /* if(status == OK) */

/****** * 03000000
 * If operation was successful, clear the message buffer and sql- * 03010000
 * state, and unset the SQL null indicator for timeOut. * 03020000
***** / 03030000
if(status == OK)
{
 /*niTimeOut = 0; 03040000
 message[0] = NULLCHAR; 03050000
 strcpy(sqlstate, "00000"); 03060000
}
/****** * 03100000
 * If errors occurred, clear the timeOut buffer and set the SQL null* 03110000
 * indicator. A diagnostic message and the SQLSTATE have been set * 03120000
 * where the error was detected. * 03130000
***** / 03140000
else
{
 timeOut[0] = NULLCHAR; 03150000
 *niTimeOut = -1; 03160000
}

```

```

 }

 return;
} /* end DSN8DUAT */

/*** functions *****/
void getTime
(char *hh12, /* out: hours (12 hour clock) */ 03300000
 char *hh24, /* out: hours (24 hour clock) */ 03310000
 char *mm, /* out: minutes */ 03320000
 char *ss /* out: seconds */ 03330000
)
/*** */
/* Obtains the current time from the system and returns the hours in * 03360000
* *hh12 (12-hour clock hours) and *hh24 (24-hour clock hours), the * 03370000
* minutes in *mm, and the seconds in *ss. * 03380000
***** */ 03390000
{
 time_t t; /* buff for system time macro */ 03410000
 struct tm *tmptr; /* buffer for time.h tm struct*/ 03420000
 char timeBuf[13]; /* buffer to receive sys time */ 03430000
 03440000
 short int i; /* len of str rtnd by strftime*/ 03450000
 char *tokPtr; /* string ptr for token parser*/ 03460000
 char hhmmss[10]; /* time in HH:MM:SS format */ 03470000
 03480000
/*** */
/* Use the C function localtime to get the current time from the * 03500000
* system, then use the C function strftime to format it into a * 03510000
* string containing both 12- and 24-hour clock hours and minutes * 03520000
* and seconds, all separated by slashes. * 03530000
***** */ 03540000
 t = time(NULL); 03550000
 tmptr = localtime(&t); 03560000
 i = strftime(timeBuf,
 sizeof(timeBuf)-1,
 "%I/%H/%M/%S",
 tmptr);
 03580000
/*** */
/* Use the strtok func to extract time components from time buffer */ 03630000
***** */
 tokPtr = strtok(timeBuf,"/"); /* Parse to first slash */ 03650000
 strcpy(hh12,tokPtr); /* for hours on 12-hour clock */ 03660000
 03670000
 tokPtr = strtok(NULL,"/"); /* Parse to 2nd slash */ 03680000
 strcpy(hh24,tokPtr); /* for hours on 24-hour clock */ 03690000
 03700000
 tokPtr = strtok(NULL,"/"); /* Parse to 3rd slash */ 03710000
 strcpy(mm,tokPtr); /* for minutes */ 03720000
 03730000
 tokPtr = strtok(NULL,"/"); /* Parse remaining bytes */ 03740000
 strcpy(ss,tokPtr); /* for seconds */ 03750000
 03760000
}
/* end getTime */

void buildTime
(char *timeStr, /* out: reformatted time */ 03810000
 char *hh, /* in: hours component */ 03820000
 char *mm, /* in: minutes component */ 03830000
 char *ss, /* in: seconds component */ 03840000
 char *delim, /* in: delimiter of choice */ 03850000
 char *suffix /* in: AM/PM suffix (if any) */ 03860000
)
/*** */
/* Builds *timeStr from hours (*hh), minutes (*mm), and seconds (*ss, * 03890000
* if not null), separated by the value in *delim and suffixed by the * 03900000
* value, if not null, in *suffix. * 03910000
***** */ 03920000
{
 /* Build timeStr from incoming time components */ 03940000
 strcpy(timeStr, hh); /* Start with hours ... */ 03970000
 strcat(timeStr,delim);
 strcat(timeStr,mm);
 if(strlen(ss) > 0) /* and, if seconds specified, */ 04000000

```

```

{
 strcat(timeStr,delim); /* ..append the delimiter */ 04010000
 strcat(timeStr,ss); /* ..append seconds */ 04020000
}
if(strlen(suffix) > 0) /* and, if suffix specified, */ 04030000
 strcat(timeStr,suffix); /* ..append it. */ 04040000
04050000
04060000
04070000
} /* end buildTime */

04080000
04090000
04100000
04110000
void remove0prefix
(char *string /* in/out: character string */ 04120000
)
***** 04130000
* Eliminates all leading zeroes from *string. Leaves a single zero * 04140000
* in the first byte of *string if *string is all zeroes. * 04150000
***** 04160000
***** 04170000
{
 short int i = 0; /* Loop control */ 04180000
 short int j = 0; /* Loop control */ 04190000
04200000
04210000
***** 04220000
* if leading zero in first byte, skip up to first non-zero * 04230000
***** 04240000
if(string[0] == '0') 04250000
 for(i=0; string[i] == '0'; i++);
04260000
04270000
***** 04280000
* if at end of string, it was all zeroes: put zero in 1st byte * 04290000
***** 04300000
if(string[i] == '\0')
 strcpy(string,"0");
04310000
04320000
***** 04330000
* otherwise, left-shift non-zero chars and terminate string * 04340000
***** 04350000
else
{
 for(j=0; string[i] != NULLCHAR; j++)
 string[j] = string[i++];
 string[j] = NULLCHAR;
}
} /* end remove0prefix */
04360000
04370000
04380000
04390000
04400000
04410000
04420000

```

## Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8DUCD

Convierte una fecha determinada de uno a otro de estos 34 formatos.

```

***** 00010000
* Module name = DSN8DUCD (DB2 sample program) * 00020000
*
* DESCRIPTIVE NAME = General date reformatter (UDF) * 00030000
*
* * 00040000
* * 00050000
* * 00120000
* LICENSED MATERIALS - PROPERTY OF IBM * 00130000
* 5675-DB2 * 00140000
* (C) COPYRIGHT 2000 IBM CORP. ALL RIGHTS RESERVED. * 00150000
*
* STATUS = VERSION 7 * 00160000
*
* * 00170000
* * 00190000
* * 00210000
*
* Function: Converts a given date from one to another of these 34 * 00220000
* formats: * 00230000
* * 00240000
*
* D MONTH YY D MONTH YYYY DD MONTH YY DD MONTH YYYY * 00250000
* D.M.YY D.M.YYYY DD.MM.YY DD.MM.YYYY * 00260000
* D-M-YY D-M-YYYY DD-MM-YY DD-MM-YYYY * 00270000
* D/M/YY D/M/YYYY DD/MM/YY DD/MM/YYYY * 00280000
* M/D/YY M/D/YYYY MM/DD/YY MM/DD/YYYY * 00290000
* YY/M/D YYYY/M/D YY/MM/DD YYYY/MM/DD * 00300000
* YY.M.D YYYY.M.D YY.MM.DD YYYY.MM.DD * 00310000
* YYYY-Y-M-D YYYY-MM-DD * 00320000
* YYYY-Y-D-XX YYYY-DD-XX * 00330000
* YYYY-XX-D YYYY-XX-DD * 00340000
* * 00350000

```

```

* where:
* D: Suppress leading zero if the day is less than 10
* DD: Retain leading zero if the day is less than 10
* M: Suppress leading zero if the month is less than 10
* MM: Retain leading zero if the month is less than 10
* MONTH: Use English-language name of month
* XX: Use a capital Roman numeral for month
* XX: Use a capital Roman numeral for month
* YY: Use non-century year format
* YYYY: Use century year format

* Example invocation:
* EXEC SQL SET :newDate = ALTDATE("3/15/1947",
* "M/D/YYYY",
* "DD MONTH YY");
* ==> newDate = "15 March 47"

* Notes:
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher
*
* Restrictions:
*
* Module type: C program
* Processor: IBM C/C++ for OS/390 V1R3 or higher
* Module size: See linkedit output
* Attributes: Re-entrant and re-usable
*
* Entry Point: DSN8DUCD
* Purpose: See Function
* Linkage: DB2SQL
* Invoked via SQL UDF call

* Input: Parameters explicitly passed to this function:
* - *dateIn : pointer to a char[18], null-terminated string having a date in the format indicated by *formatIn.
* - *formatIn : pointer to a char[14], null-terminated string having the format of date found in *dateIn (see "Function", above, for valid formats).
* - *formatOut : pointer to a char[14], null-terminated string having the format to which the date found in *dateIn is to be converted. See "Function", above, for valid formats.
* - *niDateIn : pointer to a short integer having the null indicator variable for *dateIn.
* - *niFormatIn : pointer to a short integer having the null indicator variable for *formatIn.
* - *niFormatOut : pointer to a short integer having the null indicator variable for *formatOut.
* - *fnName : pointer to a char[138], null-terminated string having the UDF family name of this function.
* - *specificName: pointer to a char[129], null-terminated string having the UDF specific name of this function.

* Output: Parameters explicitly passed by this function:
* - *dateOut : pointer to a char[18], null-terminated string to receive the reformatted date.
* - *niDateOut : pointer to a short integer to receive the null indicator variable for *dateOut.
* - *sqlstate : pointer to a char[06], null-terminated string to receive the SQLSTATE.
* - *message : pointer to a char[70], null-terminated string to receive a diagnostic message if one is generated by this function.

* Normal Exit: Return Code: SQLSTATE = 00000
* - Message: none

* Error Exit: Return Code: SQLSTATE = 38601
* - Message: DSN8DUCD Error: No input date entered
* - Message: DSN8DUCD Error: No input format entered

```

```

* - Message: DSN8DUCD Error: No output format entered * 01180000
* *
* Return Code: SQLSTATE = 38602 * 01190000
* - Message: DSN8DUCD Error: Unknown input format * 01200000
* specified
* - Message: DSN8DUCD Error: Value for year is incor- * 01210000
* rect or does not conform to input format * 01220000
* - Message: DSN8DUCD Error: Value for month is incor- * 01230000
* rect or does not conform to input format * 01240000
* - Message: DSN8DUCD Error: Value for day is incor- * 01250000
* rect or does not conform to input format * 01260000
* - Message: DSN8DUCD Error: Value for hour is incor- * 01270000
* rect or does not conform to input format * 01280000
* - Message: DSN8DUCD Error: Value for minute is incor- * 01290000
* rect or does not conform to input format * 01300000
* to input format
* *
* Return Code: SQLSTATE = 38602 * 01310000
* - Message: DSN8DUCD Error: Unknown output format * 01320000
* specified
* *
* External References:
* - Routines/Services: None * 01330000
* - Data areas : None * 01340000
* - Control blocks : None * 01350000
* *
* *
* Pseudocode:
* DSN8DUCD:
* - Issue sqlstate 38601 and a diagnostic message if no input date * 01370000
* was provided. * 01380000
* - Issue sqlstate 38601 and a diagnostic message if no input for- * 01390000
* mat was provided. * 01400000
* - Issue sqlstate 38601 and a diagnostic message if no output * 01410000
* format was provided. * 01420000
* - Call deconDate to deconstruct the input date into year, month, * 01430000
* and day components according to the input format. * 01440000
* - Call reconDate to create an output date from the year, month, * 01450000
* and day components according to the output format. * 01460000
* - If no errors, unset null indicators, and return SQLSTATE 00000 * 01470000
* else set null indicator and return null date out. * 01480000
* End DSN8DUCD
* *
* deconDate
* - Parse day, month, and year (sequence unknown) components from * 01490000
* the input date by breaking on delimiters (blank, /, ., and -). * 01500000
* - Use the input format to determine sequence of date components. * 01510000
* - if format invalid, issue SQLSTATE 38602 and a diag. message * 01520000
* - Call checkDay to validate the day component * 01530000
* - if not valid day, issue SQLSTATE 38602 and a diag. message * 01540000
* - Call checkMonth to validate the month component and convert it * 01550000
* (if required) from a calendar month name or roman numeral to * 01560000
* a month number (1-12). * 01570000
* - if not valid month, issue SQLSTATE 38602 and a diag. message * 01580000
* - Call checkYear to validate the year component. * 01590000
* - if not valid year, issue SQLSTATE 38602 and a diag. message * 01600000
* End deconDate
* *
* reconDate
* - Use the output format to edit and sequence the date components * 01610000
* - call add0prefix to prepend leading 0's to the day and/or * 01620000
* month component(s), as appropriate * 01630000
* - or call remove0prefix to drop leading 0's from the day and/ * 01640000
* or month component(s), as appropriate * 01650000
* - call nameMonth to convert the month number (1-12) to calen- * 01660000
* dar name, if appropriate * 01670000
* - call romanMonth to convert the month number (1-12) to roman * 01680000
* numeral, if appropriate * 01690000
* - call addCentury to convert a non-century year to century * 01700000
* date, if appropriate * 01710000
* - call removeCentury to convert a century year to non-century * 01720000
* if appropriate * 01730000
* - convert the month to a calendar name or roman numeral, if * 01740000
* appropriate * 01750000
* - convert the year to a non-century format, if appropriate * 01760000
* - if output format is invalid, issue SQLSTATE 38603 and a * 01770000
* diagnostic message
* - Call buildDate to create the output date from the edited, re- * 01780000
* sequenced date components
* End reconDate
* *
* buildDate
* - Generate the date out by concatenating the date components * 01790000
* (month, day, and year) with intervening delimiters (blank, ., * 01800000
* *
* *

```

```

* /, or -) in the sequence directed by reconDate * 02010000
* End buildDate * 02020000
*
* nameMonth
* - convert a month in the standard form, 1-12, to the correspond- * 02030000
* ing calendar month name. * 02040000
* End nameMonth * 02050000
*
* unameMonth
* - convert a calendar month name to the corresponding month no. * 02060000
* in the standard form, 1-12. * 02070000
* End unameMonth * 02080000
*
* romanMonth
* - convert a month in the standard form, 1-12, to the correspond- * 02090000
* ing roman numeral, I-XII. * 02100000
* End romanMonth * 02110000
*
* unromanMonth
* - convert a roman numeral (I-XII) to the corresponding month no. * 02120000
* in the standard form, 1-12. * 02130000
* End unromanMonth * 02140000
*
* checkYear
* - Verify that the year component of the input date is one of the * 02150000
* following, in accordance with the input format: * 02160000
* - A valid century year (0000-9999) * 02170000
* - A valid non-century year (00-99) * 02180000
* - If not valid, set error flag and return null value for year * 02190000
* End checkYear * 02200000
*
* checkMonth
* - Verify the month component of the input date in accordance * 02210000
* with the input format: * 02220000
* - if the month is a calendar name, call unameMonth to convert * 02230000
* it to a month number (1-12). * 02240000
* - if the month is a roman numeral, call unromanMonth to con- * 02250000
* vert it to a month number (1-12). * 02260000
* - If not valid, set error flag and return null value for month * 02270000
* End checkMonth * 02280000
*
* checkDay
* - Verify that the day component of the input date is one or two * 02290000
* numeric characters * 02300000
* - If not valid, set error flag and return null value for day * 02310000
* End checkDay * 02320000
*
* add0prefix
* - prepend a day or month with a leading 0 if it is less than 10 * 02330000
* End add0prefix * 02340000
*
* remove0prefix
* - strip leading zero from a day or month if it is less than 10 * 02350000
* End remove0prefix * 02360000
*
* addCentury
* - If the year component is non-century format, prepend it with * 02370000
* the current century. * 02380000
* End addCentury * 02390000
*
* removeCentury
* - If the year component is century format, strip off the century * 02400000
* portion. * 02410000
* End removeCentury * 02420000
*
* ****
* #pragma linkage(DSN8DUCD,fetchable) 02430000
* **** C library definitions ****/ 02440000
* #include <stdio.h> 02450000
* #include <string.h> 02460000
* #include <ctype.h> 02470000
* #include <time.h> 02480000
*
* **** Equates ****/ 02490000
* #define NULLCHAR '\0' /* Null character */ 02500000
* */ 02510000
* ****
* #define MATCH 0 /* Comparison status: Equal */ 02520000
* #define NOT_OK 0 /* Run status indicator: Error*/ 02530000
* #define OK 1 /* Run status indicator: Good */ 02540000
* */ 02550000
* */ 02560000
* */ 02570000
* */ 02580000
* */ 02590000
* */ 02600000
* */ 02610000
* */ 02620000
* */ 02630000
* */ 02640000
* */ 02650000
* */ 02660000
* */ 02670000
* */ 02680000
* */ 02682990
* */ 02685980
* */ 02690000
* */ 02700000
* */ 02710000
* */ 02720000
* */ 02730000
* */ 02740000
* */ 02750000
* */ 02760000
* */ 02770000
* */ 02780000
* */ 02790000
* */ 02800000

```

```

02810000
02820000
02830000
02840000
02850000
02860000
02870000
02880000
02890000
02900000
02910000
02920000
02930000
02940000
02950000
02960000
02970000
02980000
02990000
03000000
03010000
03020000
03030000
03040000
03050000
03060000
03070000
03080000
03090000
03100000
03110000
03120000
03130000
03140000
03150000
03160000
03170000
03180000
03190000
03200000
03210000
03220000
03230000
03240000
03250000
03260000
03270000
03280000
03290000
03300000
03310000
03320000
03330000
03340000
03350000
03360000
03370000
03380000
03390000
03400000
03410000
03420000
03430000
03440000
03450000
03460000
03470000
03480000
03490000
03500000
03510000
03520000
03530000
03540000
03550000
03560000
03570000
03580000
03590000
03600000
03610000
03620000

/***** Global constants *****/
char *char0 = "0";
char *delimiters /* Valid format delimiters */;
char *monthNames[12] /* Month names */;
char *monthNums[12] /* Month numbers (as strings) */;
char *romanNums[12] /* Roman numerals */;

/***** DSN8DUCD functions *****/
void DSN8DUCD /* main routine */;
(char *dateIn, /* in: date to be converted */;
 char *formatIn, /* in: format of dateIn */;
 char *formatOut, /* in: format for dateOut */;
 char *dateOut, /* out: reformatted date */;
 short int *nullDateIn, /* in: indic var for dateIn */;
 short int *nullFormatIn, /* in: indic var for formatIn */;
 short int *nullFormatOut, /* in: indic var, formatOut */;
 short int *nullDateOut, /* out: indic var for dateOut */;
 char *sqlstate, /* out: SQLSTATE */;
 char *fnName, /* in: family name of function */;
 char *specificName, /* in: specific name of func */;
 char *message /* out: diagnostic message */);

int deconDate /* get yr, mo, dy from dateIn */;
(char *yr, /* out: year component */;
 char *mo, /* out: month component */;
 char *dy, /* out: day component */;
 char *message, /* out: diagnostic message */;
 char *sqlstate, /* out: SQLSTATE */;
 char *dateIn, /* in: inputted date string */;
 char *fmtIn /* in: format of dateIn */);

int reconDate /* get dateOut from yr,mo,dy */;
(char *dateOut, /* out: reformatted date str */;
 char *message, /* out: diagnostic message */;
 char *sqlstate, /* out: SQLSTATE */;
 char *yr, /* in: year component */;
 char *mo, /* in: month component */;
 char *dy, /* in: day component */;
 char *fmtOut /* in: format for dateOut */);

void buildDate /* build date from parts */;
(char *dtOut, /* out: date */;
 char *d1, /* in: year, month, or day */;
 char *d2, /* in: year, month, or day */;
 char *d3, /* in: year, month, or day */;
 char *delim /* in: delimiter */);

void add0prefix /* add leading zero to string */;
(char *str3 /* in/out: string to prefix */);

void remove0prefix /* strips leading zeroes */;
(char *string /* in/out: string to strip */);

int nameMonth /* converts month num to name */;
(char *monthIn /* in/out: month to convert */);

```

```

int unameMonth
(char *monthIn
);

int romanMonth
(char *monthIn
);

int unromanMonth
(char *monthIn
);

int checkYear
(char *yearOut,
 char *yearIn,
 char *style
);

int checkMonth
(char *monthOut,
 char *monthIn,
 char *style
);

int checkDay
(char *dayOut,
 char *dayIn
);

void addCentury
(char *yearIn
);

void removeCentury
(char *yearIn
);

/***
***** main routine *****
*/
void DSN8DUCD
(char *dateIn,
 char *formatIn,
 char *formatOut,
 char *dateOut,
 short int *nullDateIn,
 short int *nullFormatIn,
 short int *nullFormatOut,
 short int *nullDateOut,
 char *sqlstate,
 char *fnName,
 char *specificName,
 char *message
)
{
/*
* Assumptions:
* - *dateIn points to a char[18], null-terminated string
* - *formatIn, points to a char[14], null-terminated string
* - *formatOut, points to a char[14], null-terminated string
* - *dateOut, points to a char[18], null-terminated string
* - *nullDateIn, points to a short integer
* - *nullFormatIn points to a short integer
* - *nullFormatOut points to a short integer
* - *nullDateOut, points to a short integer
* - *sqlstate points to a char[06], null-terminated string
* - *fnName points to a char[138], null-terminated string
* - *specificName points to a char[129], null-terminated string
* - *message points to a char[70], null-terminated string

* Local variables *****
short int i; /* loop control vars */
char year[5]; /* gets year from dateIn */
char month[10]; /* gets month from dateIn */
char day[3]; /* gets day from dateIn */
short int status = OK; /* DSN8DUCD run status */
*/
/* Verify that an input date, its current format, and its new format */

```

```

* have been passed in. * 04450000
***** *****/ 04460000
if(*nullDateIn || (strlen(dateIn) == 0))
{
 {
 status = NOT_OK;
 strcpy(message,
 "DSN8DUCD Error: No input date entered");
 strcpy(sqlstate, "38601");
 }
 else if(*nullFormatIn || (strlen(formatIn) == 0))
 {
 status = NOT_OK;
 strcpy(message,
 "DSN8DUCD Error: No input format entered");
 strcpy(sqlstate, "38601");
 }
 else if(*nullFormatOut || (strlen(formatOut) == 0))
 {
 status = NOT_OK;
 strcpy(message,
 "DSN8DUCD Error: No output format entered");
 strcpy(sqlstate, "38601");
 }
}

/***** *****/ 04690000
* Use formatIn to deconstruct date in into year, month, and day * 04700000
***** *****/ 04710000
if(status == OK)
 status = deconDate(year, month, day, message, sqlstate,
 dateIn, formatIn);
 04720000
 04730000
 04740000
 04750000
/***** *****/ 04760000
* Use formatOut to reconstruct date from year, month, and day * 04770000
***** *****/ 04780000
if(status == OK)
 status = reconDate(dateOut, message, sqlstate,
 year, month, day, formatOut);
 04790000
 04800000
 04810000
 04820000
/***** *****/ 04830000
* If conversion was successful, clear the message buffer and sql- * 04840000
* state, and unset the SQL null indicator for dateOut. * 04850000
***** *****/ 04860000
if(status == OK)
{
 {
 *nullDateOut = 0;
 message[0] = NULLCHAR;
 strcpy(sqlstate,"00000");
 }
}

/***** *****/ 04930000
* If errors occurred, clear the dateOut buffer and set the SQL null* 04940000
* indicator. A diagnostic message and the SQLSTATE have been set * 04950000
* where the error was detected. * 04960000
***** *****/ 04970000
else
{
 {
 dateOut[0] = NULLCHAR;
 *nullDateOut = -1;
 }
}

return;
} /* end of DSN8DUCD */

/***** *****/ 05080000
/***** Functions *****/ 05090000
/***** *****/ 05100000
int deconDate /* get yr, mo, dy from dateIn */ 05110000
(char *yr, /* out: year component */ 05120000
 char *mo, /* out: month component */ 05130000
 char *dy, /* out: day component */ 05140000
 char *message, /* out: diagnostic message */ 05150000
 char *sqlstate, /* out: SQLSTATE */ 05160000
 char *dateIn, /* in: inputted date string */ 05170000
 char *fmtIn /* in: format of dateIn */ 05180000
)
/***** *****/ 05190000
* Deconstructs *dateIn into *yr, *mo, and *dy according to *fmtIn. * 05210000
* Returns 1 if deconstruction succeeds, otherwise places diagnostic * 05220000
* text in *message and returns 0. * 05230000
***** *****/ 05240000
{
 /* Local variables */

```

```

short int func_status = OK; /* function status indicator */ 05270000
short int yrStatus = OK; /* indicates if year is OK */ 05280000
short int moStatus = OK; /* " " month " " */ 05290000
short int dyStatus = OK; /* " " day " " */ 05300000
short int ftStatus = OK; /* " " format " " */ 05310000
 05320000
 05330000
char workDateIn[18]; /* work copy of dateIn */ 05340000
char *token; /* Value from token parser */ 05350000
 05360000
char tok1[17]; /* Gets 1st date component */ 05370000
char tok2[17]; /* " 2nd " */ 05380000
char tok3[17]; /* " 3rd " */ 05390000
 05400000
/*** 05410000
* Parse day, month, and year (order unknown) from dateIn * 05420000
***** 05430000
strcpy(workDateIn,dateIn); 05440000
token = strtok(workDateIn," .-/"); 05450000
strcpy(tok1,token); 05460000
token = strtok(NULL," .-/"); 05470000
strcpy(tok2,token); 05480000
token = strtok(NULL," .-/"); 05490000
strcpy(tok3,token); 05500000
 05510000
/*** 05520000
* Use fmtIn to check and set year, month, and day from date tokens * 05530000
***** 05540000
if((strcmp(fmtIn,"D MONTH YY") == MATCH) 05550000
|| (strcmp(fmtIn,"DD MONTH YY") == MATCH) 05560000
{
 dyStatus = checkDay(dy,tok1); 05570000
 moStatus = checkMonth(mo,tok2,"MONTH"); 05580000
 yrStatus = checkYear(yr,tok3,"YY"); 05590000
}
else if((strcmp(fmtIn,"D MONTH YYYY") == MATCH) 05600000
|| (strcmp(fmtIn,"DD MONTH YYYY") == MATCH)) 05610000
{
 dyStatus = checkDay(dy,tok1); 05620000
 moStatus = checkMonth(mo,tok2,"MONTH"); 05630000
 yrStatus = checkYear(yr,tok3,"YYYY"); 05640000
}
else if((strcmp(fmtIn,"D.M.YY") == MATCH) 05650000
|| (strcmp(fmtIn,"DD.MM.YY") == MATCH) 05660000
|| (strcmp(fmtIn,"D-M-YY") == MATCH) 05670000
|| (strcmp(fmtIn,"DD-MM-YY") == MATCH) 05680000
|| (strcmp(fmtIn,"D/M/YY") == MATCH) 05690000
|| (strcmp(fmtIn,"DD/MM/YY") == MATCH)) 05700000
{
 dyStatus = checkDay(dy,tok1); 05710000
 moStatus = checkMonth(mo,tok2,"M/MM"); 05720000
 yrStatus = checkYear(yr,tok3,"YY"); 05730000
}
else if((strcmp(fmtIn,"D.M.YYYY") == MATCH) 05740000
|| (strcmp(fmtIn,"DD.MM.YYYY") == MATCH) 05750000
|| (strcmp(fmtIn,"D-M-YYYY") == MATCH) 05760000
|| (strcmp(fmtIn,"DD-MM-YYYY") == MATCH) 05770000
|| (strcmp(fmtIn,"D/M/YYYY") == MATCH) 05780000
|| (strcmp(fmtIn,"DD/MM/YYYY") == MATCH)) 05790000
{
 dyStatus = checkDay(dy,tok1); 05800000
 moStatus = checkMonth(mo,tok2,"M/MM"); 05810000
 yrStatus = checkYear(yr,tok3,"YYYY"); 05820000
}
else if((strcmp(fmtIn,"M/D/YY") == MATCH) 05830000
|| (strcmp(fmtIn,"MM/DD/YY") == MATCH)) 05840000
{
 moStatus = checkMonth(mo,tok1,"M/MM"); 05850000
 dyStatus = checkDay(dy,tok2); 05860000
 yrStatus = checkYear(yr,tok3,"YY"); 05870000
}
else if((strcmp(fmtIn,"M/D/YYYY") == MATCH) 05880000
|| (strcmp(fmtIn,"MM/DD/YYYY") == MATCH)) 05890000
{
 moStatus = checkMonth(mo,tok1,"M/MM"); 05900000
 dyStatus = checkDay(dy,tok2); 05910000
 yrStatus = checkYear(yr,tok3,"YYYY"); 05920000
}
else if((strcmp(fmtIn,"YY/M/D") == MATCH) 05930000
|| (strcmp(fmtIn,"YY/MM/DD") == MATCH) 05940000
|| (strcmp(fmtIn,"YY.M.D") == MATCH) 05950000
|| (strcmp(fmtIn,"YY.MM.DD") == MATCH)) 05960000
{
 moStatus = checkMonth(mo,tok1,"M/MM"); 05970000
 dyStatus = checkDay(dy,tok2); 05980000
 yrStatus = checkYear(yr,tok3,"YYYY"); 05990000
}
else if((strcmp(fmtIn,"YY/MM/DD") == MATCH) 06000000
|| (strcmp(fmtIn,"YY.D") == MATCH) 06010000
|| (strcmp(fmtIn,"YY.MM.D") == MATCH)) 06020000
{
 moStatus = checkMonth(mo,tok1,"M/MM"); 06030000
 dyStatus = checkDay(dy,tok2); 06040000
 yrStatus = checkYear(yr,tok3,"YYYY"); 06050000
}

```

```

 {
 yrStatus = checkYear(yr,tok1,"YY");
 moStatus = checkMonth(mo,tok2,"M/MM");
 dyStatus = checkDay(dy,tok3);
 }
 else if((strcmp(fmtIn,"YYYY/M/D") == MATCH)
 || (strcmp(fmtIn,"YYYY/MM/DD") == MATCH)
 || (strcmp(fmtIn,"YYYY.M.D") == MATCH)
 || (strcmp(fmtIn,"YYYY.MM.DD") == MATCH)
 || (strcmp(fmtIn,"YYYY-M-D") == MATCH)
 || (strcmp(fmtIn,"YYYY-MM-DD") == MATCH))
 {
 yrStatus = checkYear(yr,tok1,"YYYY");
 moStatus = checkMonth(mo,tok2,"M/MM");
 dyStatus = checkDay(dy,tok3);
 }
 else if((strcmp(fmtIn,"YYYY-D-XX") == MATCH)
 || (strcmp(fmtIn,"YYYY-DD-XX") == MATCH))
 {
 yrStatus = checkYear(yr,tok1,"YYYY");
 dyStatus = checkDay(dy,tok2);
 moStatus = checkMonth(mo,tok3,"XX");
 }
 else if((strcmp(fmtIn,"YYYY-XX-D") == MATCH)
 || (strcmp(fmtIn,"YYYY-XX-DD") == MATCH))
 {
 yrStatus = checkYear(yr,tok1,"YYYY");
 moStatus = checkMonth(mo,tok2,"XX");
 dyStatus = checkDay(dy,tok3);
 }
 else /* date-in format is invalid or unknown */
 ftStatus = NOT_OK;
}

/********************* * 06420000
* set up error handling * 06430000
***** 06440000
func_status = NOT_OK;
strcpy(message,"DSN8DUCD Error: ");
strcpy(sqlstate, "38602");

/********************* 06490000
* if error detected, issue diagnostic message and return NOT_OK * 06500000
***** 06510000
if(ftStatus != OK)
 strcpy(message,
 "Unknown input format specified");
else if(yrStatus != OK)
 strcpy(message,
 "Value for year "
 "is incorrect or does not "
 "conform to input format");
else if(moStatus != OK)
 strcpy(message,
 "Value for month "
 "is incorrect or does not "
 "conform to input format");
else if(dyStatus != OK)
 strcpy(message,
 "Value for day "
 "is incorrect or does not "
 "conform to input format");

/********************* 06710000
* if no error detected, clear message and sqlstate and return OK * 06720000
***** 06730000
else
{
 *message = NULLCHAR;
 func_status = OK;
 strcpy(sqlstate, "00000");
}

return(func_status);
} /* end deconDate */

int reconDate /* get dateOut from yr,mo,dy */ 06840000
(char *dateOut, /* out: reformatted date str */ 06850000
 char *message, /* out: diagnostic message */ 06860000
 char *sqlstate, /* out: SQLSTATE */ 06870000
 char *yr, /* in: year component */ 06880000
 char *mo, /* in: month component */ 06890000
 char *dy, /* in: day component */ 06900000

```

```

 char *fmtOut /* in: format for dateOut */ 06910000
)
/*** 06920000
* Reconstructs *yr, *mo, and *dy into *dateOut according to *fmtOut. * 06930000
* Returns 1 if reconstruction succeeds, otherwise places diagnostic * 06940000
* text in *message and returns 0. * 06950000
* text in *message and returns 0. * 06960000
**/ 06970000
{
/* Local variables *****/
short int func_status = OK; /* function status indicator */ 07010000
07020000
/*** 07030000
* Use fmtOut to reformat date from year, month, and day tokens * 07040000
**/ 07050000
if(strcmp(fmtOut,"D MONTH YY") == MATCH)
{
 remove0prefix(dy); /* strip leading 0 if day < 10*/ 07080000
 nameMonth(mo); /* convert month no. to name */ 07090000
 removeCentury(yr); /* strip century from year */ 07100000
 buildDate(dateOut, dy, mo, yr, " ");
07110000
07120000
}
else if(strcmp(fmtOut,"DD MONTH YY") == MATCH)
{
 add0prefix(dy); /* add leading 0 if day < 10 */ 07150000
 nameMonth(mo); /* convert month no. to name */ 07160000
 removeCentury(yr); /* strip century from year */ 07170000
 buildDate(dateOut, dy, mo, yr, " ");
07180000
07190000
}
else if(strcmp(fmtOut,"D MONTH YYYY") == MATCH)
{
 remove0prefix(dy); /* strip leading 0 if day < 10*/ 07220000
 nameMonth(mo); /* convert month no. to name */ 07230000
 addCentury(yr); /* ensure year has century */ 07240000
 buildDate(dateOut, dy, mo, yr, " ");
07250000
07260000
}
else if(strcmp(fmtOut,"DD MONTH YYYY") == MATCH)
{
 add0prefix(dy); /* add leading 0 if day < 10 */ 07290000
 nameMonth(mo); /* convert month no. to name */ 07300000
 addCentury(yr); /* ensure year has century */ 07310000
 buildDate(dateOut, dy, mo, yr, " ");
07320000
07330000
}
else if(strcmp(fmtOut,"D.M.YY") == MATCH)
{
 remove0prefix(dy); /* strip leading 0 if day < 10*/ 07360000
 remove0prefix(mo); /* strip leading 0 if mon < 10*/ 07370000
 removeCentury(yr); /* strip century from year */ 07380000
 buildDate(dateOut, dy, mo, yr, ". ");
07390000
07400000
}
else if(strcmp(fmtOut,"DD.MM.YY") == MATCH)
{
 add0prefix(dy); /* add leading 0 if day < 10 */ 07430000
 add0prefix(mo); /* add leading 0 if mon < 10 */ 07440000
 removeCentury(yr); /* strip century from year */ 07450000
 buildDate(dateOut, dy, mo, yr, ". ");
07460000
07470000
}
else if(strcmp(fmtOut,"D-M-YY") == MATCH)
{
 remove0prefix(dy); /* strip leading 0 if day < 10*/ 07500000
 remove0prefix(mo); /* strip leading 0 if mon < 10*/ 07510000
 removeCentury(yr); /* strip century from year */ 07520000
 buildDate(dateOut, dy, mo, yr, "- ");
07530000
07540000
}
else if(strcmp(fmtOut,"DD-MM-YY") == MATCH)
{
 add0prefix(dy); /* add leading 0 if day < 10 */ 07570000
 add0prefix(mo); /* add leading 0 if mon < 10 */ 07580000
 removeCentury(yr); /* strip century from year */ 07590000
 buildDate(dateOut, dy, mo, yr, "- ");
07600000
07610000
}
else if(strcmp(fmtOut,"D/M/YY") == MATCH)
{
 remove0prefix(dy); /* strip leading 0 if day < 10*/ 07640000
 remove0prefix(mo); /* strip leading 0 if mon < 10*/ 07650000
 removeCentury(yr); /* strip century from year */ 07660000
 buildDate(dateOut, dy, mo, yr, "/");
07670000
07680000
}
else if(strcmp(fmtOut,"DD/MM/YY") == MATCH)
{
 add0prefix(dy); /* add leading 0 if day < 10 */ 07710000
 add0prefix(mo); /* add leading 0 if mon < 10 */ 07720000
}

```



```

removeCentury(yr); /* strip century from year */ 08550000
add0prefix(mo); /* add leading 0 if mon < 10 */ 08560000
add0prefix(dy); /* add leading 0 if day < 10 */ 08570000
buildDate(dateOut, yr, mo, dy, "/");
 08580000
 08590000
}
else if(strcmp(fmtOut,"YY.M.D") == MATCH)
{
 removeCentury(yr); /* strip century from year */ 08620000
 remove0prefix(mo); /* strip leading 0 if mon < 10*/ 08630000
 remove0prefix(dy); /* strip leading 0 if day < 10*/ 08640000
 buildDate(dateOut, yr, mo, dy, ".");
 08650000
 08660000
}
else if(strcmp(fmtOut,"YY.MM.DD") == MATCH)
{
 removeCentury(yr); /* strip century from year */ 08670000
 add0prefix(mo); /* add leading 0 if mon < 10 */ 08680000
 add0prefix(dy); /* add leading 0 if day < 10 */ 08690000
 buildDate(dateOut, yr, mo, dy, ".");
 08700000
 08710000
 08720000
 08730000
}
else if(strcmp(fmtOut,"YYYY/M/D") == MATCH)
{
 addCentury(yr); /* ensure year has century */ 08740000
 remove0prefix(mo); /* strip leading 0 if mon < 10*/ 08750000
 remove0prefix(dy); /* strip leading 0 if day < 10*/ 08760000
 buildDate(dateOut, yr, mo, dy, "/");
 08770000
 08780000
 08790000
 08800000
}
else if(strcmp(fmtOut,"YYYY/MM/DD") == MATCH)
{
 addCentury(yr); /* ensure year has century */ 08810000
 add0prefix(mo); /* add leading 0 if mon < 10 */ 08820000
 add0prefix(dy); /* add leading 0 if day < 10 */ 08830000
 buildDate(dateOut, yr, mo, dy, "/");
 08840000
 08850000
 08860000
 08870000
}
else if(strcmp(fmtOut,"YYYY.M.D") == MATCH)
{
 addCentury(yr); /* ensure year has century */ 08880000
 remove0prefix(mo); /* strip leading 0 if mon < 10*/ 08890000
 remove0prefix(dy); /* strip leading 0 if day < 10*/ 08900000
 buildDate(dateOut, yr, mo, dy, ".");
 08910000
 08920000
 08930000
 08940000
}
else if(strcmp(fmtOut,"YYYY.MM.DD") == MATCH)
{
 addCentury(yr); /* ensure year has century */ 08950000
 add0prefix(mo); /* add leading 0 if mon < 10 */ 08960000
 add0prefix(dy); /* add leading 0 if day < 10 */ 08970000
 buildDate(dateOut, yr, mo, dy, ".");
 08980000
 08990000
 09000000
 09010000
}
else if(strcmp(fmtOut,"YYYY-M-D") == MATCH)
{
 addCentury(yr); /* ensure year has century */ 09020000
 remove0prefix(mo); /* strip leading 0 if mon < 10*/ 09030000
 remove0prefix(dy); /* strip leading 0 if day < 10*/ 09040000
 buildDate(dateOut, yr, mo, dy, "-");
 09050000
 09060000
 09070000
 09080000
}
else if(strcmp(fmtOut,"YYYY-MM-DD") == MATCH)
{
 addCentury(yr); /* ensure year has century */ 09090000
 add0prefix(mo); /* add leading 0 if mon < 10 */ 09100000
 add0prefix(dy); /* add leading 0 if day < 10 */ 09110000
 buildDate(dateOut, yr, mo, dy, "-");
 09120000
 09130000
 09140000
 09150000
}
else if(strcmp(fmtOut,"YYYY-D-XX") == MATCH)
{
 addCentury(yr); /* ensure year has century */ 09160000
 remove0prefix(dy); /* strip leading 0 if day < 10*/ 09170000
 romanMonth(mo); /* convert month# to roman no.*/ 09180000
 buildDate(dateOut, yr, dy, mo, "-");
 09190000
 09200000
 09210000
 09220000
}
else if(strcmp(fmtOut,"YYYY-DD-XX") == MATCH)
{
 addCentury(yr); /* ensure year has century */ 09230000
 add0prefix(dy); /* add leading 0 if day < 10 */ 09240000
 romanMonth(mo); /* convert month# to roman no.*/ 09250000
 buildDate(dateOut, yr, dy, mo, "-");
 09260000
 09270000
 09280000
 09290000
}
else if(strcmp(fmtOut,"YYYY-XX-D") == MATCH)
{
 addCentury(yr); /* ensure year has century */ 09300000
 romanMonth(mo); /* convert month# to roman no.*/ 09310000
 remove0prefix(dy); /* strip leading 0 if day < 10*/ 09320000
 buildDate(dateOut, yr, mo, dy, "-");
 09330000
 09340000
 09350000
 09360000
}

```



```

(char *monthIn /* in/out: month to convert */ 10190000
)
/*** 10200000
* Converts *monthIn from a name to a number string. Returns 1 if * 10210000
* conversion succeeds, otherwise returns 0. * 10220000
** 10230000
{
/* Local variables */ 10240000
10250000
10260000
10270000
short int i; /* loop control */ 10280000
short int func_status = OK; /* function status indicator */ 10290000
10300000
/*** 10310000
* Make 1st char of month name upper case and the rest lower case * 10320000
** 10330000
monthIn[0] = toupper(monthIn[0]);
for(i=1;i<strlen(monthIn);i++) 10340000
 monthIn[i] = tolower(monthIn[i]); 10350000
10360000
10370000
/*** 10380000
* Look up *monthIn in the month names array * 10390000
** 10400000
for(i=0; i<12 && strcmp(monthIn,monthNames[i]) != MATCH; i++); 10410000
10420000
/*** 10430000
* If found assign month no. str else set function error indicator * 10440000
** 10450000
if(i < 12)
 strcpy(monthIn,monthNums[i]);
else
 func_status = NOT_OK;
return(func_status);
}
/* end unameMonth */

int romanMonth /* converts month# to roman# */ 10560000
(char *monthIn /* in/out: month to convert */ 10570000
)
/*** 10580000
* Converts *monthIn from a number string to a roman numeral. Returns * 10590000
* 1 if conversion succeeds, otherwise returns 0. * 10600000
* 1 if conversion succeeds, otherwise returns 0. * 10610000
** 10620000
{
/* Local variables */ 10630000
10640000
10650000
short int i; /* loop control */ 10660000
short int func_status = OK; /* function status indicator */ 10670000
10680000
/*** 10690000
* Strip leading zero (if any) from monthIn * 10700000
** 10710000
remove0prefix(monthIn);
10720000
10730000
/*** 10740000
* Look up *monthIn in the month number strings array * 10750000
** 10760000
for(i=0; i<12 && strcmp(monthIn,monthNums[i]) != MATCH; i++); 10770000
10780000
/*** 10790000
* If found assign roman numeral else set function error indicator * 10800000
** 10810000
if(i < 12)
 strcpy(monthIn,romanNums[i]);
else
 func_status = NOT_OK;
return(func_status);
}
/* end romanMonth */

int unromanMonth /* converts roman# to month# */ 10920000
(char *monthIn /* in/out: month to convert */ 10930000
)
/*** 10940000
* Converts *monthIn from a roman numeral to a number string. Returns * 10950000
* 1 if conversion succeeds, otherwise returns 0. * 10960000
* 1 if conversion succeeds, otherwise returns 0. * 10970000
** 10980000
{
/* Local variables */ 10990000
11000000

```

```

short int i; /* loop control */ 11010000
short int func_status = OK; /* function status indicator */ 11020000
 11030000
 11040000
/****** */
* Convert all chars of *monthIn to upper case * 11050000
***** */ 11060000
11070000
for(i=0; i<strlen(monthIn); i++) 11080000
 monthIn[i] = toupper(monthIn[i]); 11090000
 11100000
/****** */
* Look up *monthIn in the roman numerals array * 11110000
***** */ 11120000
11130000
for(i=0; i<12 && strcmp(monthIn,romanNums[i]) != MATCH; i++); 11140000
 11150000
/****** */
* If found assign month no. str else set function error indicator */ 11160000
***** */ 11170000
11180000
if(i < 12) 11190000
 strcpy(monthIn,monthNums[i]);
else
 func_status = NOT_OK;
 11200000
 11210000
11220000
11230000
return(func_status);
11240000
11250000
11260000
11270000
11280000
int checkYear /* verify/standardize yearIn */ 11290000
(char *yearOut, /* out: 4-digit yr, validated */ 11300000
 char *yearIn, /* in: 2- or 4-digit year */ 11310000
 char *style; /* in: style of yearIn */ 11320000
)
11330000
/****** */
* Verifies that *yearIn is either of the following: * 11340000
* - 2 numeric characters if *style is YY; or * 11350000
* - 4 numeric characters if *style is YYYY. * 11360000
* If criteria satisfied, copies *yearIn to *yearOut and returns 1. * 11370000
* If criteria not satisfied, sets *yearOut to null and returns 0. * 11380000
***** */ 11390000
11400000
11410000
{
 /****** Local variables ******/ 11420000
11430000
 short int i; /* loop control */ 11440000
 short int yearIn_len; /* length of *yearIn */ 11450000
 = strlen(yearIn);
 short int func_status = OK; /* function status indicator */ 11460000
 11470000
11480000
/****** */
* Verify that all bytes of *yearIn are numeric characters * 11490000
***** */ 11500000
11510000
for(i=0; (i<yearIn_len) && (isdigit(yearIn[i])); i++);
11520000
if(i < yearIn_len)
 func_status = NOT_OK;
11530000
11540000
/****** */
* If input format is YY, verify that *yearIn has 2 bytes * 11550000
***** */ 11560000
11570000
else if((strcmp(style,"YY") == MATCH) && (yearIn_len != 2))
 func_status = NOT_OK;
11580000
11590000
/****** */
* If input format is YYYY, verify that *yearIn has 4 bytes * 11600000
***** */ 11610000
11620000
else if((strcmp(style,"YYYY") == MATCH) && (yearIn_len != 4))
 func_status = NOT_OK;
11630000
11640000
11650000
/****** */
* If all checks satisfied, copy *yearIn to *yearOut and return 1 * 11660000
***** */ 11670000
11680000
if(func_status == OK)
 strcpy(yearOut, yearIn);
11690000
11700000
/****** */
* If a check failed, sets *yearOut to null and return 0 * 11710000
***** */ 11720000
11730000
else
 *yearOut = NULLCHAR;
11740000
11750000
11760000
11770000
11780000
11790000
11800000
int checkMonth /* verify/standardize monthIn */ 11810000
(char *monthOut, /* out: month#, validated */ 11820000

```

```

char *monthIn, /* in: month name, #, roman# */ 11830000
char *style, /* in: style of monthIn */ 11840000
)
/*** 11850000
* Verifies that *monthIn is one of the following: * 11870000
* - A valid month name, January - December, if *style is MONTH; or * 11880000
* - A valid roman numeral, I - XII, if *style is XX; or * 11890000
* - 1 or 2 numeric characters between 1 and 12 if *style is M or MM. * 11900000
* If criteria satisfied, copies *monthIn to *monthOut and returns 1. * 11910000
* - if *monthIn is a month name or a roman numeral, it will have * 11920000
* been standardized to the form 1-12. * 11930000
* If criteria not satisfied, sets *monthOut to null and returns 0. * 11940000
***/ 11950000
{
/* Local variables *****/ 11960000
11980000
short int i; /* loop control */ 11990000
short int func_status = OK; /* function status indicator */ 12000000
12010000
/** 12020000
* If *style is MONTH, verify that *monthIn is a valid month name * 12030000
**/ 12040000
if(strcmp(style,"MONTH") == MATCH) 12050000
 func_status = unnameMonth(monthIn); 12060000
/** 12070000
* If *style is XX, verify that *monthIn is a roman numeral, I - XII* 12080000
**/ 12090000
else if(strcmp(style,"XX") == MATCH) 12100000
 func_status = unromanMonth(monthIn); 12110000
/** 12120000
* Otherwise, verify that *monthIn is valid month number, 1 - 12 * 12130000
**/ 12140000
else 12150000
{
 remove0prefix(monthIn); /* strip any leading zero */ 12170000
 for(i=0; i<12 && strcmp(monthIn,monthNums[i]) != MATCH; i++); 12180000
 if(i >= 12) 12190000
 func_status = NOT_OK; 12200000
}
/** 12220000
* If all checks satisfied, copy *monthIn to *monthOut and return 1 * 12230000
**/ 12240000
if(func_status == OK) 12250000
 strcpy(monthOut, monthIn); 12260000
/** 12270000
* If a check failed, set *monthOut to null and return 0 * 12280000
**/ 12290000
else 12300000
 *monthOut = NULLCHAR; 12310000
12320000
 return(func_status); 12330000
} /* end checkMonth */
12340000
12350000
12360000
int checkDay /* verify/standardize dayIn */ 12370000
(char *dayOut, /* out: day, validated */ 12380000
 char *dayIn, /* in: day number */ 12390000
)
/** 12410000
* Verifies that *dayIn is either 1 or 2 numeric characters. * 12420000
* If criteria satisfied, copies *dayIn to *dayOut and returns 1. * 12430000
* If criteria not satisfied, set *dayOut to null and returns 0. * 12440000
**/ 12450000
{
/* Local variables *****/ 12460000
12480000
short int i; /* loop control */ 12490000
short int dayIn_len; /* length of *dayIn */ 12500000
 = strlen(dayIn); 12510000
short int func_status = OK; /* function status indicator */ 12520000
12530000
/** 12540000
* Verify that *dayIn is 1 or 2 numeric characters * 12550000
**/ 12560000
for(i=0; (i<dayIn_len) && (isdigit(dayIn[i])); i++); 12570000
if(i < dayIn_len || dayIn_len < 1 || dayIn_len > 2) 12580000
 func_status = NOT_OK; 12590000
/** 12600000
* If all checks satisfied, copy *dayIn to *dayOut and return 1 * 12610000
**/ 12620000
if(func_status == OK) 12630000
 strcpy(dayOut, dayIn);
12640000

```

```

/***** Local variables *****/
 *dayOut = NULLCHAR;
 return(func_status);
} /* end checkDay */

void add0prefix
(char *str3 /* in/out: string to prefix */
)
/***** Local variables *****/
{
 /* Local variables */
 if(strlen(str3) == 1)
 {
 str3[1] = str3[0]; /* Right-shift *str3 1 byte */
 str3[0] = *char0; /* Prefix it with "0" */
 str3[2] = NULLCHAR; /* And terminate it */
 }
} /* end add0prefix */

void remove0prefix /* strips leading zeroes */
(char *string /* in/out: string to strip */
)
/***** Local variables *****/
{
 if(strncmp(string,"0",1) == MATCH)
 {
 string[0] = string[1]; /* Left-shift *string */
 string[1] = NULLCHAR; /* And terminate it */
 }
} /* end remove0prefix */

void addCentury /* adds century to yearIn */
(char *yearIn /* in/out: year */
)
/***** Local variables *****/
{
 /* Local variables */
 time_t t; /* receives calendar time */
 struct tm *timeptr; /* receives local time */
 char centyear[4]; /* receives current century */
 if(strlen(yearIn) == 2)
 {
 t = time(NULL); /* Get calendar time from sys */
 timeptr = localtime(&t); /* Convert to local time */
 strftime(centyear,
 sizeof(centyear)-1,
 "%Y",
 timeptr); /* Format current century year*/
 /* ..sized for receiving field*/
 /* ..as century year */
 /* ..from current local time */
 }
 yearIn[3] = yearIn[1]; /* Prefix *yearIn with century*/
 yearIn[2] = yearIn[0]; /* ..Right-shift *yearIn */
 yearIn[1] = centyear[1]; /* ..by 2 bytes */
 yearIn[0] = centyear[0]; /* ..Place the century portion*/
 yearIn[4] = NULLCHAR; /* ..in bytes 1-2 */
} /* end addCentury */

```

```

void removeCentury /* strip century from yearIn */ 13470000
(char *yearIn /* in/out: year */ 13480000
)
/******
* Strips the century portion from *yearIn if it consists of 4 bytes. *
******/ 13490000
13500000
13510000
13520000
13530000
13540000
13550000
13560000
13570000
13580000
13590000
13600000
13610000
13620000
13630000
13640000
} /* end removeCentury */

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8DUCT

Convierte una hora determinada de uno a otro de estos 8 formatos.

```

***** 00010000
* Module name = DSN8DUCT (DB2 sample program) 00020000
* 00030000
* DESCRIPTIVE NAME = General time reformatter (UDF) 00040000
* 00050000
* LICENSED MATERIALS - PROPERTY OF IBM 00060000
* 5675-DB2 00109990
* (C) COPYRIGHT 2000 IBM CORP. ALL RIGHTS RESERVED. 00149980
* 00190000
* STATUS = VERSION 7 00200000
* 00210000
* Function: Converts a given time from one to another of these 8 00220000
* formats: 00230000
* 00240000
* H:MM AM/PM HH:MM AM/PM HH:MM:SS AM/PM HH:MM:SS 00250000
* H.MM HH.MM H.MM.SS HH.MM.SS 00260000
* 00270000
* where: 00280000
* 00290000
* H: Suppress leading zero if the hour is less than 10 00300000
* HH: Retain leading zero if the hour is less than 10 00310000
* M: Suppress leading zero if the minute is less than 10 00320000
* MM: Retain leading zero if the minute is less than 10 00330000
* AM/PM: Return time in 12-hour clock format, else 24-hour 00340000
* 00350000
* Example invocation: 00360000
* EXEC SQL SET :then = ALTTIME("01:34:59 PM", 00370000
* "HH:MM:SS AM/PM", 00380000
* "H.MM"); 00390000
* ==> then = "13.34" 00400000
* 00410000
* Notes: 00420000
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher 00430000
* 00440000
* Restrictions: 00450000
* 00460000
* Module type: C program 00470000
* Processor: IBM C/C++ for OS/390 V1R3 or higher 00480000
* Module size: See linkedit output 00490000
* Attributes: Re-entrant and re-usable 00500000
* 00510000
* Entry Point: DSN8DUCT 00520000
* Purpose: See Function 00530000
* Linkage: DB2SQL 00540000
* Invoked via SQL UDF call 00550000
* 00560000
* Input: Parameters explicitly passed to this function: 00570000
* - *timeIn : pointer to a char[12], null-termi- 00580000
* nated string having a time in the 00590000
* format indicated by *formatIn. 00600000
* - *formatIn : pointer to a char[15], null-termi- 00610000

```

```

* nated string having the format of * 00620000
* time found in *timeIn (see "Func- * 00630000
* tion", above, for valid formats). * 00640000
* - *formatOut : pointer to a char[15], null-termi- * 00650000
* nated string having the format to * 00660000
* which the time found in *timeIn is * 00670000
* to be converted. See "Function", * 00680000
* above, for valid formats. * 00690000
* - *niTimeIn : pointer to a short integer having * 00700000
* the null indicator variable for * 00710000
* *timeIn. * 00720000
* - *niFormatIn : pointer to a short integer having * 00730000
* the null indicator variable for * 00740000
* *formatIn. * 00750000
* - *niFormatOut : pointer to a short integer having * 00760000
* the null indicator variable for * 00770000
* *formatOut. * 00780000
* - *fnName : pointer to a char[138], null-termi- * 00790000
* nated string having the UDF family * 00800000
* name of this function. * 00810000
* - *specificName: pointer to a char[129], null-termi- * 00820000
* nated string having the UDF specific * 00830000
* name of this function. * 00840000
* * 00850000
* * 00860000
* * 00870000
* Output: Parameters explicitly passed by this function:
* - *timeOut : pointer to a char[15], null-termi- * 00880000
* nated string to receive the refor- * 00890000
* matted time. * 00900000
* - *niTimeOut : pointer to a short integer to re- * 00910000
* ceive the null indicator variable * 00920000
* for *timeOut. * 00930000
* - *sqlstate : pointer to a char[06], null-termi- * 00940000
* nated string to receive the SQLSTATE.* 00950000
* - *message : pointer to a char[70], null-termi- * 00960000
* nated string to receive a diagnostic * 00970000
* message if one is generated by this * 00980000
* function. * 00990000
* * 01000000
* Normal Exit: Return Code: SQLSTATE = 00000
* - Message: none * 01010000
* * 01020000
* * 01030000
* Error Exit: Return Code: SQLSTATE = 38601
* - Message: DSN8DUCT Error: No input time entered * 01040000
* - Message: DSN8DUCT Error: No input format entered * 01050000
* - Message: DSN8DUCT Error: No output format entered * 01060000
* * 01070000
* * 01080000
* Return Code: SQLSTATE = 38602
* - Message: DSN8DUCT Error: Unknown input format * 01090000
* specified * 01100000
* - Message: DSN8DUCT Error: Inputted time must indi- * 01110000
* cate either AM or PM * 01120000
* - Message: DSN8DUCT Error: Hour not in expected range * 01130000
* of 1-12 * 01140000
* - Message: DSN8DUCT Error: Hour not in expected range * 01150000
* of 0-23 * 01160000
* - Message: DSN8DUCT Error: Minute must be 2 numerics * 01170000
* between 00 and 59 * 01180000
* to input format * 01190000
* - Message: DSN8DUCT Error: Second must be 2 numerics * 01200000
* between 00 and 59 * 01210000
* to input format * 01220000
* * 01230000
* * 01240000
* Return Code: SQLSTATE = 38603
* - Message: DSN8DUCT Error: Unknown output format * 01250000
* specified * 01260000
* * 01270000
* * 01280000
* External References:
* - Routines/Services: None * 01290000
* - Data areas : None * 01300000
* - Control blocks : None * 01310000
* * 01320000
* * 01330000
* * 01340000
* Pseudocode:
* DSN8DUCT:
* - Issue sqlstate 38601 and a diagnostic message if no input time * 01350000
* was provided. * 01360000
* - Issue sqlstate 38601 and a diagnostic message if no input for- * 01370000
* mat was provided. * 01380000
* - Issue sqlstate 38601 and a diagnostic message if no output * 01390000
* format was provided. * 01400000
* - Call decontime to deconstruct the input time into hour, minute, * 01410000
* * 01420000
* - Call decontime to deconstruct the input time into hour, minute, * 01430000

```

```

* and, if either, second and AM/PM indicator, according to the * 01440000
* input format. * 01450000
* - Call reftime to create an output time from the hour, minute, * 01460000
* and, if either, second and AM/PM indicator, according to the * 01470000
* output format. * 01480000
* - If no errors, unset null indicators, and return SQLSTATE 00000 * 01490000
* else set null indicator and return null time out. * 01500000
* End DSN8DUCT * 01510000
* * 01520000
* deconTime * 01530000
* - Parse hour, minute, and, if either, second and AM/PM indicator * 01540000
* from the input time by breaking on delimiters (: and .). * 01550000
* - Use the input format to determine sequence of time components. * 01560000
* - if format invalid, issue SQLSTATE 38602 and a diag. message * 01570000
* - Call checkHour to validate the hour component and to standard- * 01580000
* ize it (if required) from a 12-hour clock to a 24-hour clock. * 01590000
* - if not valid hour, issue SQLSTATE 38602 and a diag. message * 01600000
* - Call checkMinute to validate the minute component * 01610000
* - if not valid minute, issue SQLSTATE 38602 and a diag. msg. * 01620000
* - If applicable, call checkSecond to validate the second comp. * 01630000
* - if not valid second, issue SQLSTATE 38602 and a diag. msg. * 01640000
* - If applicable, call checkAMPMIndicator to validate the AM/PM * 01650000
* indicator * 01660000
* - if not valid indicator, issue SQLSTATE 38602 and a diag. msg.* 01670000
* End deconTime * 01680000
* * 01690000
* reconTime * 01700000
* - Use the output format to edit the time components * 01710000
* - call set12HrClock to convert the hours component from 24- * 01720000
* hour clock format, as appropriate * 01730000
* - call remove0prefix to strip the leading 0 from the hour com- * 01740000
* ponent, if appropriate * 01750000
* - if output format is invalid, issue SQLSTATE 38603 and a * 01760000
* diagnostic message * 01770000
* - Call buildTime to create the output time from the edited time * 01780000
* components * 01790000
* End reconTime * 01800000
* * 01810000
* buildTime * 01820000
* - Generate the time out by concatenating the time components * 01830000
* (hour, minute, and, optionally, second and/or AM/PM indicator) * 01840000
* with intervening delimiters (: or .). * 01850000
* End buildTime * 01860000
* * 01870000
* checkHour * 01880000
* - Verify that the hour component of the input time is: * 01890000
* - in the range 01 - 12 if the input format carries an AM/PM * 01900000
* indicator * 01910000
* - call set24HrClock to standardize the hour to a 24-hour * 01920000
* clock format. * 01930000
* - in the range 00 - 23 if the input format does not carry an * 01940000
* AM/PM indicator * 01950000
* - If not valid, set error flag and return null value for hour * 01960000
* End checkHour * 01970000
* * 01980000
* checkMinute * 01990000
* - Verify the minute component is 2 digits ranging from 00 - 59. * 02000000
* - If not valid, set error flag and return null value for minute * 02010000
* End checkMinute * 02020000
* * 02030000
* checkSecond * 02040000
* - Verify the second component is 2 digits ranging from 00 - 59. * 02050000
* - If not valid, set error flag and return null value for second * 02060000
* End checkSecond * 02070000
* * 02080000
* checkAMPMIndicator * 02090000
* - Verify the AM/PM indicator is either "AM" or "PM" * 02100000
* - If not valid, set error flag and return null value for * 02110000
* AM/PM indicator * 02120000
* End checkAMPMIndicator * 02130000
* * 02140000
* set12HrClock * 02150000
* - Convert a 24-hour clock hour to a 12-hour clock hour * 02160000
* End set12HrClock * 02170000
* * 02180000
* set24HrClock * 02190000
* - Convert a 12-hour clock hour to a 24-hour clock hour * 02200000
* End set24HrClock * 02210000
* * 02220000
* add0prefix * 02230000
* - prepend an hour with a leading 0 if it is less than 10 * 02240000
* End add0prefix * 02250000

```

```

/*
 * remove0prefix
 * - strip leading zero from an hour if it is less than 10
 * End remove0prefix
 *

 * 02260000
 * 02270000
 * 02280000
 * 02290000
 * 02300000
 * 02310000

 * 02320000
 * 02321990
 * 02323980
 * 02325970
 #pragma linkage(DSN8DUCT,fetchable)
 **** C library definitions ****
#include <stdio.h> 02330000
#include <string.h> 02340000
#include <time.h> 02350000
#include <ctype.h> 02360000
 **** Equates ****
#define NULLCHAR '\0' /* Null character */ 02400000
#define MATCH 0 /* Comparison status: Equal */ 02410000
#define NOT_OK 1 /* Run status indicator: Error */ 02420000
#define OK 0 /* Run status indicator: Good */ 02430000
02440000
 **** Global constants ****
char *clock12[24] /* map of 12-hour clock */ 02450000
= {"12", "01", "02", "03", "04", "05",
 "06", "07", "08", "09", "10", "11"};
02470000
02480000
02490000
char *clock24[24] /* map of 24-hour clock */ 02500000
= {"00", "01", "02", "03", "04", "05",
 "06", "07", "08", "09", "10", "11",
 "12", "13", "14", "15", "16", "17",
 "18", "19", "20", "21", "22", "23"};
02510000
02520000
02530000
02540000
02550000
char *char0 = "0"; /* string with character "0" */ 02560000
02570000
02580000
 **** DSN8DUCT functions ****
void DSN8DUCT /* main routine */ 02590000
(char *timeIn, /* in: time to be converted */ 02610000
 char *formatIn, /* in: format of timeIn */ 02620000
 char *formatOut, /* in: format for timeOut */ 02630000
 char *timeOut, /* out: reformatted time */ 02640000
 short int *nullTimeIn, /* in: indic var for timeIn */ 02650000
 short int *nullFormatIn, /* in: indic var for formatIn */ 02660000
 short int *nullFormatOut, /* in: indic var, formatOut */ 02670000
 short int *nullTimeOut, /* out: indic var for timeOut */ 02680000
 char *sqlstate, /* out: SQLSTATE */ 02690000
 char *fnName, /* in: family name of function */ 02700000
 char *specificName, /* in: specific name of func */ 02710000
 char *message /* out: diagnostic message */ 02720000
); 02730000
02740000
02750000
int deconTime /* get hr,min,sec from timeIn */ 02760000
(char *hour, /* out: hour component */ 02770000
 char *minute, /* out: minute component */ 02780000
 char *second, /* out: second component */ 02790000
 char *message, /* out: diagnostic message */ 02800000
 char *sqlstate, /* out: SQLSTATE */ 02810000
 char *timeIn, /* in: time to deconstruct */ 02820000
 char *formatIn /* in: format of timeIn */ 02830000
); 02840000
02850000
int reconTime /* get timeOut from hr,min,sec*/ 02860000
(char *timeOut, /* out: reformatted time */ 02870000
 char *message, /* out: diagnostic message */ 02880000
 char *sqlstate, /* out: SQLSTATE */ 02890000
 char *hour, /* in: hour component */ 02900000
 char *minute, /* in: minute component */ 02910000
 char *second, /* in: second component */ 02920000
 char *formatOut /* in: format for timeOut */ 02930000
); 02940000
02950000
void buildTime /* bld timeOut from hr,min,sec*/ 02960000
(char *timeOut, /* out: reformatted time */ 02970000
 char *hour, /* in: hour component */ 02980000
 char *minute, /* in: minute component */ 02990000
 char *second, /* in: second component */ 03000000
 char *delim, /* in: delimiter */ 03010000
 char *AMPMind /* in: AM/PM indic. (if any) */ 03020000
); 03030000
03040000

```

```

int checkHour /* verify/standardize hourIn */ 03050000
(char *hourOut,
 char *hourIn,
 char *AMPMind
);

int checkMinute /* verify minute from timeIn */ 03110000
(char *minOut,
 char *minIn
);

int checkSecond /* verify second from timeIn */ 03160000
(char *secOut,
 char *secIn
);

int checkAMPMIndicator /* verify AM/PM ind. of timeIn */ 03210000
(char *indOut,
 char *indIn
);

int set12HrClock /* hour to 12-hr clock format */ 03260000
(char *hour,
 char *AMPMind
);

int set24HrClock /* hour to 24-hr clock format */ 03310000
(char *hour,
 char *AMPMind
);

void add0Pref /* add leading zero to string */ 03360000
(char *str3
);

void remove0prefix /* strip leading zeroes */ 03400000
(char *string
);

/*** main routine *****/
/*** 03450000
/*** 03460000
void DSN8DUCT
(char *timeIn, /* in: time to be converted */ 03480000
 char *formatIn, /* in: format of timeIn */ 03490000
 char *formatOut, /* in: format for timeOut */ 03500000
 char *timeOut, /* out: reformatted time */ 03510000
 short int *nullTimeIn, /* in: indic var for timeIn */ 03520000
 short int *nullFormatIn, /* in: indic var for formatIn */ 03530000
 short int *nullFormatOut, /* in: indic var, formatOut */ 03540000
 short int *nullTimeOut, /* out: indic var for timeOut */ 03550000
 char *sqlstate, /* out: SQLSTATE */ 03560000
 char *fnName, /* in: family name of function*/ 03570000
 char *specificName, /* in: specific name of func */ 03580000
 char *message /* out: diagnostic message */ 03590000
)
/*** 03610000
* * 03620000
* Assumptions: * 03630000
* - *timeIn points to a char[12], null-terminated string 03640000
* - *formatIn, points to a char[15], null-terminated string 03650000
* - *formatOut, points to a char[15], null-terminated string 03660000
* - *timeOut, points to a char[12], null-terminated string 03670000
* - *nullTimeIn, points to a short integer 03680000
* - *nullFormatIn points to a short integer 03690000
* - *nullFormatOut points to a short integer 03700000
* - *nullTimeOut points to a short integer 03710000
* - *sqlstate points to a char[06], null-terminated string 03720000
* - *fnName points to a char[138], null-terminated string 03735990
* - *specificName points to a char[129], null-terminated string 03741980
* - *message points to a char[70], null-terminated string 03750000
** 03760000
{ 03770000
 /***** Local variables *****/
 short int i; /* loop control */ 03790000
 char hour[3]; /* gets hour from timeIn */ 03800000
 char minute[3]; /* gets minute from timeIn */ 03810000
 char second[3]; /* gets second from timeIn */ 03820000
 short int status = OK; /* DSN8DUCT run status */ 03840000

```

```

***** 03870000
* Verify that an input time, its current format, and its new format* 03880000
* have been passed in. * 03890000
***** 03900000
if(*nullTimeIn || (strlen(timeIn) == 0))
{
 status = NOT_OK;
 strcpy(message,
 "DSN8DUCT Error: No input time entered");
 strcpy(sqlstate, "38601");
}
else if(*nullFormatIn || (strlen(formatIn) == 0))
{
 status = NOT_OK;
 strcpy(message,
 "DSN8DUCT Error: No input format entered");
 strcpy(sqlstate, "38601");
}
else if(*nullFormatOut || (strlen(formatOut) == 0))
{
 status = NOT_OK;
 strcpy(message,
 "DSN8DUCT Error: No output format entered");
 strcpy(sqlstate, "38601");
}
/***** 04050000
* Use formatIn to deconstruct timeIn into hour and minute and, if * 04140000
* applicable, second. * 04150000
***** 04160000
if(status == OK)
 status = deconTime(hour, minute, second, message, sqlstate,
 timeIn, formatIn);
/***** 04210000
* Use formatOut to reconstruct timeOut from ours and minute and, * 04220000
* if applicable, second. * 04230000
***** 04240000
if(status == OK)
 status = reconTime(timeOut, message, sqlstate,
 hour, minute, second, formatOut);
/***** 04290000
* If conversion was successful, clear the message buffer and sql- * 04300000
* state, and unset the SQL null indicator for timeOut. * 04310000
***** 04320000
if(status == OK)
{
 *nullTimeOut = 0;
 message[0] = NULLCHAR;
 strcpy(sqlstate,"00000");
}
/***** 04390000
* If errors occurred, clear the timeOut buffer and set the SQL null* 04400000
* indicator. A diagnostic message and the SQLSTATE have been set * 04410000
* where the error was detected. * 04420000
***** 04430000
else
{
 timeOut[0] = NULLCHAR;
 *nullTimeOut = -1;
}

return;
} /* end DSN8DUCT */
/***** 04550000
***** functions *****/
int deconTime
(
 char *hour, /* out: hour component */
 char *minute, /* out: minute component */
 char *second, /* out: second component */
 char *message, /* out: diagnostic message */
 char *sqlstate, /* out: SQLSTATE */
 char *timeIn, /* in: time to deconstruct */
 char *formatIn /* in: format of timeIn */
)
/***** 04670000
* Deconstructs *timeIn into *hour and *minute and, if applicable, *
* 04680000

```

```

* *second. The deconstruction is done according to the value in * 04690000
* formatIn. Returns OK if deconstruction succeeds, otherwise places * 04700000
* diagnostic text in *message and returns NOT_OK. * 04710000
******/ 04720000
{
 char AMPMind[3]; /* AM/PM indicator */ 04740000
 04750000
 char workTimeIn[12]; /* work copy of timeIn */ 04760000
 char *token; /* string ptr for token parser */ 04770000
 char tok1[3]; /* holds 1st time component */ 04780000
 char tok2[3]; /* holds 2nd time component */ 04790000
 char tok3[3]; /* holds 3rd time component */ 04800000
 char tok4[3]; /* holds 4th time component */ 04810000
 04820000
 short int func_status = OK; /* function status indicator */ 04830000
 short int fmtStatus = OK; /* indicates if format is OK */ 04840000
 short int hrStatus = OK; /* indicates if hour is OK */ 04850000
 short int minStatus = OK; /* indicates if minute is OK */ 04860000
 short int secStatus = OK; /* indicates if second is OK */ 04870000
 short int indStatus = OK; /* indicates if AMPMind is OK */ 04880000
 04890000
/***** 04900000
* Use C strtok function to parse the hour and minute from timeIn * 04910000
******/ 04920000
strcpy(workTimeIn,timeIn); 04930000
token = strtok(workTimeIn,".: "); 04940000
strcpy(tok1,token); 04950000
token = strtok(NULL,".: "); 04960000
strcpy(tok2,token); 04970000
 04980000
/***** 04990000
* Parse second, if any, and AM/PM indicator, if any, from timeIn * 05000000
******/ 05010000
token = strtok(NULL,".: "); 05020000
strcpy(tok3,token); 05030000
token = strtok(NULL," "); 05040000
strcpy(tok4,token); 05050000
 05060000
/***** 05070000
* Use formatIn to check and set hour, minute, etc. * 05080000
******/ 05090000
if((strcmp(formatIn,"H:MM AM/PM") == MATCH) 05100000
 || (strcmp(formatIn,"HH:MM AM/PM") == MATCH)) 05110000
{
 indStatus = checkAMPMIndicator(AMPMind,tok3); 05120000
 hrStatus = checkHour(hour,tok1,AMPMind); 05130000
 minStatus = checkMinute(minute,tok2); 05140000
 strcpy(second,"00"); 05150000
}
 05160000
else if(strcmp(formatIn,"HH:MM:SS AM/PM") == MATCH) 05170000
{
 indStatus = checkAMPMIndicator(AMPMind,tok4); 05180000
 hrStatus = checkHour(hour,tok1,AMPMind); 05190000
 minStatus = checkMinute(minute,tok2); 05200000
 secStatus = checkSecond(second,tok3); 05210000
}
 05220000
else if((strcmp(formatIn,"HH:MM:SS") == MATCH) 05230000
 || (strcmp(formatIn,"H.MM.SS") == MATCH)) 05240000
{
 hrStatus = checkHour(hour,tok1,""); 05250000
 minStatus = checkMinute(minute,tok2); 05260000
 secStatus = checkSecond(second,tok3); 05270000
}
 05280000
else if((strcmp(formatIn,"H.MM") == MATCH) 05290000
 || (strcmp(formatIn,"HH.MM") == MATCH))) 05300000
{
 hrStatus = checkHour(hour,tok1,""); 05310000
 minStatus = checkMinute(minute,tok2); 05320000
 strcpy(second,"00"); 05330000
}
 05340000
else
 fmtStatus = NOT_OK; 05350000
 05360000
/***** 05430000
* set up error handling * 05440000
******/ 05450000
func_status = NOT_OK; 05460000
strcpy(message,"DSN8DUCT Error: "); 05470000
strcpy(sqlstate, "38602"); 05480000
 05490000
/***** 05500000

```

```

* if error detected, issue diagnostic message and return NOT_OK * 05510000
*****if(fmtStatus != OK)
 strcat(message,"Unknown input format specified");
else if(indStatus != OK)
 strcat(message,"Inputted time must indicate either AM or PM");
else if(hrStatus != OK)
 if(strcmp(AMPMind,"AM") == MATCH
 || strcmp(AMPMind,"PM") == MATCH)
 strcat(message,"Hour not in expected range of 1-12");
 else
 strcat(message,"Hour not in expected range of 0-23");
else if(minStatus != OK)
 strcat(message,"minute must be 2 numerics between 00 and 59");
else if(secStatus != OK)
 strcat(message,"second must be 2 numerics between 00 and 59");
/* if no error detected, clear message and sqlstate and return OK * 05720000
*****else
{
 *message = NULLCHAR;
 func_status = OK;
 strcpy(sqlstate, "00000");
}
return(func_status);
} /* end deconTime */

int reconTime
(char *timeOut, /* out: reformatted time */ 05870000
char *message, /* out: diagnostic message */ 05880000
char *sqlstate, /* out: SQLSTATE */ 05890000
char *hour, /* in: hour component */ 05900000
char *minute, /* in: minute component */ 05910000
char *second, /* in: second component */ 05920000
char *formatOut /* in: format for timeOut */ 05930000
)
/* Reconstructs *timeOut from *hour and *minute and, if applicable, * 05940000
* *second. The reconstruction is done according to the value in * 05950000
* formatOut. Returns OK if reconstruction succeeds, otherwise * 05960000
* places diagnostic text in *message and returns NOT_OK. * 05970000
*****short int func_status = OK; /* function status indicator */ 06010000
char AMPMind[3]; /* AM/PM indicator */ 06020000
/* Use formatOut to reformat time from hour, minute, second * 06030000
*****if(strcmp(formatOut,"H:MM AM/PM") == MATCH) * 06040000
{
 set12HrClock(hour,AMPMind);
 remove0prefix(hour);
 buildTime(timeOut, hour, minute, "", ":" , AMPMind);
}
else if(strcmp(formatOut,"HH:MM AM/PM") == MATCH) * 06100000
{
 set12HrClock(hour,AMPMind);
 buildTime(timeOut, hour, minute, "", ":" , AMPMind);
}
else if(strcmp(formatOut,"HH:MM:SS AM/PM") == MATCH) * 06110000
{
 set12HrClock(hour,AMPMind);
 buildTime(timeOut, hour, minute, second, ":" , AMPMind);
}
else if(strcmp(formatOut,"HH:MM:SS") == MATCH) * 06120000
{
 buildTime(timeOut, hour, minute, second, ":" , "");
}
else if(strcmp(formatOut,"H.MM.SS") == MATCH) * 06130000
{
 remove0prefix(hour);
}

```

```

 buildTime(timeOut, hour, minute, second, ".", "");
 }
else if(strcmp(formatOut,"HH.MM.SS") == MATCH)
{
 buildTime(timeOut, hour, minute, second, ".", "");
}
else if(strcmp(formatOut,"H.MM") == MATCH)
{
 remove0prefix(hour);
 buildTime(timeOut, hour, minute, "", ".", "");
}
else if(strcmp(formatOut,"HH.MM") == MATCH)
{
 buildTime(timeOut, hour, minute, "", ".", "");
}
else
{
 func_status = NOT_OK;
 strcpy(message,"DSN8DUCT Error: ");
 strcat(message,"Unknown output format specified");
 strcpy(sqlstate, "38603");
}

return(func_status);
} /* end reconTime */

void buildTime
(char *timeOut, /* out: reformatted time */ 06620000
 char *hour, /* in: hour component */ 06630000
 char *minute, /* in: minute component */ 06640000
 char *second, /* in: second component */ 06650000
 char *delim, /* in: delimiter */ 06660000
 char *AMPMind /* in: AM/PM indic. (if any) */ 06670000
)
/*** 06680000
* Builds *timeOut from *hour, *minute, and (if specified) *second, *
* separated by the value in *delim and, if specified, suffixed by the*
* value in *AMPMind. * 06700000
* * 06710000
* * 06720000
******/ 06730000
{
/* **** 06740000
* Build timeOut from incoming time components * 06750000
* **** 06760000
******/ 06770000
strcpy(timeOut,hour); /* Start with hour ... */ 06780000
strcat(timeOut,delim); /* append the delimiter */ 06790000
strcat(timeOut,minute); /* append minute */ 06800000
if(strlen(second) > 0) /* and, if second specified, */ 06810000
{
 strcat(timeOut,delim); /* ..append the delimiter */ 06820000
 strcat(timeOut,second); /* ..append second */ 06830000
}
if(strlen(AMPMind) > 0) /* and, if AM/PM ind. spec'd */ 06840000
{
 strcat(timeOut," "); /* ..append separator blank */ 06850000
 strcat(timeOut,AMPMind); /* ..append AM/PM indicator */ 06860000
}
/* **** 06870000
* **** 06880000
* **** 06890000
* **** 06900000
* **** 06910000
* **** 06920000
* **** 06930000
* **** 06940000
******/ 06950000
int checkHour
(char *hourOut, /* out: hour (24-hour clock) */ 06960000
 char *hourIn, /* in: hour (12- or 24-hr clk) */ 06970000
 char *AMPMind /* in: AM/PM indicator */ 06980000
)
/*** 06990000
* Verifies that *hourIn meets one of these criteria: * 07010000
* - if *AMPMind is "AM" or "PM", *hourIn ranges from "01" to "12" * 07020000
* - otherwise, *hourIn ranges from "0" to "23" * 07030000
* * 07040000
* If the appropriate criterion is met, *hourOut is assigned as * 07050000
* follows: * 07060000
* - if *AMPMind is "AM" or "PM", *hourOut is assigned the 24-hour * 07070000
* clock equivalent of *hourIn. * 07080000
* - otherwise, *hourIn is copied to *hourOut * 07090000
* and checkHour returns OK. * 07100000
* * 07110000
* If the appropriate criterion is not met, *hourOut is assigned * 07120000
* NULLCHAR, and checkHour returns NOT_OK. * 07130000
******/ 07140000

```

```

{
 short int i; /* loop control */ 07150000
 short int func_status = OK; /* function status indicator */ 07160000
 /* */ 07170000
 /* */ 07180000
/****** */ 07190000
* add leading 0, if needed, to *hourIn * 07200000
***** */ 07210000
add0Pref(hourIn);
***** */ 07220000
 /* */ 07230000
***** */ 07240000
* if AMPMind is AM or PM, convert *hourIn to 24-clock format * 07250000
***** */ 07260000
if(strcmp(AMPMind,"AM") == MATCH 07270000
|| strcmp(AMPMind,"PM") == MATCH 07280000
 func_status = set24HrClock(hourIn,AMPMind);
 /* */ 07290000
 /* */ 07300000
***** */ 07310000
* if AMPMind not "AM" or "PM", verify hourIn ranges from 00 to 23 * 07320000
***** */ 07330000
else
{
 for(i=0; i<24 && strcmp(hourIn,clock24[i]) != MATCH; i++);
 if(i >= 24) /* if hourIn < 00 & > 23 */ 07370000
 func_status = NOT_OK; /* ..set error flag */ 07380000
}
 /* */ 07390000
 /* */ 07400000
***** */ 07410000
* if *hourIn is valid, copy it to *hourOut else set *hourOut to * 07420000
* NULLCHAR * 07430000
***** */ 07440000
if(func_status == OK)
 strcpy(hourOut,hourIn);
else
 hourOut[0] = NULLCHAR;
 /* */ 07450000
 /* */ 07460000
 /* */ 07470000
 /* */ 07480000
 /* */ 07490000
return(func_status);
} /* end checkHour */

int checkMinute
(char *minOut, /* out: minute, validated */ 07550000
 char *minIn /* in: minute, unvalidated */ 07560000
)
***** */ 07570000
* Verifies that *minIn is 2 bytes of numeric characters between "00" * 07590000
* and "59". * 07600000
* * 07610000
* If so, minIn is copied to minOut and checkMinute returns OK. * 07620000
* If not, NULLCHAR is copied to minOut and checkMinute returns * 07630000
* NOT_OK. * 07640000
***** */ 07650000
{
 short int i; /* loop control */ 07660000
 short int func_status = OK; /* function status indicator */ 07670000
 /* */ 07680000
 /* */ 07690000
***** */ 07700000
* verify that *minIn is 2 numeric characters between "00" and "59" * 07710000
***** */ 07720000
if(strlen(minIn) != 2)
 func_status = NOT_OK;
else if(isdigit(minIn[0]) == MATCH || isdigit(minIn[1]) == MATCH)
 func_status = NOT_OK;
else if(strcmp(minIn,"00") < 0 || strcmp(minIn,"59") > 0)
 func_status = NOT_OK;
 /* */ 07740000
 /* */ 07750000
 /* */ 07760000
 /* */ 07770000
 /* */ 07780000
 /* */ 07790000
***** */ 07800000
* if minIn is valid, assign it to minOut * 07810000
***** */ 07820000
if(func_status == OK)
 strcpy(minOut,minIn);
else
 minOut[0] = NULLCHAR;
 /* */ 07840000
 /* */ 07850000
 /* */ 07860000
 /* */ 07870000
 /* */ 07880000
 /* */ 07890000
 /* */ 07900000
 /* */ 07910000
 /* */ 07920000
int checkSecond
(char *secOut, /* out: second, validated */ 07930000
 char *secIn /* in: second, unvalidated */ 07940000
)
***** */ 07950000
 /* */ 07960000

```

```

* Verifies that *secIn is 2 bytes of numeric characters between "00" * 07970000
* and "59". * 07980000
* * 07990000
* If so, secIn is copied to secOut and checkSecond returns OK. * 08000000
* If not, NULLCHAR is copied to secOut and checkSecond returns * 08010000
* NOT_OK. * 08020000
******/ 08030000
{
 short int i; /* loop control */ 08040000
 short int func_status = OK; /* function status indicator */ 08050000
 08060000
 08070000
/***** 08080000
* verify that *secIn is 2 numeric characters between "00" and "59" * 08090000
******/ 08100000
if(strlen(secIn) != 2) 08110000
 func_status = NOT_OK; 08120000
else if(isdigit(secIn[0]) == MATCH || isdigit(secIn[1]) == MATCH) 08130000
 func_status = NOT_OK; 08140000
else if(strcmp(secIn,"00") < 0 || strcmp(secIn,"59") > 0) 08150000
 func_status = NOT_OK; 08160000
 08170000
/***** 08180000
* if secIn is valid, assign it to secOut * 08190000
******/ 08200000
if(func_status == OK) 08210000
 strcpy(secOut,secIn); 08220000
else
 secOut[0] = NULLCHAR; 08230000
 08240000
 08250000
return(func_status); 08260000
} /* end checkSecond */ 08270000
 08280000
 08290000
08300000
int checkAMPMIndicator
(char *indOut, /* out: AM/PM indic, validated*/ 08310000
 char *indIn /* in: AM/PM ind, unvalidated */ 08320000
)
/***** 08340000
* Verifies that *indIn is 2 bytes, containing either "AM" or "PM". * 08350000
* * 08360000
* If so, indIn is copied to indOut and checkAMPMIndicator returns * 08370000
* OK. * 08380000
* If not, NULLCHAR is copied to indOut and checkAMPMIndicator re- * 08390000
* turns NOT_OK. * 08400000
******/ 08410000
{
 short int i; /* loop control */ 08420000
 short int func_status = OK; /* function status indicator */ 08430000
 08440000
 08450000
/***** 08460000
* verify that *indIn is 2 bytes containing either "AM" or "PM" * 08470000
******/ 08480000
if(strlen(indIn) != 2) 08490000
 func_status = NOT_OK; 08500000
else if(strcmp(indIn,"AM") != MATCH && strcmp(indIn,"PM") != MATCH) 08510000
 func_status = NOT_OK; 08520000
 08530000
/***** 08540000
* if indIn is valid, assign it to indOut * 08550000
******/ 08560000
if(func_status == OK) 08570000
 strcpy(indOut,indIn); 08580000
else
 indOut[0] = NULLCHAR; 08590000
 08600000
return(func_status); 08610000
} /* end checkAMPMIndicator */ 08620000
 08630000
 08640000
 08650000
08660000
int set12HrClock
(char *hour, /* in/out: hour */ 08670000
 char *AMPMInd /* out: AM/PM indicator */ 08680000
)
/***** 08700000
* Changes *hour from 24-hour clock format to 12-hour clock format. * 08710000
* * 08720000
* If the incoming value for *hour is:
* - between "00" and "11", *hour is assigned the 12-hour clock * 08730000
* equivalent ("12" - "11"), *AMPMInd is assigned "AM", and * 08740000
* set12HrClock returns OK. * 08750000
* - between "12" and "23", *hour is assigned the 12-hour clock * 08760000
* equivalent ("12" - "11"), *AMPMInd is assigned "PM", and * 08770000
* set12HrClock returns OK. * 08780000

```

```

* set12HrClock returns OK. * 08790000
* - any other value, *hour is unchanged, *AMPMind is assigned * 08800000
* NULLCHAR, and set12HrClock returns NOT_OK. * 08810000
******/ 08820000
{
 short int i; /* loop control */ 08830000
 short int func_status = OK; /* function status indicator */ 08840000
 08850000
 08860000
 /***** 08870000
 * locate *hour in the 24-hour clock map * 08880000
 *****/ 08890000
 for(i=0; i<24 && strcmp(hour,clock24[i]) != MATCH; i++);
 08900000
 08910000
 /***** 08920000
 * assign *hour its 12-hour clock equivalent * 08930000
 *****/ 08940000
 if(i < 12) /* if hour betw/ "00" & "11" */ 08950000
 {
 strcpy(hour,clock12[i]); /* ..set hour in 12-hour fmt */ 08960000
 strcpy(AMPMind,"AM"); /* ..set AM/PM indic to AM */ 08970000
 08980000
 08990000
 } else if(i < 24) /* if hour betw/ "12" & "23" */ 09000000
 {
 strcpy(hour,clock12[i-12]); /* ..set hour in 12-hour fmt */ 09010000
 strcpy(AMPMind,"PM"); /* ..set AM/PM indic to PM */ 09020000
 09030000
 09040000
 } else
 {
 func_status = NOT_OK; /* ..set error flag */ 09050000
 AMPMind[0] = NULLCHAR; /* ..null out AM/PM indicator */ 09060000
 09070000
 09080000
 09090000
 09100000
 }
 return(func_status); /* return function status */ 09110000
} /* end set12HrClock */

int set24HrClock
(char *hour, /* in/out: hour */ 09120000
 char *AMPMind /* in: AM/PM indicator */ 09130000
)
***** 09140000
* Changes *hour from 12-hour clock format to 24-hour clock format. * 09150000
*
* If the incoming value for *hour is not between "01" and "12", * 09160000
* then *hour is unchanged and set24HrClock returns NOT_OK. * 09170000
*
* Otherwise:
* - if *AMPMind is "AM", then *hour is assigned the 24-hour equiva- * 09210000
* lent of morning hour ("00"- "11") and set24HrClock returns OK. * 09220000
* - else *hour is assigned the 24-hour equivalent of afternoon * 09230000
* hour ("12"- "23") and set24HrClock returns OK. * 09240000
***** 09250000
{
 short int i; /* loop control */ 09260000
 short int func_status = OK; /* function status indicator */ 09270000
 09280000
 09290000
 09300000
 09310000
 /***** 09320000
 * locate *hour in the 12-hour clock map * 09330000
 *****/ 09340000
 09350000
 * 09360000
 *****/ 09370000
 for(i=0; i<12 && strcmp(hour,clock12[i]) != MATCH; i++);
 09380000
 09390000
 /***** 09400000
 * assign *hour its 24-hour clock equivalent * 09410000
 *****/ 09420000
 if(i > 11) /* if hour not betw/ 01 & 12 */ 09430000
 {
 func_status = NOT_OK; /* ..set error flag */ 09440000
 } else if(strcmp(AMPMind,"AM")==MATCH)/* else if betw/ 12 AM - 11 AM*/ 09450000
 strcpy(hour,clock24[i]); /* ..set hour in 12-hour fmt */ 09460000
 } else
 /* else betw/ 12 PM - 11 PM */ 09470000
 strcpy(hour,clock24[i+12]); /* ..set hour in 12-hour fmt */ 09480000
 09490000
 return(func_status); /* return function status */ 09500000
} /* end set24HrClock */

void add0Pref
(char *str3 /* in/out: string to prefix */ 09510000
)
***** 09520000
* Prefixes *str3 with a leading 0 if it is only 1 byte long * 09530000
***** 09540000
* 09550000
* 09560000
***** 09570000
* 09580000
***** 09590000
* 09600000
{

```

```

***** 09610000
* if str3 is just 1 byte long, prefix it with a "0" * 09620000
***** 09630000
if(strlen(str3) == 1)
{
 str3[1] = str3[0]; /* Right-shift *str3 1 byte */ 09660000
 str3[0] = *char0; /* Prefix it with "0" */ 09670000
 str3[2] = NULLCHAR; /* And terminate it */ 09680000
}
} /* end add0Pref */

void remove0prefix
(char *string /* in/out: character string */ 09740000
)
***** 09760000
* Eliminates all leading zeroes from *str3. Leaves a single zero in *
* the first byte of *str3 if *str3 is all zeroes. * 09770000
***** 09790000
{
 short int i = 0; /* Loop control */ 09810000
 short int j = 0; /* Loop control */ 09820000
}
***** 09830000
* if leading zero in first byte, skip up to first non-zero * 09850000
***** 09860000
if(string[0] == '0')
 for(i=0; string[i] == '0'; i++);
***** 09870000
* if at end of string, it was all zeroes: put zero in 1st byte * 09910000
***** 09920000
if(string[i] == '\0')
 strcpy(string, "0");
***** 09930000
* otherwise, left-shift non-zero chars and terminate string * 09960000
***** 09970000
else
{
 for(j=0; string[i] != NULLCHAR; j++)
 string[j] = string[i++];
 string[j] = NULLCHAR;
}
} /* end remove0prefix */
***** 09980000
***** 09990000
***** 10000000
***** 10010000
***** 10020000
***** 10030000
***** 10040000

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8DUCY

Da formato a una cantidad numérica determinada con un símbolo de moneda especificado y, si se especifica, uno de los siguientes indicadores de débito/crédito.

```

***** 00000100
* Module name = DSN8DUCY (DB2 sample program) * 00000200
* * 00000300
* DESCRIPTIVE NAME = General currency formatter (UDF) * 00000400
* * 00000500
* * 00000600
* LICENSED MATERIALS - PROPERTY OF IBM * 00000700
* 5675-DB2 * 00000800
* (C) COPYRIGHT 1998, 2000 IBM CORP. ALL RIGHTS RESERVED. * 00000900
* * 00001000
* STATUS = VERSION 7 * 00001100
* * 00001200
* Function: Formats a given numeric amount with a specified currency * 00001300
* symbol and, if specified, one of the following debit/ * 00001400
* credit indicators. * 00001500
* * 00001600
* +/-: Place a hyphen between the currency symbol and the * 00001700
* amount if the amount is less than 0. * 00001800
* (): Place a left parenthesis between currency symbol * 00001900
* and the amount and place a right parenthesis to the * 00002000
* right of the amount if the amount is less than 0. * 00002100
* CR/DB: Place CR to the right of the amount if it is less * 00002200
* than 0; otherwise place DB to the right of the * 00002300
* amount. * 00002400

```

```

*
* Example invocations:
* EXEC SQL SET :money = CURRENCY(-123,
* "DM");
* ==> money = DM -123.00
*
*
* EXEC SQL SET :money = CURRENCY(-123,
* "DM",
* "(/)");
* ==> money = DM (123.00)
*
* Notes:
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher
*
* Restrictions:
*
* Module type: C program
* Processor: IBM C/C++ for OS/390 V1R3 or higher
* Module size: See linkedit output
* Attributes: Re-entrant and re-usable
*
* Entry Point: DSN8DUCY
* Purpose: See Function
* Linkage: DB2SQL
* Invoked via SQL UDF call
*
* Parameters: DSN8DUCY uses the C "main" argument convention of
* argv (argument vector) and argc (argument count).
*
* The location of input and output parameters depends
* on whether the CURRENCY UDF is invoked with two
* input arguments (amount and currency symbol) or with
* three input arguments (amount, currency symbol, and
* credit/debit indicator).
*
* If the CURRENCY UDF is invoked with two arguments
* only (an amount and a currency symbol):
* - ARGV[0] = (input) pointer to a char[9], null-
* terminated string having the name of
* this program (DSN8DUCY)
* - ARGV[1] = (input) pointer to a double word having
* the amount to be formatted as currency.
* - ARGV[2] = (input) pointer to a char[3], null-
* terminated string having the currency
* symbol.
* - ARGV[3] = (output) pointer to a char[20], null-
* terminated string to receive the cur-
* rency result.
* - ARGV[4] = (input) pointer to a short integer
* having the null indicator for the input
* amount
* - ARGV[5] = (input) pointer to a short integer
* having the null indicator for the cur-
* rency symbol
* - ARGV[6] = (output) pointer to a short integer
* having the null indicator for the result
* - ARGV[7] = (output) pointer to a char[6], null-
* terminated string to receive the
* SQLSTATE
* - ARGV[8] = (input) pointer to a char[138], null-
* terminated string having the UDF family
* name of the function
* - ARGV[9] = (input) pointer to a char[129], null-
* terminated string having the UDF
* specific name of the function
* - ARGV[10] = (output) pointer to a char[70],
* null- terminated string to receive any
* diagnostic message issued by this
* function
*
* If the CURRENCY UDF is invoked with three arguments
* (an amount, a currency symbol, and a credit/debit
* indicator):
* - ARGV[0] = (input) pointer to a char[9], null-
* terminated string having the name of
* this program (DSN8DUCY)
* - ARGV[1] = (input) pointer to a double word having
* the amount to be formatted as currency.
* - ARGV[2] = (input) pointer to a char[3], null-
* terminated string having the currency
* symbol.
*
* * 00002500
* * 00002600
* * 00002700
* * 00002800
* * 00002900
* * 00003000
* * 00003100
* * 00003200
* * 00003300
* * 00003400
* * 00003500
* * 00003600
* * 00003700
* * 00003800
* * 00003900
* * 00004000
* * 00004100
* * 00004200
* * 00004300
* * 00004400
* * 00004500
* * 00004600
* * 00004700
* * 00004800
* * 00004900
* * 00005000
* * 00005100
* * 00005200
* * 00005300
* * 00005400
* * 00005500
* * 00005600
* * 00005700
* * 00005800
* * 00005900
* * 00006000
* * 00006100
* * 00006200
* * 00006300
* * 00006400
* * 00006500
* * 00006600
* * 00006700
* * 00006800
* * 00006900
* * 00007000
* * 00007100
* * 00007200
* * 00007300
* * 00007400
* * 00007500
* * 00007600
* * 00007700
* * 00007800
* * 00007900
* * 00008000
* * 00008100
* * 00008200
* * 00008300
* * 00008400
* * 00008500
* * 00008600
* * 00008700
* * 00008800
* * 00008900
* * 00009000
* * 00009100
* * 00009200
* * 00009300
* * 00009400
* * 00009500
* * 00009600
* * 00009700
* * 00009800
* * 00009900
* * 00010000
* * 00010100
* * 00010200
* * 00010300
* * 00010400
* * 00010500
* * 00010600

```

```

* - ARGV[3] = (input) pointer to a char[6], null- * 00010700
* terminated string having the credit/ * 00010800
* debit indicator (see under "Function:", * 00010900
* above, for valid credit/debit indicators) * 00011000
* - ARGV[4] = (output) pointer to a char[20], null- * 00011100
* terminated string to receive the cur- * 00011200
* rency result. * 00011300
* - ARGV[5] = (input) pointer to a short integer * 00011400
* having the null indicator for the input * 00011500
* amount * 00011600
* - ARGV[6] = (input) pointer to a short integer * 00011700
* having the null indicator for the cur- * 00011800
* rency symbol * 00011900
* - ARGV[7] = (input) pointer to a short integer * 00012000
* having the null indicator for the * 00012100
* credit/debit indicator * 00012200
* - ARGV[8] = (output) pointer to a short integer * 00012300
* having the null indicator for the result * 00012400
* - ARGV[9] = (output) pointer to a char[6], null- * 00012500
* terminated string to receive the * 00012600
* SQLSTATE * 00012700
* - ARGV[10] = (input) pointer to a char[138], null- * 00012800
* terminated string having the UDF family * 00012900
* name of the function * 00013000
* - ARGV[11] = (input) pointer to a char[129], null- * 00013100
* terminated string having the UDF * 00013200
* specific name of the function * 00013300
* - ARGV[12] = (output) pointer to a char[70], * 00013400
* null- terminated string to receive any * 00013500
* diagnostic message issued by this * 00013600
* function * 00013700
* * 00013800
* Normal Exit: Return Code: SQLSTATE = 00000 * 00013900
* - Message: none * 00014000
* * 00014100
* Error Exit: Return Code: SQLSTATE = 38601 * 00014200
* - Message: DSN8DUCY Error: No amount entered * 00014300
* - Message: DSN8DUCY Error: No currency symbol entered * 00014400
* * 00014500
* Return Code: SQLSTATE = 38602 * 00014600
* - Message: DSN8DUCY Error: Error performing * 00014700
* setlocale() * 00014800
* * 00014900
* External References: * 00015000
* - Routines/Services: None * 00015100
* - Data areas : None * 00015200
* - Control blocks : None * 00015300
* * 00015400
* * 00015500
* * 00015600
* * 00015700
* Pseudocode: * 00015800
* DSN8DUCY: * 00015900
* - Walk down the argv list, locating the input and output parms * 00016000
* - Issue sqlstate 38601 and a diagnostic message if no input * 00016100
* amount was provided. * 00016200
* - Issue sqlstate 38601 and a diagnostic message if no currency * 00016300
* symbol was provided. * 00016400
* - Call formatAmount to assemble the currency expression from the * 00016500
* input amount and the currency symbol and, optionally, the * 00016600
* credit/debit indicator. * 00016700
* - If no errors, unset null indicators, and return SQLSTATE 00000 * 00016800
* else set null indicator and return null time out. * 00016900
* End DSN8DUCY
* formatAmount * 00017000
* - If the amount in is less than zero ... * 00017100
* - if a CR/DB indicator of -/+ has been specified, prefix * 00017200
* the currency expression with a hyphen * 00017300
* - else if a CR/DB of (/) has been specified, prefix the curr- * 00017400
* rency expression with a left parenthesis * 00017500
* - Append the currency symbol to the currency expression * 00017600
* - if the currency symbol is just 1 byte, concatenate a blank * 00017700
* - Call the C function setlocale() to initialize the C function * 00017800
* strftime(). * 00017900
* - if error, issue SQLSTATE 38602 and a diagnostic message * 00018000
* - Call the C function strftime() to reformat the input amount * 00018100
* with the currency symbol. * 00018200
* - If the amount in is less than zero ... * 00018300
* - if a CR/DB of (/) has been specified, append a right paren- * 00018400
* thesis to the currency expression. * 00018500
* End formatAmount * 00018600
* * 00018700
* * 00018800

```

```

***** C library definitions *****
#include <localdef.h> 00019000
#include <monetary.h> 00019100
#include <stdio.h> 00019200
#include <stdlib.h> 00019300
#include <string.h> 00019400
 00019500
 00019600
***** Equates *****
#define NULLCHAR '\0' /* Null character */ 00019700
 00019800
 00019900
#define MATCH 0 /* Comparison status: Equal */ 00020000
#define NOT_OK 0 /* Run status indicator: Error */ 00020100
#define OK 1 /* Run status indicator: Good */ 00020200
 00020300
 00020400
***** DSN8DUCY functions *****
int main /* main routine */ 00020500
(int argc, /* standard argument count */ 00020600
 char *argv[] /* standard argument vector */ 00020700
);
 00020800
 00020900
 00021000
int formatAmount /* format amountIn as currency */ 00021100
(char *moneyOut, /* out: formatted amountIn */ 00021200
 char *message, /* out: diagnostic message */ 00021300
 char *sqlstate, /* out: SQLSTATE */ 00021400
 double *amountIn, /* in: value to be formatted */ 00021500
 char *currSymbol, /* in: currency symbol */ 00021600
 char *CRDBInd, /* in: credit/debit indicator */ 00021700
);
 00021800
 00021900
***** main routine *****
int main /* main routine */ 00022000
(int argc, /* standard argument count */ 00022100
 char *argv[] /* standard argument vector */ 00022200
);
 00022300
 00022400
 00022500
 00022600
 00022700
*
 00022800
***** local variables *****
short int minus1 = -1; /* default null indic setting */ 00022900
 00023000
 00023100
 00023200
short int minus1 = -1; /* default null indic setting */ 00023300
 00023400
 00023500
double *amountIn; /* in: value to be formatted */ 00023600
char currSymbol[3]; /* in: currency symbol */ 00023700
char CRDBInd[6]; /* in: credit/debit indicator */ 00023800
char *moneyOut; /* out: formatted amountIn */ 00023900
short int *niAmountIn; /* in: indic var, amountIn */ 00024000
short int *niCurrSymbol; /* in: indic var, currSymbol */ 00024100
short int *niCRDBInd; /* in: indic var, CRDBInd */ 00024200
short int *niMoneyOut; /* out: indic var, moneyOut */ 00024300
char *sqlstate; /* out: SQLSTATE */ 00024400
char fnName[138]; /* in: family name of function */ 00024500
char specificName[129]; /* in: specific name of func */ 00024600
char *message; /* out: diagnostic message */ 00024700
 00024800
short int status = OK; /* DSN8DUCY run status */ 00024900
 00025000
 00025100
 00025200
***** Walk down the argv list, locating the input and output parms *****
* Walk down the argv list, locating the input and output parms * 00025300
***** convert argv to base 0 index *****
argc--; /* convert argv to base 0 index */ 00025400
 00025500
 00025600
 00025700
message = (char *)argv[argc--]; /* out: point to UDF diag msg */ 00025800
 00025900
strcpy(specificName, /* in: save UDF specific name */ 00026000
 argv[argc--]);
 00026100
 00026200
strcpy(fnName,argv[argc--]); /* in: save UDF function name */ 00026300
 00026400
sqlstate = (char *)argv[argc--]; /* out: point to UDF sqlstate */ 00026500
 00026600
niMoneyOut = (short int *)argv[argc--]; /* out: point to null indicator*/ 00026700
 00026800
 00026900
if(argc == 7) /* if 3 input parms passed */ 00027000

```

```

 niCRDBInd /* ..in: point to null indic. */ 00027100
 = (short int *)argv[argc--]; /* var for CR/DB indic. */ 00027200
else /* otherwise it wasn't passed */ 00027300
 niCRDBInd = &minus1; /* ..so define it as null */ 00027400
 00027500
niCurrSymbol /* in: point to null indicator */ 00027600
 = (short int *)argv[argc--]; /* var for currency symbol */ 00027700
 00027800
niAmountIn /* in: point to null indicator */ 00027900
 = (short int *)argv[argc--]; /* var for input amount */ 00028000
 00028100
moneyOut = (char *)argv[argc--]; /* out: point to UDF result */ 00028200
 00028300
if(argc == 3) /* if 3 input parms passed */ 00028400
 strcpy(CRDBInd, /* ..in: save object location */ 00028500
 argv[argc--]); /* name */ 00028600
else /* otherwise it wasn't passed */ 00028700
 CRDBInd[0] = NULLCHAR; /* ..so define it as null */ 00028800
 00028900
strcpy(currSymbol,argv[argc--]); /* in: save currency symbol */ 00029000
 00029100
amountIn = (double *)argv[argc]; /* in: save input amount */ 00029200
 00029300
 00029400
/*** 00029500
* Initialize output parms * 00029600
***/ 00029700
message[0] = NULLCHAR; 00029800
strcpy(sqlstate,"00000"); 00029900
*niMoneyOut = 0; 00030000
moneyOut[0] = NULLCHAR; 00030100
 00030200
/*** 00030300
* Verify that an amount and a currency symbol have been passed in * 00030400
***/ 00030500
if(*niAmountIn)
{
 status = NOT_OK; 00030600
 strcpy(message, 00030700
 "DSN8DUCY Error: No amount entered");
 strcpy(sqlstate, "38601");
}
else if(*niCurrSymbol || (strlen(currSymbol) == 0))
{
 status = NOT_OK; 00031300
 strcpy(message, 00031400
 "DSN8DUCY Error: No currency symbol entered");
 strcpy(sqlstate, "38601");
}
/*** 00032100
* Format the amount into currency notation * 00032200
***/ 00032300
if(status == OK)
 status = formatAmount(moneyOut, message, sqlstate,
 amountIn, currSymbol, CRDBInd);
 00032400
 00032500
 00032600
 00032700
/*** 00032800
* If formatting was successful, clear the message buffer and sql- * 00032900
* state, and unset the SQL null indicator for moneyOut. * 00033000
***/ 00033100
if(status == OK)
{
 *niMoneyOut = 0; 00033200
 message[0] = NULLCHAR; 00033300
 strcpy(sqlstate,"00000");
}
/*** 00033800
* If errors occurred, clear the moneyOut buff and set the SQL null * 00033900
* indicator. A diagnostic message and the SQLSTATE have been set * 00034000
* where the error was detected. * 00034100
***/ 00034200
else
{
 moneyOut[0] = NULLCHAR; 00034300
 *niMoneyOut = -1;
}
 00034400
 00034500
 00034600
 00034700
 00034800
return(0);
} /* end main */
/*** 00035200

```

```

/********************* functions ********************/ 00035300
/********************* */ 00035400
int formatAmount /* format amountIn as currency*/ 00035500
(char *moneyOut, /* out: formatted amountIn */ 00035600
 char *message, /* out: diagnostic message */ 00035700
 char *sqlstate, /* out: SQLSTATE */ 00035800
 double *amountIn, /* in: value to be formatted */ 00035900
 char *currSymbol, /* in: currency symbol */ 00036000
 char *CRDBInd /* in: credit/debit indicator */ 00036100
)
/********************* */ 00036200
* Converts amountIn to a string, including the currency type in * 00036400
* currSymbol and, if specified, the credit/debit indicator in CRDB- * 00036500
* Symbol. The result is placed in moneyOut. * 00036600
* * 00036700
* currSymbol may be any string of characters, up to 2 bytes long. * 00036800
* CRDBInd is used only its value is: * 00036900
* - "+/-", indicating that moneyOut and its prefix from currSymbol * 00037000
* should be prefixed by a hyphen ("") if amountIn is negative. * 00037100
* - "()", indicating that moneyOut should be enclosed, with the * 00037200
* prefix from currSymbol, in parentheses if amountIn is negative. * 00037300
* - "CR/DB", indicating that moneyOut should be prefixed with DB * 00037400
* if amountIn is negative, otherwise with CR. * 00037500
/********************* */ 00037600
{
 int i; /* loop control */ 00037800
 int negFlag = 0; /* negative currency flag */ 00037900
 00038000
 double amount; /* work var for amountIn */ 00038100
 char moneyStr[200]; /* work string for type conv. */ 00038200
 00038300
/********************* */ 00038400
* Clear any residual value from moneyOut * 00038500
/********************* */ 00038600
for(i=0;i<strlen(moneyOut);i++)
 moneyOut[i] = NULLCHAR;
 00038700
 00038800
 00038900
/********************* */ 00039000
* If amountIn is negative, prefix moneyOut with neg curr indicators* 00039100
/********************* */ 00039200
amount = *amountIn;
 00039300
if(amount < 0)
{
 00039400
 00039500
 negFlag = 1;
 if(CRDBInd[0] != NULLCHAR)
 {
 00039600
 amount = -amount;
 00039700
 00039800
 00039900
 00040000
 if(strcmp(CRDBInd,"+/-") == 0)
 strcpy(moneyOut,"-");
 00040100
 else if(strcmp(CRDBInd,"()") == 0)
 strcpy(moneyOut,"(");
 00040200
 00040300
 00040400
 }
 00040500
 00040600
 00040700
 }
 00040800
* Append the currency type (currSymbol) to moneyOut. If currSymbol* 00040900
* is more than one byte, place a blank between it and the amount * 00041000
/********************* */ 00041100
strcat(moneyOut,currSymbol);
 00041200
if(strlen(currSymbol) > 1)
 00041300
 strcat(moneyOut, " ");
 00041400
 00041500
/********************* */ 00041600
* Set the local for the strftime function * 00041700
/********************* */ 00041800
if(setlocale(LC_ALL, "En_US") == NULL)
 00041900
 00042000
 {
 strcpy(message,
 "DSN8DUCY Error: Error performing setlocale()");
 00042100
 strcpy(sqlstate, "38602");
 00042200
 return(NOT_OK);
 00042300
 00042400
 }
 00042500
 00042600
/********************* */ 00042700
* Reformat amount to a string type with thousands grouping * 00042800
/********************* */ 00042900
strftime(moneyStr,100,"%n",amount);
 00043000
strcat(moneyOut,moneyStr);
 00043100
 00043200
/********************* */ 00043300
* If amount < 0, append negative currency indicators, if passed * 00043400

```

```

***** ****
if(CRDBInd[0] != NULLCHAR)
{
 if(negFlag == 1)
 {
 if(strcmp(CRDBInd,"CR/DB") == 0)
 strcat(moneyOut," DB");
 else if(strcmp(CRDBInd,"(/)") == 0)
 strcat(moneyOut,")");
 }
 else
 {
 if(strcmp(CRDBInd,"CR/DB") == 0)
 strcat(moneyOut," CR");
 }
}

return(OK);
} /* end formatAmount */

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8DUTI

Devuelve el nombre o el nombre de esquema o el nombre de localización de un alias según el nombre del UDF y el número de parámetros de entrada pasados, de la siguientes maneras.

```

***** ****
* Module name = DSN8DUTI (DB2 sample program) 00010000
* * 00020000
* * 00030000
* DESCRIPTIVE NAME = Resolve a fully-qualified (3 part), partially- * 00040000
* qualified (2 part), or unqualified alias to the * 00050000
* name, schema, or location of its base table or * 00060000
* view. * 00070000
* * 00080000
* LICENSED MATERIALS - PROPERTY OF IBM * 00090000
* 5675-DB2 * 00100000
* (C) COPYRIGHT 1997, 2000 IBM CORP. ALL RIGHTS RESERVED. * 00110000
* * 00150000
* STATUS = VERSION 7 * 00160000
* * 00190000
* Function: Returns the name or the schema name or the location name * 00250000
* of an alias according to the name of the UDF and the * 00260000
* number of input parameters passed, as follows: * 00270000
* * 00280000
* * 00290000
* TABLE_NAME(objectname) * 00300000
* returns the unqualified name of the object found after * 00310000
* any alias chains have been resolved. The specified * 00320000
* object name, the default schema, and a location name * 00330000
* of "%" (for any location) are used as the starting * 00340000
* point of the resolution. If the starting point does * 00350000
* not refer to an alias, the unqualified name of the * 00360000
* starting point is returned. The resulting name may be * 00370000
* of a table, view, or undefined object. * 00380000
* * 00390000
* TABLE_NAME(objectname, objectschema) * 00400000
* returns the unqualified name of the object found after * 00410000
* any alias chains have been resolved. The specified * 00420000
* object name and schema, and a location name of "%" * 00430000
* (for any location), are used as the starting point of * 00440000
* the resolution. If the starting point does not refer * 00450000
* to an alias, the unqualified name of the starting * 00460000
* point is returned. The resulting name may be of a * 00470000
* table, view, or undefined object. * 00480000
* * 00490000
* TABLE_NAME(objectname, objectschema, objectlocation) * 00500000
* returns the unqualified name of the object found after * 00510000
* any alias chains have been resolved. The specified * 00520000
* object name, schema, and location name are used as the * 00530000
* starting point of the resolution. If the starting * 00540000
* point does not refer to an alias, the unqualified name * 00550000
* of the starting point is returned. The resulting name * 00560000
* may be of a table, view, or undefined object. * 00570000
* * 00580000
* TABLE_SCHEMA(objectname) returns the schema name of * 00590000
* the object found after any alias chains have been

```

```

* resolved. The specified object name, the default * 00600000
* schema, and a location name of "%" (for any location) * 00610000
* are used as the starting point of the resolution. If * 00620000
* the starting point does not refer to an alias, the * 00630000
* schema name of the starting point is returned. The * 00640000
* resulting schema name may be of a table, view, or * 00650000
* undefined object. * 00660000
* * 00670000
* TABLE_SCHEMA(objectname, objectschema) returns the * 00680000
* schema name of the object found after any alias chains * 00690000
* have been resolved. The specified object name and * 00700000
* schema and a location name of "%" (for any location) * 00710000
* are used as the starting point of the resolution. If * 00720000
* the starting point does not refer to an alias, the * 00730000
* schema name of the starting point is returned. The * 00740000
* resulting schema name may be of a table, view, or * 00750000
* undefined object. * 00760000
* * 00770000
* TABLE_SCHEMA(objectname, objectschema, objectlocation) * 00780000
* returns the schema name of the object found after any * 00790000
* alias chains have been resolved. The specified object * 00800000
* name, schema, and location name are used as the * 00810000
* starting point of the resolution. If the starting * 00820000
* point does not refer to an alias, the schema name of * 00830000
* starting point is returned. The resulting schema name * 00840000
* may be of a table, view, or undefined object. * 00850000
* * 00860000
* TABLE_LOCATION(objectname) returns the location name * 00870000
* of the object found after any alias chains have been * 00880000
* resolved. The specified object name, the default * 00890000
* schema, and a location name of "%" (for any location) * 00900000
* are used as the starting point of the resolution. If * 00910000
* the starting point does not refer to an alias, a blank * 00920000
* location name (indicating the current server) is * 00930000
* returned. The resulting location name may be of a * 00940000
* table, view, or undefined object. * 00950000
* * 00960000
* TABLE_LOCATION(objectname, objectschema) returns the * 00970000
* location name of the object found after any alias * 00980000
* chains have been resolved. The specified object name, * 00990000
* schema, and a location name of "%" (for any location) * 01000000
* are used as the starting point of the resolution. If * 01010000
* the starting point does not refer to an alias, a blank * 01020000
* location name (indicating the current server) is * 01030000
* returned. The resulting location name may be of a * 01040000
* table, view, or undefined object. * 01050000
* * 01060000
* TABLE_LOCATION(objectname, objectschema, objectlocation) * 01070000
* returns the location name of the object found after * 01080000
* any alias chains have been resolved. The specified * 01090000
* object name, schema, and location name are used as the * 01100000
* starting point of the resolution. If the starting * 01110000
* point does not refer to an alias, a blank location * 01120000
* name (indicating the current server) is returned. The * 01130000
* resulting location name may be of a table, view, or * 01140000
* undefined object. * 01150000
* * 01160000
* Notes: * 01170000
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher * 01180000
* * 01190000
* Restrictions: * 01200000
* * 01210000
* Module type: C program * 01220000
* Processor: IBM C/C++ for OS/390 V1R3 or subsequent release * 01230000
* Module size: See linkedit output * 01240000
* Attributes: Re-entrant and re-usable * 01250000
* * 01260000
* Entry Point: CEESTART (Language Environment entry point) * 01270000
* Purpose: See Function * 01280000
* Linkage: DB2SQL * 01290000
* Invoked via SQL UDF call * 01300000
* * 01310000
* Parameters: DSN8DUTI uses the C "main" argument convention of * 01320000
* argv (argument vector) and argc (argument count). * 01330000
* * 01340000
* * 01350000
* * 01360000
* * 01370000
* * 01380000
* * 01390000
* If the UDF was invoked with the object name only: * 01400000
* - ARGV[0] = (input) pointer to a char[9], null- * 01410000

```

```

* terminated string having the name of * 01420000
* this program (DSN8DUTI) * 01430000
* - ARGV[1] = (input) pointer to a char[19], null- * 01440000
* terminated string having the object name * 01450000
* to be used as the starting point of the * 01460000
* alias resolution * 01470000
* - ARGV[2] = (output) pointer to a null-terminated * 01480000
* string to receive the result as follows: * 01490000
* - char[19] for the TABLE_NAME UDF * 01500000
* - char[9] for the TABLE_SCHEMA UDF * 01510000
* - char[17] for the TABLE_LOCATION UDF * 01520000
* - ARGV[3] = (input) pointer to a short integer * 01530000
* having the null indicator for the object * 01540000
* name * 01550000
* - ARGV[4] = (output) pointer to a short integer * 01560000
* having the null indicator for the result * 01570000
* - ARGV[5] = (output) pointer to a char[6], null- * 01580000
* terminated string to receive the * 01590000
* SQLSTATE * 01600000
* - ARGV[6] = (input) pointer to a char[138], null- * 01610000
* terminated string having the UDF family * 01620000
* name of the function * 01630000
* - ARGV[7] = (input) pointer to a char[129], * 01645990
* null-terminated * 01651980
* string having the UDF specific name of * 01660000
* the function * 01670000
* - ARGV[8] = (output) pointer to a char[70], * 01680000
* null-terminated string to receive any * 01690000
* diagnostic message issued by this * 01700000
* function * 01710000
* * 01720000
* If the UDF was invoked with the object name and the * 01730000
* object schema (but not the object location name): * 01740000
* - ARGV[0] = (input) pointer to a char[9], null- * 01750000
* terminated string having the name of * 01760000
* this program (DSN8DUTI) * 01770000
* - ARGV[1] = (input) pointer to a char[19], null- * 01780000
* terminated string having the object name * 01790000
* to be used in conjunction with the * 01800000
* object schema as the starting point of * 01810000
* the alias resolution * 01820000
* - ARGV[2] = (input) pointer to a char[9], null- * 01830000
* terminated string having the object * 01840000
* schema to be used in conjunction with * 01850000
* the object name as the starting point of * 01860000
* the alias resolution * 01870000
* - ARGV[3] = (output) pointer to a null-terminated * 01880000
* string to receive the result as follows: * 01890000
* - char[19] for the TABLE_NAME UDF * 01900000
* - char[9] for the TABLE_SCHEMA UDF * 01910000
* - char[17] for the TABLE_LOCATION UDF * 01920000
* - ARGV[4] = (input) pointer to a short integer * 01930000
* having the null indicator for the object * 01940000
* name * 01950000
* - ARGV[5] = (input) pointer to a short integer * 01960000
* having the null indicator for the object * 01970000
* schema * 01980000
* - ARGV[6] = (output) pointer to a short integer * 01990000
* having the null indicator for the result * 02000000
* - ARGV[7] = (output) pointer to a char[6], null- * 02010000
* terminated string to receive the * 02020000
* SQLSTATE * 02030000
* - ARGV[8] = (input) pointer to a char[138], null- * 02040000
* terminated string having the UDF family * 02050000
* name of the function * 02060000
* - ARGV[9] = (input) pointer to a char[129], * 02070000
* null-terminated * 02080000
* string having the UDF specific name of * 02090000
* the function * 02100000
* - ARGV[10] = (output) pointer to a char[70], * 02110000
* null- terminated string to receive any * 02120000
* diagnostic message issued by this * 02130000
* function * 02140000
* * 02150000
* If the UDF was invoked with the object name and the * 02160000
* object schema and the object location name: * 02170000
* - ARGV[0] = (input) pointer to a char[9], null- * 02180000
* terminated string having the name of * 02190000
* this program (DSN8DUTI) * 02200000
* - ARGV[1] = (input) pointer to a char[19], null- * 02210000
* terminated string having the object name * 02220000
* to be used in conjunction with the * 02230000

```

```

* object schema and the object location * 02240000
* name as the starting point of the alias * 02250000
* resolution * 02260000
* - ARGV[2] = (input) pointer to a char[9], null- * 02270000
* terminated string having the object * 02280000
* schema to be in used in conjunction with * 02290000
* the object name and the object location * 02300000
* name as the starting point of the alias * 02310000
* resolution * 02320000
* - ARGV[3] = (input) pointer to a char[17], null- * 02330000
* terminated string having the object * 02340000
* location name to be used in conjunction * 02350000
* with the object name and the object * 02360000
* schema as the starting point of the * 02370000
* alias resolution * 02380000
* - ARGV[4] = (output) pointer to a null-terminated * 02390000
* string to receive the result as follows: * 02400000
* - char[19] for the TABLE_NAME UDF * 02410000
* - char[9] for the TABLE_SCHEMA UDF * 02420000
* - char[17] for the TABLE_LOCATION UDF * 02430000
* - ARGV[5] = (input) pointer to a short integer * 02440000
* having the null indicator for the object * 02450000
* name * 02460000
* - ARGV[6] = (input) pointer to a short integer * 02470000
* having the null indicator for the object * 02480000
* schema * 02490000
* - ARGV[7] = (input) pointer to a short integer * 02500000
* having the null indicator for the object * 02510000
* location name * 02520000
* - ARGV[8] = (output) pointer to a short integer * 02530000
* having the null indicator for the result * 02540000
* - ARGV[9] = (output) pointer to a char[6], null- * 02550000
* terminated string to receive the * 02560000
* SQLSTATE * 02570000
* - ARGV[10] = (input) pointer to a char[138], null- * 02580000
* terminated string having the UDF family * 02590000
* name of the function * 02600000
* - ARGV[11] = (input) pointer to a char[129], * 02610000
* null- terminated * 02620000
* string having the UDF specific name of * 02630000
* the function * 02640000
* - ARGV[12] = (output) pointer to a char[70], * 02650000
* null- terminated string to receive any * 02660000
* diagnostic message issued by this * 02670000
* function * 02680000
* * 02690000
* Normal Exit: Return Code: SQLSTATE = 00000
* - Message: none * 02700000
* * 02710000
* * 02720000
* Error Exit: Return Code: SQLSTATE = 38601
* - Message: DSN8DUTI Error: Invocation by unexpected * 02730000
* UDF having specific name * 02740000
* <specific name> * 02750000
* * 02760000
* Return Code: SQLSTATE = 38602 * 02770000
* - Message: DSN8DUTI Error: Unexpected SQLCODE, * 02780000
* <SQLCODE>, from SQL SELECT * 02790000
* * 02800000
* * 02810000
* External References:
* - Routines/Services: None * 02820000
* - Data areas : None * 02830000
* - Control blocks : None * 02840000
* * 02850000
* * 02860000
* Pseudocode:
* DSN8DUTI:
* - Walk down the argv list, locating the input and output parms * 02870000
* - If no object name passed, return null result * 02880000
* - If no object schema passed, assign default schema (current * 02890000
* SQLID) to object schema * 02900000
* - Concatenate wildcard ("%) to object location name * 02910000
* - SELECT TBNAME, TBCREATOR, and LOCATION from SYSIBM.SYSTABLES * 02920000
* where NAME is the object name, CREATOR is the object creator, * 02930000
* LOCATION is LIKE the object location name, and TYPE is "A" for * 02940000
* alias. * 02950000
* - if there's a result (SQLCODE = 0) then * 02960000
* - if the TABLE_NAME UDF is the invoker, assign the result * 02970000
* from TBNAME and return * 02980000
* - else if the TABLE_SCHEMA UDF is the invoker, assign the * 02990000
* result from TBCREATOR and return * 03000000
* - else if the TABLE_LOCATION UDF is the invoker, assign the * 03010000
* result from LOCATION and return * 03020000
* - else an unexpected UDF is the invoker so issue SQLSTATE * 03030000
* * 03040000
* * 03050000

```

```

* 38601 and a diagnostic message and return * 03060000
* - else if there's no result (SQLCODE = 100) then * 03070000
* - if the TABLE_NAME UDF is the invoker, assign the result * 03080000
* from the object name and return * 03090000
* - else if the TABLE_SCHEMA UDF is the invoker, assign the result * 03100000
* result from the object schema and return * 03110000
* - else if the TABLE_LOCATION UDF is the invoker, remove the * 03120000
* trailing search wildcard ("%) from and assign the result * 03130000
* from LOCATION and return * 03140000
* - else an unexpected UDF is the invoker so issue SQLSTATE * 03150000
* 38601 and a diagnostic message and return * 03160000
* - else there's an unexpected SQLCODE so issue SQLSTATE 38602 * 03170000
* and a diagnostic message and return * 03180000
* End DSN8DUTI * 03190000
*

***** C library definitions *****/
#include <stdio.h> 03200000
#include <stdlib.h> 03210000
#include <string.h> 03220000
03230000
***** Equates *****/
#define NULLCHAR '\0' /* Null character */ 03300000
03310000
#define MATCH 0 /* Comparison status: Equal */ 03320000
03330000
#define NOT_OK 0 /* Run status indicator: Error */ 03340000
03350000
#define OK 1 /* Run status indicator: Good */ 03360000
03370000
03380000
03390000
***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA; 03400000
03410000
03420000
03430000
03440000
***** DB2 Host Variables *****/
EXEC SQL BEGIN DECLARE SECTION; 03450000
03460000
char hvObjName[19]; /* host var for object name */ 03470000
03480000
short int *nObjName; /* indic var for hvObjName */ 03490000
03500000
char hvObjSchema[9]; /* host var for obj schema */ 03510000
03520000
short int *nObjSchema; /* indic var for hvObjSchema */ 03530000
03540000
char hvObjLocation[18]; /* host var for obj location */ 03550000
03560000
short int *nObjLocation; /* indic var for hvObjLocation*/ 03570000
03580000
char hvLOCATION[17]; /* host var for LOCATION col */ 03590000
03600000
char hVTBCREATOR[9]; /* host var for TBCREATOR col */ 03610000
03620000
char hVTBNAME[19]; /* host var for TBNAME column */ 03630000
03640000
03650000
03660000
03675990
03681980
03690000
03700000
03710000
03720000
03730000
03740000
03750000
03760000
03763000
03766000
03770000
03780000
03790000
03800000
03810000
03820000
03830000
03840000
03850000
03860000
03870000
03880000
***** local variables *****/
int main(int argc, char *argv[]) 03890000
{
 **** local variables *****/
 short int minus1 = -1; /* default null indic setting */ 03900000
 03910000
 char *result; /* result of this function */ 03920000
 short int *nResult; /* indic var, result */ 03930000
 char *sqlstate; /* SQLSTATE */ 03940000
 char fnName[138]; /* function name */ 03950000
 char specificName[129]; /* specific name of function */ 03960000
 char *message; /* diagnostic message */ 03970000
 short int status = OK; /* DSN8DUTI run status */ 03980000
 03990000

 * Walk down the argv list, locating the input and output parms * 04000000

 argc--; /* convert argc to base 0 index */ 04010000
 04020000
 message = (char *)argv[argc--]; /* out: point to UDF diag msg */ 04030000
 04040000
 strcpy(specificName, /* in: save UDF specific name */ 04050000
 argv[argc--]); 04060000
 04070000
 strcpy(fnName,argv[argc--]); /* in: save UDF function name */ 04080000
 04090000
 sqlstate = (char *)argv[argc--]; /* out: point to UDF sqlstate */ 04100000
 04110000
 nResult = (short int *)argv[argc--]; /* out: point to null indicator*/ 04120000
 04130000
 /* variable for result */ 04140000
 04150000

```

```

if(argc == 7) /* if 3 input parms passed */ 03890000
 niObjLocation /* ..in: point to null indic. */ 03900000
 = (short int *)argv[argc--]; /* var for object loc'n */ 03910000
else /* otherwise it wasn't passed */ 03920000
 niObjLocation = &minus1; /* ..so define it as null */ 03930000
 03940000
if(argc >= 5) /* if 2 or 3 input parms passed*/ 03950000
 niObjSchema /* ..in: point to null indic. */ 03960000
 = (short int *)argv[argc--]; /* var for object schema */ 03970000
else /* otherwise it wasn't passed */ 03980000
 niObjSchema = &minus1; /* ..so define it as null */ 03990000
 04000000
niObjName /* in: point to null indicator */ 04010000
 = (short int *)argv[argc--]; /* var for object name */ 04020000
 04030000
result = (char *)argv[argc--]; /* out: point to UDF result */ 04040000
 04050000
if(argc == 3) /* if 3 input parms passed */ 04060000
 strcpy(hvObjLocation, /* ..in: save object location */ 04070000
 argv[argc--]); /* name */ 04080000
else /* otherwise it wasn't passed */ 04090000
 hvObjLocation[0] = NULLCHAR;
 /* ..so define it as null */ 04100000
 04110000
if(argc >= 2) /* if 2 or 3 input parms passed*/ 04120000
 strcpy(hvObjSchema, /* ..in: save object schema */ 04130000
 argv[argc--]); /* */ 04140000
else /* otherwise it wasn't passed */ 04150000
 hvObjSchema[0] = NULLCHAR;
 /* ..so define it as null */ 04160000
 04170000
strcpy(hvObjName,argv[argc]); /* in: save object name */ 04180000
 04190000
 04200000
/*** 04210000
* Initialize output parms * 04220000
***/ 04230000
message[0] = NULLCHAR; 04240000
strcpy(sqlstate,"00000"); 04250000
*niResult = 0; 04260000
result[0] = NULLCHAR; 04270000
 04280000
/*** 04290000
* If no object name provided, return null result * 04300000
***/ 04310000
if((*niObjName != 0) || (strlen(hvObjName) == 0)) 04320000
 status = NOT_OK;
 04330000
/*** 04340000
* If no object schema provided, assign default schema * 04350000
***/ 04360000
if((*niObjSchema != 0) || (strlen(hvObjSchema) == 0)) 04370000
 EXEC SQL SET :hvObjSchema = CURRENT SQLID;
 04380000
/*** 04390000
* Concatenate "wildcard" to object location * 04400000
***/ 04410000
strcat(hvObjLocation,"%");
 04420000
 04430000
/*** 04440000
* Look for alias with the object name (and schema (and location)) * 04450000
***/ 04460000
if(status == OK)
{
 EXEC SQL SELECT TBNAME,
 TBCREATOR,
 LOCATION
 INTO :hvTBNAME,
 :hvTBCREATOR,
 :hvLOCATION
 FROM SYSIBM.SYSTABLES
 WHERE NAME = :hvObjName
 AND CREATOR = :hvObjSchema
 AND LOCATION LIKE :hvObjLocation
 AND TYPE = 'A';
 04490000
 04500000
 04510000
 04520000
 04530000
 04540000
 04550000
 04560000
 04570000
 04580000
 04590000
 04600000
 04610000
 04620000
 * 04630000
 04640000
if(SQLCODE == 0)
 /* If such an alias was found ... */ 04650000
 if(strncmp(specificName,"DSN8DUTIN",9) == 0)
 /* TABLE_NAME UDF: return true name of table or view */ 04660000
 * TABLE_NAME UDF: return true name of table or view */ 04670000
 strcpy(result,hvTBNAME);
 04680000
 04690000
 else if(strncmp(specificName,"DSN8DUTIS",9) == 0) 04700000

```

```

/********************* 04710000
 * TABLE_SCHEMA UDF: return true schema of table or view * 04720000
***** 04730000
strcpy(result,hvTBCREATOR);
04740000
else if(strncmp(specificName,"DSN8DUTIL",9) == 0) 04750000
***** 04760000
* TABLE_LOCATION UDF: return true loc'n of table or view * 04770000
***** 04780000
strcpy(result,hvLOCATION);
04790000
else 04800000
***** 04810000
* Unknown UDF: signal error * 04820000
***** 04830000
{
 status = NOT_OK; 04850000
 strcpy(sqlstate,"38601");
04860000
 sprintf(message,
 "DSN8DUTI Error: Invocation by unexpected UDF ",
04880000
 "having specific name %s",
04890000
 specificName);
04900000
}

else if(SQLCODE == 100) 04930000
***** 04940000
* If no such alias was found ... * 04950000
***** 04960000
if(strncmp(specificName,"DSN8DUTIN",9) == 0) 04970000
***** 04980000
* TABLE_NAME UDF: return starting point * 04990000
***** 05000000
strcpy(result,hvObjName);
05010000
else if(strncmp(specificName,"DSN8DUTIS",9) == 0) 05020000
***** 05030000
* TABLE_SCHEMA UDF: return schema of starting point * 05040000
***** 05050000
strcpy(result,hvObjSchema);
05060000
else if(strncmp(specificName,"DSN8DUTIL",9) == 0) 05070000
***** 05080000
* TABLE_LOCATION UDF: Remove trailing search wildcard byte * 05090000
* and return location of starting point * 05100000
***** 05110000
{
 hvObjLocation[strlen(hvObjLocation)-1] = NULLCHAR;
05120000
 strcpy(result,hvObjLocation);
05130000
}
else 05150000
***** 05160000
* Unknown UDF: signal error * 05180000
***** 05190000
{
 status = NOT_OK;
 strcpy(sqlstate,"38601");
05210000
 sprintf(message,
 "DSN8DUTI Error: Invocation by unexpected UDF ",
05240000
 "having specific name %s",
05250000
 specificName);
05260000
}

else 05290000
***** 05300000
* If unexpected SQLCODE, issue message * 05310000
***** 05320000
{
 status = NOT_OK;
 strcpy(sqlstate,"38602");
05350000
 sprintf(message,
 "DSN8DUTI Error: Unexpected SQLCODE, %d,
05370000
 "from SQL SELECT",
05380000
 SQLCODE);
05390000
}
} /* end if(status == OK) */

***** 05430000
* If null starting point or unexpected SQLCODE, return null result * 05440000
***** 05450000
if(status == NOT_OK)
{
 result[0] = NULLCHAR;
05480000
 *niResult = -1;
05490000
}
else
{

```

```

 *niResult = 0;
 strcpy(sqlstate,"00000");
 }

} /* end DSN8DUTI */

```

	05530000
	05540000
	05560000
	05570000
	05620000

### Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8DUWC

Invoca la función de tabla UDF de ejemplo WEATHER para mostrar cómo se manejan una UDF y una tabla UDF utilizando SQL estático.

```

* Module name = DSN8DUWC (DB2 sample program)
*
* DESCRIPTIVE NAME = Client for sample UDF table function WEATHER
*
*
* LICENSED MATERIALS - PROPERTY OF IBM
* 5645-DB2
* (C) COPYRIGHT 1998 IBM CORP. ALL RIGHTS RESERVED.
*
* STATUS = VERSION 6
*
* Function: Invokes the sample UDF table function WEATHER to demon-
* strate how a UDF and UDF table handling using static SQL.
*
* Notes:
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher
*
* Restrictions:
*
* Module type: C program
* Processor: IBM C/C++ for OS/390 V1R3 or higher
* Module size: See linkedit output
* Attributes: Re-entrant and re-usable
*
* Entry Point: DSN8DUWC
* Purpose: See Function
* Linkage: DB2SQL
* Invoked via SQL UDF call
*
*
* Parameters: DSN8DUWC uses the C "main" argument convention of
* argv (argument vector) and argc (argument count).
*
* - ARGV[0] = (input) pointer to a char[9], null-
* terminated string having the name of
* this program (DSN8DUWC)
* - ARGV[1] = (input) pointer to a char[45], null-
* terminated string having the name of the
* source data for the weather reports.
*
* Normal Exit: Return Code: 0000
* - Message: none
*
* Error Exit: Return Code: 0008
* - Message: DSN8DUWC failed: Invalid parameter count
*
* - Message: <formatted SQL text from DSNTIAR>
*
*
* External References:
* - Routines/Services: DSNTIAR: DB2 msg text formatter
* - Data areas : None
* - Control blocks : None
*
* Pseudocode:
* DSN8DUWC:
* - Verify that 2 input parameters (program name and weather data
* set name) were passed.
* - if not, issue diagnostic message and end with code 0008
* - Open WEATHER_CURSOR, the client cursor for the WEATHER UDF
* table function, passing the weather data set name as a host
* variable
*
```

```

* - if unsuccessful, call sql_error to issue a diagnostic mes- *
* sage, then end with code 0008. *
* - Do while not end of cursor *
* - Read the cursor *
* - If successful, print the result *
* - else if not end of cursor, call sql_error to issue a diag- *
* nistic message, then end with code 0008. *
* - Close the cursor *
* - if unsuccessful, call sql_error to issue a diagnostic mes- *
* sage, then end with code 0008. *
* End DSN8DUWC *
* * *
* sql_error: *
* - call DSNTIAR to format the unexpected SQLCODE. *
* End sql_error *
* * *
***** C library definitions *****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/***** Equates *****/
#define NULLCHAR '\0' /* Null character */
#define OUTLEN 80 /* Length of output line */
#define DATA_DIM 10 /* Number of message lines */
#define NOT_OK 0 /* Run status indicator: Error */
#define OK 1 /* Run status indicator: Good */

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

/***** DB2 Host Variables *****/
EXEC SQL BEGIN DECLARE SECTION;
char hvWeatherDSN[44]; /* host var for weather dsn */
short int niWeatherDSN = 0; /* indic var for weather dsn */
char hvCity[31]; /* host var for name of city */
short int niCity = 0; /* indic var for city name */
long int hvTemp_in_f = 0; /* host var, fahrenheit temp */
short int niTemp_in_f = 0; /* indic var for temperature */
long int hvHumidity = 0; /* host var, percent humidity */
short int niHumidity = 0; /* indic var for humidity */
char hvWind[5]; /* host var, wind direction */
short int niWind = 0; /* indic var for wind direct */
long int hvWind_velocity = 0; /* host var, wind velocity */
short int niWind_velocity = 0; /* indic var for wind velocity */
double hvBarometer = 0; /* host var, barometric press */
short int niBarometer = 0; /* indic var for baro pressure */
char hvForecast[26]; /* host var, forecast */
short int niForecast = 0; /* indic var for forecast */

EXEC SQL END DECLARE SECTION;

/***** DB2 SQL Cursor Declarations *****/
EXEC SQL BEGIN DECLARE SECTION;

EXEC SQL DECLARE WEATHER_CURSOR
CURSOR FOR
SELECT CITY, 00008900
 TEMP_IN_F, 00008900
 HUMIDITY, 00008900
 WIND, 00008900
 WIND_VELOCITY, 00008900
 BAROMETER, 00008900
 FORECAST
FROM TABLE(DSN8.WEATHER(:hvWeatherDSN)) AS W
WHERE CITY = 'Juneau, AK';

EXEC SQL END DECLARE SECTION;

```

```

/***** DB2 Message Formatter *****/
struct error_struct /* DSNTIAR message structure */
{
 short int error_len;
 char error_text[DATA_DIM][OUTLEN];
} error_message = {DATA_DIM * (OUTLEN)};

#pragma linkage(dsntiar, OS)

extern short int dsntiar(struct sqlca *sqlca,
 struct error_struct *msg,
 int *len);

/***** DSN8DUWC Global Variables *****/
short int status = OK; /* DSN8DUWC run status */

long int completion_code = 0; /* DSN8DUWC return code */

/***** DSN8DUWC Function Prototypes *****/
int main(int argc, char *argv[]);
void sql_error(char locmsg[]);

int main(int argc, char *argv[])
/*****
*
***** ****/
{
 if(argc == 2)
 strcpy(hvWeatherDSN, argv[1]);
 else
 {
 printf("DSN8DUWC failed: Invalid parameter count\n");
 status = NOT_OK;
 }

 if(status == OK)
 {
 EXEC SQL OPEN WEATHER_CURSOR;
 if(SQLCODE != 0)
 sql_error(" *** Open weather cursor");
 }

 while(SQLCODE == 0 && status == OK)
 {
 EXEC SQL FETCH WEATHER_CURSOR
 INTO :hvCity :niCity,
 :hvTemp_in_f :niTemp_in_f,
 :hvHumidity :niHumidity,
 :hvWind :niWind,
 :hvWind_velocity:niWind_velocity,
 :hvBarometer :niBarometer,
 :hvForecast :niForecast;

 if(SQLCODE == 0)
 {
 printf("Weather Report for %s\n", hvCity);
 printf("... Temperature : %d\n", hvTemp_in_f);
 printf("... Humidity : %d\n", hvHumidity);
 printf("... Wind direction: %s\n", hvWind);
 printf("... Wind velocity : %d\n", hvWind_velocity);
 printf("... Barometer : %.2f\n", hvBarometer);
 printf("... Forecast : %s\n", hvForecast);
 }
 else if(SQLCODE != 100)
 sql_error(" *** Fetch from weather cursor");
 }

 if(status == OK)
 {
 EXEC SQL CLOSE WEATHER_CURSOR;
 if(SQLCODE != 0)
 sql_error(" *** Close weather cursor");
 }

 if(status != OK)
 completion_code = 8;
}

return(completion_code);

```

```

} /* end main */

/*****
** SQL error handler
*****/
void sql_error(char locmsg[]) /*proc*/
{

 short int rc; /* DSNTIAR Return code */
 int j,k; /* Loop control */
 static int lrecl = OUTLEN; /* Width of message lines */
 /* */

/* set status to prevent further processing */
status = NOT_OK;

/* print the locator message */
printf(" %.80s\n", locmsg);

/* format and print the SQL message */
rc = dsntiar(&sqlca, &error_message, &lrecl);
if(rc == 0)
 for(j=0; j<DATA_DIM; j++)
 {
 for(k=0; k<OUTLEN; k++)
 putchar(error_message.error_text[j][k]);
 putchar('\n');
 }
else
{
 printf(" *** ERROR: DSNTIAR could not format the message\n");
 printf(" *** SQLCODE is %d\n",SQLCODE);
 printf(" *** SQLERRM is \n");
 for(j=0; j<sqlca.sqlerrml; j++)
 printf("%c", sqlca.sqlerrmc[j]);
 printf("\n");
}
}

} /* end of sql_error */

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

### DSN8DUWF

Devuelve información metereológica para varias ciudades, como se lee desde el conjunto de datos pasado como argumento al parámetro de entrada 'weatherDSN'.

```

/*****
* Module name = DSN8DUWF (DB2 sample program) * 00000100
* DESCRIPTIVE NAME = Weather (DB2 user-defined table function) * 00000200
* STATUS = VERSION 7 * 00000300
* Function: Returns weather information for various cities, as read * 00000400
* from the data set passed as the argument to input para- * 00000500
* meter 'weatherDSN'. The data includes the name of a * 00000600
* city followed by its weather information: Temperature in * 00000700
* fahrenheit, percent humidity, wind direction, wind velo- * 00000800
* city, barometric pressure, and the forecast. See the * 00000900
* structure 'weatherRec' for the record format. * 00001000
* File pointer information is retained between calls in * 00001100
* * 00001200
* * 00001300
* * 00001400
* * 00001500
* * 00001600
* * 00001700
* * 00001800
* * 00001900

```

```

* the UDF scratchpad area. * 00002000
*
* Data read from the input data set is returned by this * 00002100
* function as a DB2 table with the following structure: * 00002200
* with this structure: * 00002300
*
* CITY VARCHAR(30), * 00002400
* TEMP_IN_F INTEGER, * 00002500
* HUMIDITY INTEGER, * 00002600
* WIND VARCHAR(5), * 00002700
* WIND_VELOCITY INTEGER, * 00002800
* BAROMETER FLOAT, * 00002900
* FORECAST VARCHAR(25) * 00003000
*
* Clients invoking this function can use standard SQL * 00003100
* syntax to create a desired result set. * 00003200
*
* Example invocation: * 00003300
*
* EXEC SQL DECLARE WEATHER_CURSOR
* CURSOR FOR
* SELECT CITY,
* FORECAST
* FROM TABLE(DSN8.WEATHER(:hvWeatherDSN)) * 00003400
* AS W
* WHERE CITY = 'Juneau, AK'; * 00003500
*
* Notes: * 00003600
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher * 00003700
*
* Restrictions: * 00003800
*
* Module type: C program * 00003900
* Processor: IBM C/C++ for OS/390 V1R3 or higher * 00004000
* Module size: See linkedit output * 00004100
* Attributes: Re-entrant and re-usable * 00004200
*
* Entry Point: DSN8DUWF * 00004300
* Purpose: See Function * 00004400
* Linkage: DB2SQL * 00004500
* Invoked via SQL UDF call * 00004600
*
* Input: Parameters explicitly passed to this function: * 00004700
* - *weatherDSN : pointer to a char[45], null-termi- * 00004800
* nated string having the name of the * 00004900
* source data for the weather reports. * 00005000
* - *niWeatherDSN: pointer to a short integer having * 00005100
* the null indicator variable for * 00005200
* *weatherDSN. * 00005300
* - *fnName : pointer to a char[138], null-termi- * 00005400
* nated string having the UDF family * 00005500
* name of this function. * 00005600
* - *specificName: pointer to a char[129], null-termi- * 00005700
* nated string having the UDF specific * 00005800
* name of this function. * 00005900
* * 00006000
* * 00006100
* * 00006200
* * 00006300
* * 00006400
* * 00006500
* * 00006600
* * 00006700
* * 00006800
* * 00006900
* * 00007000
* * 00007100
* * 00007200
* * 00007300
* * 00007400
* * 00007500
* * 00007600
* * 00007700
* * 00007800
* * 00007900
* * 00008000
* * 00008100
* * 00008200
* * 00008300
* * 00008400
* * 00008500
* * 00008600
* * 00008700
* * 00008800
* * 00008900
* * 00009000
* * 00009100
* * 00009200
* * 00009300
* * 00009400
* * 00009500
* * 00009600
* * 00009700
* * 00009800
* * 00009900
* * 00010000
* * 00010100
*
* Output: Parameters explicitly passed by this function: * 00009000
* - *city : pointer to a char[31], null-termi- * 00009100
* nated string to receive the name of * 00009200
* the city. * 00009300
* - *temp_in_f : pointer to a long integer to receive * 00009400
* the temperature in fahrenheit for * 00009500
* the city. * 00009600
* - *humidity : pointer to a long integer to receive * 00009700
* the percent humidity for the city. * 00009800
* - *wind : pointer to a char[6], null-termi- * 00009900
* nated string to receive the wind di- * 00010000
* rection for the city. * 00010100
* * 00009000
* * 00009100
* * 00009200
* * 00009300
* * 00009400
* * 00009500
* * 00009600
* * 00009700
* * 00009800
* * 00009900
* * 00010000
* * 00010100

```

```

*
* - *niHumidity : pointer to a short integer to re- * 00010200
* - *niWind : pointer to a short integer to re- * 00010300
* - *niWind_velocity: pointer to a short integer to re- * 00010400
* - *niBarometer : pointer to a short integer to re- * 00010500
* - *niForecast : pointer to a short integer to re- * 00010600
* - *sqlstate : pointer to a char[06], null-termi- * 00010700
* - *message : pointer to a char[70], null-termi- * 00010800
* nated string to receive the SQLSTATE.* 00010900
* nated string to receive a diagnostic * 00011000
* message if one is generated by this * 00011100
* function. * 00011200
* function. * 00011300
* function. * 00011400
* function. * 00011500
* function. * 00011600
* function. * 00011700
* function. * 00011800
* function. * 00011900
* function. * 00012000
* function. * 00012100
* function. * 00012200
* function. * 00012300
* function. * 00012400
* Normal Exit: Return Code: SQLSTATE = 00000 * 00012500
* - Message: none * 00012600
*
* Return Code: SQLSTATE = 02000 (end of input) * 00012700
* - Message: none * 00012800
* - Message: none * 00012900
* - Message: none * 00013000
* Error Exit: Return Code: SQLSTATE = 38601 * 00013100
* - Message: DSN8DUWF Error: Unable to allocate DD * 00013200
* <ddname>: Error code=<x>, * 00013300
* info code=<y> * 00013400
* * 00013500
* Return Code: SQLSTATE = 38602 * 00013600
* - Message: DSN8DUWF Error: Error opening weather data * 00013700
* set * 00013800
* * 00013900
* Return Code: SQLSTATE = 38603 * 00014000
* - Message: DSN8DUWF Error: Error reading weather data * 00014100
* set * 00014200
* * 00014300
* Return Code: SQLSTATE = 38604 * 00014400
* - Message: DSN8DUWF Error: Error closing weather data * 00014500
* set * 00014600
* * 00014700
* Return Code: SQLSTATE = 38605 * 00014800
* - Message: DSN8DUWF Error: FREE failed for DDNAME <x>.* 00014900
* Error code=<y>, info code= * 00015000
* <z> * 00015100
* * 00015200
*
* External References:
* - Routines/Services: dyninit: IBM C/C++, dynith.h * 00015300
* - Initializes control block for * 00015400
* dynamic file allocation * 00015500
* * 00015600
* dynalloc: IBM C/C++, dynith.h * 00015700
* - Dynamic file allocation * 00015800
* dynfree: IBM C/C++, dynith.h * 00015900
* - Dynamic file deallocation * 00016000
* - Data areas : None * 00016100
* - Control blocks : __dyn_t: IBM C/C++, dynint.h * 00016200
* - for dynamic file allocation * 00016300
* * 00016400
*
* Pseudocode:
* DSN8DUWF:
* - If SQLUDF call type is SQLUDF_TF_FIRST (-2)
* - call allocWeatherDSN to allocate the data set name passed as * 00016600
* the argument to the weatherDSN parameter * 00016700
* - Else if SQLUDF call type is SQLUDF_TF_OPEN (-1)
* - call openWeatherDS to open the weather data set * 00016800
* - Else if SQLUDF call type is SQLUDF_TF_FETCH (0)
* - call readWeatherDS to read the next record from the weather * 00016900
* data set
* - if EOF, set sqlstate to 02000 to signal end of cursor to * 00017000
* client
* - else call buildReturnRow to populate the UDF table function * 00017100
* output parameters with data from the input record * 00017200
* - Else if SQLUDF call type is SQLUDF_TF_CLOSE (1)
* - call closeWeatherDS to close the weather data set * 00017300
* - Else (SQLUDF call type is SQLUDF_TF_FINAL (2))
* - call freeWeatherDS to deallocate the weather data set * 00017400
* End DSN8DUWF

```

```

* * 00018400
* allocWeatherDS: * 00018500
* - if the data set name passed in as the argument to the input * 00018600
* parameter weatherDSN is for a partitioned data set, extract * 00018700
* the member name * 00018800
* - use dynalloc to dynamically allocate the data set (and member, * 00018900
* if applicable) * 00019000
* - if allocation error occurs, issue sqlstate 38601 and a diag- * 00019100
* nostic message * 00019200
* End allocWeatherDS * 00019300
* * 00019400
* openWeatherDS: * 00019500
* - open the weather data set * 00019600
* - if the data cannot be opened, issue sqlstate 38602 and a diag- * 00019700
* nostic message * 00019800
* End openWeatherDS * 00019900
* * 00020000
* readWeatherDS: * 00020100
* - read the next record from the data set * 00020200
* - this implicitly updates the file pointer variable, which is * 00020300
* maintained in the UDF scratchpad area * 00020400
* - if the data set cannot be read, issue sqlstate 38603 and a * 00020500
* diagnostic message * 00020600
* End readWeatherDS * 00020700
* * 00020800
* buildReturnRow: * 00020900
* - extract weather data fields from the weather data set * 00021000
* - perform appropriate data type conversions * 00021100
* - copy the (converted) data to the appropriate output parameters * 00021200
* End buildReturnRow * 00021300
* * 00021400
* closeWeatherDS: * 00021500
* - close the weather data set * 00021600
* - if the data cannot be closed, issue sqlstate 38604 and a diag- * 00021700
* nostic message * 00021800
* End closeWeatherDS * 00021900
* * 00022000
* freeWeatherDS: * 00022100
* - use dynfree to dynamically deallocate the weather data set * 00022200
* - if deallocation error occurs, issue sqlstate 38605 and a diag- * 00022300
* nostic message * 00022400
* End freeWeatherDS * 00022500
* * 00022600
******/ 00022700
00022800
00022900
00023000
#pragma linkage(DSN8DUWF,fetchable) 00023100
***** C library definitions *****/
#include <dynit.h> 00023200
#include <stdlib.h> 00023300
#include <string.h> 00023400
#include <stdio.h> 00023500
00023600
***** Equates *****/
#define NO 0 /* Negative */ 00023800
#define YES 1 /* Affirmative */ 00023900
00024000
#define NULLCHAR '\0' /* Null character */ 00024100
00024200
#define SQLUDF_TF_FIRST -2 /* First call */ 00024300
#define SQLUDF_TF_OPEN -1 /* Open table call */ 00024400
#define SQLUDF_TF_FETCH 0 /* Fetch next row call */ 00024500
#define SQLUDF_TF_CLOSE 1 /* Close table call */ 00024600
#define SQLUDF_TF_FINAL 2 /* Final call */ 00024700
00024800
***** Weather Data Set *****/
struct scr /* Struct for scratchpad area */ 00025000
{
 long len; /* Length of scratchpad data */ 00025200
 FILE *WEATHRin; /* ptr to weather data set */ 00025300
 char not_used[100]; /* filler */ 00025400
}; 00025500
00025600
char WEATHRinBuffer[8188]; /* Input buffer for weather ds*/ 00025700
short int moreWeatherRecs = YES; /* EOF indicator, weather ds */ 00025800
00025900
typedef struct /* Weather record structure */ 00026000
{ 00026100
 char cityField[30]; /* name of city 01-30 */ 00026200
 char filler1[1]; /* 31-31 */ 00026300
 char temp_in_fField[3]; /* temp in fahrenheit 32-34 */ 00026400
 char filler2[1]; /* 35-35 */ 00026500
}

```

```

char humidityField[3]; /* percent humidity 36-38 */ 00026600
char filler3[1]; /* 39-39 */ 00026700
char windField[5]; /* wind direction 40-44 */ 00026800
char filler4[1]; /* 45-45 */ 00026900
char windVelocityField[3]; /* wind velocity 46-48 */ 00027000
char filler5[1]; /* 49-49 */ 00027100
char barometerField[5]; /* barometric pressure 50-54 */ 00027200
char filler6[1]; /* 55-55 */ 00027300
char forecastField[25]; /* forecast 56-80 */ 00027400
} weatherRec; 00027500
 00027600
weatherRec *pweatherRec = (weatherRec *)&WEATHRinBuffer; 00027700
 00027800
 00027900
***** Functions *****/
void *DSN8DUWF (*weatherDSN, /* Weather function */ 00028100
(char *city, /* in: input ds, weather data */ 00028200
 long int *temp_in_f, /* out: name of city */ 00028300
 long int *humidity, /* out: temp in fahrenheit */ 00028400
 char *wind, /* out: relative humidity */ 00028500
 long int *wind_velocity, /* out: wind direction */ 00028600
 double *barometer, /* out: wind velocity */ 00028700
 char *forecast, /* out: barometric pressure */ 00028800
 short int *niWeatherDSN, /* out: forecast */ 00028900
 short int *niCity, /* in: indic var, weather dsn */ 00029000
 short int *niTemp_in_f, /* out: indic var, city name */ 00029100
 short int *niHumidity, /* out: indic var, temperature*/ 00029200
 short int *niWind, /* out: indic var, humidity */ 00029300
 short int *niWind_velocity, /* out: indic var, wind dir */ 00029400
 short int *niBarometer, /* out: indic var, wind veloc */ 00029500
 short int *niForecast, /* out: indic var, baro press */ 00029600
 char *sqlstate, /* out: indic var, forecast */ 00029700
 char *fnName, /* in: family name of func */ 00029800
 char *specificName, /* in: specific name of func */ 00029900
 char *msgtext, /* out: diagnostic message */ 00030000
 struct scr *scratchptr, /* i/o: scratchpad area */ 00030200
 long *callType, /* i/o: call type parameter */ 00030300
);
 00030400
 00030500
void allocWeatherDS (*Dynam allocates weather ds */ 00030600
(char *weatherDSN, /* in: name of weather ds */ 00030700
 char *sqlstate, /* out: sqlstate */ 00030800
 char *msgtext, /* out: diag message text */ 00030900
);
 00031000
 00031100
void openWeatherDS (* Opens weather data set */ 00031200
(struct scr *scratchptr, /* in: ptr to scratch pad */ 00031300
 char *sqlstate, /* out: sqlstate */ 00031400
 char *msgtext, /* out: diag message text */ 00031500
);
 00031600
 00031700
void readWeatherDS (* Reads from weather data set*/ 00031800
(struct scr *scratchptr, /* in: ptr to scratch pad */ 00031900
 char *sqlstate, /* out: sqlstate */ 00032000
 char *msgtext, /* out: diag message text */ 00032100
);
 00032200
 00032300
void buildReturnRow (* Builds function return row */ 00032400
(char *city, /* out: name of city */ 00032500
 long int *temp_in_f, /* out: temp in fahrenheit */ 00032600
 long int *humidity, /* out: relative humidity */ 00032700
 char *wind, /* out: wind direction */ 00032800
 long int *wind_velocity, /* out: wind velocity */ 00032900
 double *barometer, /* out: barometric pressure */ 00033000
 char *forecast, /* out: forecast */ 00033100
 short int *niCity, /* out: indic var, city name */ 00033200
 short int *niTemp_in_f, /* out: indic var, temperature*/ 00033300
 short int *niHumidity, /* out: indic var, humidity */ 00033400
 short int *niWind, /* out: indic var, wind dir */ 00033500
 short int *niWind_velocity, /* out: indic var, wind veloc */ 00033600
 short int *niBarometer, /* out: indic var, baro press */ 00033700
 short int *niForecast, /* out: indic var, forecast */ 00033800
);
 00033900
 00034000
void closeWeatherDS (* Closes weather data set */ 00034100
(struct scr *scratchptr, /* in: ptr to scratch pad */ 00034200
 char *sqlstate, /* out: sqlstate */ 00034300
 char *msgtext, /* out: diag message text */ 00034400
);
 00034500
 00034600
void freeWeatherDS (* Dynam frees the weather ds */ 00034700

```

```

(char *sqlstate, /* out: sqlstate */ 00034800
char *msgtext, /* out: diag message text */ 00034900
);

void *DSN8DUWF
(char *weatherDSN, /* in: input ds, weather data */ 00035400
char *city, /* out: name of city */ 00035500
long int *temp_in_f, /* out: temp in fahrenheit */ 00035600
long int *humidity, /* out: relative humidity */ 00035700
char *wind, /* out: wind direction */ 00035800
long int *wind_velocity, /* out: wind velocity */ 00035900
double *barometer, /* out: barometric pressure */ 00036000
char *forecast, /* out: forecast */ 00036100
short int *niWeatherDSN, /* in: indic var, weather dsn */ 00036200
short int *niCity, /* out: indic var, city name */ 00036300
short int *niTemp_in_f, /* out: indic var, temperature*/ 00036400
short int *niHumidity, /* out: indic var, humidity */ 00036500
short int *niWind, /* out: indic var, wind dir */ 00036600
short int *niWind_velocity, /* out: indic var, wind veloc */ 00036700
short int *niBarometer, /* out: indic var, baro press */ 00036800
short int *niForecast, /* out: indic var, forecast */ 00036900
char *sqlstate, /* out: SQLSTATE */ 00037000
char *fnName, /* in: family name of function*/ 00037100
char *specificName, /* in: specific name of func */ 00037200
char *msgtext, /* out: diagnostic message */ 00037300
struct scr *scratchptr, /* i/o: scratchpad area */ 00037400
long *callType, /* i/o: call type parameter */ 00037500
 00037600
) **** 00037700
* Main routine for Weather table function * 00037800
***** 00037900
{
 **** 00038000
 **** 00038100
 * First call: Dynamically allocate the weather data set * 00038200
***** 00038300
if(*callType == SQLUDF_TF_FIRST)
{
 {
 strcpy(sqlstate,"00000"); /* Init sqlstate return var */ 00038600
 msgtext = NULLCHAR; / Init message text rtrn var */ 00038700
 allocWeatherDS(weatherDSN, sqlstate, msgtext);
 00038800
 }
 **** 00038900
 **** 00039000
 * Second call: Open the weather data set * 00039100
***** 00039200
else if(*callType == SQLUDF_TF_OPEN)
{
 {
 strcpy(sqlstate,"00000"); /* Init sqlstate return var */ 00039500
 msgtext = NULLCHAR; / Init message text rtrn var */ 00039600
 moreWeatherRecs = YES; /* EOF indicator, weather ds */ 00039700
 openWeatherDS(scratchptr, sqlstate, msgtext);
 00039800
 }
 **** 00039900
 **** 00040000
 * Subsequent calls: Read a record from the weather data set * 00040100
***** 00040200
else if(*callType == SQLUDF_TF_FETCH)
{
 {
 readWeatherDS(scratchptr, sqlstate, msgtext);
 00040500
 if(moreWeatherRecs == NO) /* If no more weather data */ 00040600
 strcpy(sqlstate,"02000"); /* ..signal for FINAL CALL */ 00040700
 else
 {
 buildReturnRow(city,
 temp_in_f,
 humidity,
 wind,
 wind_velocity,
 barometer,
 forecast,
 niCity,
 niTemp_in_f,
 niHumidity,
 niWind,
 niWind_velocity,
 niBarometer,
 niForecast);
 00041000
 }
 }
 **** 00042500
 **** 00042600
 * End of file: Close weather data set * 00042700
***** 00042800
else if(*callType == SQLUDF_TF_CLOSE)
 00042900

```

```

{
 closeWeatherDS(scratchptr, sqlstate, msgtext);
}
/*** 00043000
* Final call: De-allocate weather data set * 00043100
***** 00043200
else /* *callType == SQLUDF_TF_FINAL */
{
{
 freeWeatherDS(sqlstate, msgtext);
}
return;
}

void allocWeatherDS
(char *weatherDSN, /* in: name of weather ds */ 00044000
char *sqlstate, /* out: sqlstate */ 00044100
char *msgtext, /* out: diag message text */ 00044200
)
/*** 00044300
* Dynamically allocates weatherDSN to the WEATHRIN DD. If the value * 00044400
* in weatherDSN contains parentheses, it is assumed to specify a * 00044500
* partitioned data set; otherwise it is assumed to specify a * 00044600
* physical sequential data set. * 00044700
***** 00044800
{
__dyn_t ip; /* pointer to control block */ 00044900
char DSname[45]; /* recv's copy of weather dsn */ 00045000
char *tokPtr; /* string ptr for token parser*/ 00045100
00045200
00045300
00045400
00045500
00045600
00045700
00045800
00045900
00046000
00046100
00046200
00046300
00046400
00046500
00046600
00046700
00046800
00046900
00047000
00047100
00047200
00047300
00047400
00047500
00047600
00047700
00047800
00047900
00048000
00048100
00048200
00048300
00048400
00048500
00048600
00048700
00048800
00048900
00049000
00049100
00049200
00049300
00049400
00049500
00049600
00049700
00049800
00049900
00050000
00050100
00050200
00050300
00050400
00050500
00050600
00050700
00050800
00050900
00051000
00051100
}

void openWeatherDS
(struct scr *scratchptr, /* in: ptr to scratch pad */ 00049400
char *sqlstate, /* out: sqlstate */ 00049500
char *msgtext, /* out: diag message text */ 00049600
)
/*** 00049700
* Opens the weather data set, which has been allocated to the DD * 00049800
* WEATHRIN, for record-type input, and assigns the file pointer to * 00049900
* the scratchpad area indicated by scratchptr. * 00050000
***** 00050100
00050200
00050300
00050400
00050500
00050600
00050700
00050800
00050900
00051000
00051100
}

if(scratchptr->WEATHRin == NULL) /* If unable to open data set */ 00050700
{
 /* ..set return msg and state */ 00050800
 strcpy(msgtext,"Error opening weather data set");
 strcpy(sqlstate,"38602");
}

```

```

} /* end openWeatherDS */ 00051200
 00051300
 00051400
 00051500
void readWeatherDS
(struct scr *scratchptr, /* in: ptr to scratch pad */ 00051600
 char *sqlstate, /* out: sqlstate */ 00051700
 char *msgtext, /* out: diag message text */ 00051800
)
/*** 00052000
* Reads the next record from the weather data set * 00052100
***** 00052200
{
 short int recordLength = 0; /* Receives len of current rec*/ 00052400
 recordLength /* */ 00052500
 = fread(WEATHRinBuffer, /* Read into WEATHRinBuffer */ 00052600
 1, /* ..a record */ 00052700
 sizeof(WEATHRinBuffer),/* ..<= len of WEATHRinBuffer */ 00052800
 scratchptr->WEATHRin); /* ..from the weather data set*/ 00052900
 scratchptr->WEATHRin); /* ..from the weather data set*/ 00053000
 00053100
 if(ferror(scratchptr->WEATHRin)) /* If an error occurs */ 00053200
 {
 /* ..set return msg and state */ 00053300
 strcpy(msgtext,"Error reading weather data set");
 strcpy(sqlstate,"38603");
 }
 else if(feof(scratchptr->WEATHRin))/* Else if end of file reached*/ 00053700
 moreWeatherRecs = NO; /* ..get ready to quit */ 00053800
} /* end readWeatherDS */ 00053900
 00054000
 00054100

void buildReturnRow /* Builds function return row */ 00054200
(
 char *city, /* out: name of city */ 00054300
 long int *temp_in_f, /* out: temp in fahrenheit */ 00054400
 long int *humidity, /* out: relative humidity */ 00054500
 char *wind, /* out: wind direction */ 00054600
 long int *wind_velocity, /* out: wind velocity */ 00054700
 double *barometer, /* out: barometric pressure */ 00054800
 char *forecast, /* out: forecast */ 00054900
 short int *niCity, /* out: indic var, city name */ 00055000
 short int *niTemp_in_f, /* out: indic var, temperature*/ 00055100
 short int *niHumidity, /* out: indic var, humidity */ 00055200
 short int *niWind, /* out: indic var, wind dir */ 00055300
 short int *niWind_velocity, /* out: indic var, wind veloc */ 00055400
 short int *niBarometer, /* out: indic var, baro press */ 00055500
 short int *niForecast, /* out: indic var, forecast */ 00055600
)
/*** 00055800
* Build a return row for the current call to the WEATHER table * 00055900
* function. * 00056000
***** 00056100
{
 char workBuff[6]; /* for datatype conversions */ 00056300
 00056400
 /* Move the city name to its table variable */ 00056500
 strcpy(city,pweatherRec->cityField,3);
 *niCity = 0;
 00056900
 00057000
 /* Move the temperature to its table var after making it numeric */ 00057100
 memset(workBuff,'0',6);
 strncpy(workBuff,pweatherRec->temp_in_fField,3);
 *temp_in_f = atoi(workBuff);
 *niTemp_in_f = 0;
 00057700
 00057800
 /* Move the humidity factor to its table var after making it numeric*/ 00057900
 memset(workBuff,'0',6);
 strncpy(workBuff,pweatherRec->humidityField,3);
 *humidity = atoi(workBuff);
 *niHumidity = 0;
 00058600
 /* Move the wind direction to its table variable */ 00058700
 strncpy(wind,pweatherRec->windField,5);
 *niWind = 0;
 00059100
 00059200
}
/*** 00059300

```

```

* Move the wind velocity to its table var after making it numeric * 00059400
*****memset(workBuff,'0',6); 00059500
strcpy(workBuff,pweatherRec->windVelocityField,3); 00059600
*wind_velocity = atoi(workBuff); 00059700
*nWind_velocity = 0; 00059800
00059900
00060000
*****/ 00060100
* Move the forecast to its table variable * 00060200
*****memset(workBuff,'0',6); 00060300
strcpy(workBuff,pweatherRec->barometerField,5); 00060400
*barometer = atof(workBuff); 00060500
*nBarometer = 0; 00060600
00060700
00060800
*****/ 00060900
* Move the forecast to its table variable * 00061000
*****strcpy(forecast,pweatherRec->forecastField,25); 00061100
*nForecast = 0; 00061200
00061300
00061400
00061500
00061600
00061700
00061800
void closeWeatherDS 00061900
(struct scr *scratchptr, /* in: ptr to scratch pad */ */
 char *sqlstate, /* out: sqlstate */ */
 char *msgtext /* out: diag message text */ */
)
*****/ 00062300
* Closes the weather data set and resets the file pointer in the * 00062400
* scratchpad area. * 00062500
*****/ 00062600
{
 if(fclose(scratchptr->WEATHRin) != 0)
 /* If unable to close data set*/ 00062900
 {
 /* ..set return msg and state */
 strcpy(msgtext,"Error closing weather data set");
 strcpy(sqlstate,"38604");
 }
 else
 scratchptr->WEATHRin = NULLCHAR; /* Otherwise, reset file ptr */ 00063500
}
/* end closeWeatherDS */

00063600
00063700
00063800
00063900
void freeWeatherDS 00064000
(char *sqlstate, /* out: sqlstate */ */
 char *msgtext /* out: diag message text */ */
)
*****/ 00064300
* Dynamically frees the weather data set, which has been allocated * 00064400
* to the WEATHRIN DD. * 00064500
*****/ 00064600
{
 __dyn_t free_ip; /* pointer to control block */ 00064700
 dyninit(&free_ip);
 free_ip._ddname = "WEATHRIN"; /* Set DD name of weather ds */ 00064800
 00064900
 if(dynfree(&free_ip) != 0)
 {
 sprintf(msgtext,"FREE failed for DDNAME %s. "
 "Error code=%hX, info code=%hX\n",
 free_ip._errcode,
 free_ip._errcode,
 free_ip._infocode);
 strcpy(sqlstate,"38605");
 }
}
/* end freeWeatherDS */
00065000
00065100
00065200
00065300
00065400
00065500
00065600
00065700
00065800
00065900
00066000
00066100
00066200

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8EUDN

Devuelve el día de la semana (de lunes a domingo) en el que una fecha determinada en formato ISO (AAAA-MM-DD) falla.

```

/*****
* Module name = DSN8EUDN (DB2 sample program) *
* DESCRIPTIVE NAME = Query day of the week (UDF) *
* LICENSED MATERIALS - PROPERTY OF IBM *
* 5675-DB2 *
* (C) COPYRIGHT 2000 IBM CORP. ALL RIGHTS RESERVED. *
* STATUS = VERSION 7 *
* Function: Returns the day of the week (Monday through Sunday) on *
* which a given date in ISO format (YYYY-MM-DD) falls. *
* Example invocation: *
* EXEC SQL SET :dayname = DAYNAME("2000-01-29"); *
* ==> dayname = Tuesday *
* Notes: *
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher *
* Restrictions: Assumes the Gregorian calendar was adopted in *
* September, 1752. Code modifications are required *
* to handle a different adoption date. *
* Module type: C++ program *
* Processor: IBM C/C++ for OS/390 V1R3 or higher *
* Module size: See linkedit output *
* Attributes: Re-entrant and re-usable *
* Entry Point: DSN8EUDN *
* Purpose: See Function *
* Linkage: DB2SQL *
* Invoked via SQL UDF call *
* Input: Parameters explicitly passed to this function: *
* - *ISOdateIn : pointer to a char[11], null-termin- *
* nated string having a date in ISO *
* format. *
* - *niISOdateIn : pointer to a short integer having *
* the null indicator variable for *
* *ISOdateIn. *
* - *fnName : pointer to a char[138], null-termin- *
* nated string having the UDF family *
* name of this function. *
* - *specificName: pointer to a char[129], null-termin- *
* nated string having the UDF specific *
* name of this function. *
* Output: Parameters explicitly passed by this function: *
* - *dayNameOut : pointer to a char[10], null-termin- *
* nated string to receive the dayname *
* for ISOdateIn. *
* - *niDayNameOut: pointer to a short integer to re- *
* ceive the null indicator variable *
* for *dayNameOut. *
* - *sqlstate : pointer to a char[06], null-termin- *
* nated string to receive the SQLSTATE. *
* - *message : pointer to a char[70], null-termin- *
* nated string to receive a diagnostic *
* message if one is generated by this *
* function. *
* Normal Exit: Return Code: SQLSTATE = 00000 *
* - Message: none *
* Error Exit: Return Code: SQLSTATE = 38601 *
* - Message: DSN8EUDN Error: No date entered *
* Return Code: SQLSTATE = 38602 *
* - Message: DSN8EUDN Error: Input date not valid *
* or not in ISO format" *
* External References: *
* - Routines/Services: *
* - strftime: Formatted time conversion routine *
* - from IBM C/C++ for z/OS run-time library *
* - strptime: Date and time conversion routine *
* - from IBM C/C++ for z/OS run-time library *
* - Data areas : None *
* - Control blocks : None *
*

```

```

/*
 * Pseudocode:
 * DSN8EUDN:
 * - Verify that a date was passed in:
 * - if *ISOdateIn blank or niISOdateIn is not 0, no date passed: */
 * - issue SQLSTATE 38601 and a diagnostic message.
 * - Use strftime to validate the entry
 * - if *ISOdateIn is not a valid ISO date:
 * - issue SQLSTATE 38602 and a diagnostic message
 * - Parse out the year, month, and day
 * - Compute the weekday number (0=Sunday, ..., 6=Saturday)
 * - Use strftime and strftime to convert the day number to the
 * full weekday name of the current locale.
 * End DSN8EUDN
 *
 * Change log:
 * 2004-02-25: Rewritten due to demise of IBM Open Class library
 *

extern "C" void DSN8EUDN /* Establish linkage */
(char *ISOdateIn, /* in: date to look up */
 char *dayNameOut, /* out: ISOdateIn's day name */
 short int *niISOdateIn, /* in: indic var, ISOdateIn */
 short int *niDayNameOut, /* out: indic var, dayNameOut */
 char *sqlstate, /* out: SQLSTATE */
 char *fnName, /* in: family name of function */
 char *specificName, /* in: specific name of func */
 char *message /* out: diagnostic message */
);
#pragma linkage(DSN8EUDN,fetchable) /* Establish linkage */

/***** C library definitions *****/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/***** Equates *****/
#define NULLCHAR '\0' /* Null character */

#define MATCH 0 /* Comparison status: Equal */
#define NOT_OK 0 /* Run status indicator: Error */
#define OK 1 /* Run status indicator: Good */

/***** DSN8EUDN functions *****/
/*****
/***** main routine *****/
/*****
void DSN8EUDN /* main routine */
(char *ISOdateIn, /* in: date to look up */
 char *dayNameOut, /* out: ISOdateIn's day name */
 short int *niISOdateIn, /* in: indic var, ISOdateIn */
 short int *niDayNameOut, /* out: indic var, dayNameOut */
 char *sqlstate, /* out: SQLSTATE */
 char *fnName, /* in: family name of function */
 char *specificName, /* in: specific name of func */
 char *message /* out: diagnostic message */
)
/*****
 * Returns the weekday name of the date in ISOdateIn.
 *
 * Assumptions:
 * - *ISOdateIn points to a char[11], null-terminated string
 * - *dayNameOut points to a char[10], null-terminated string
 * - *niISOdateIn points to a short integer
 * - *niDayNameOut points to a short integer
 * - *sqlstate points to a char[06], null-terminated string
 * - *fnName points to a char[138], null-terminated string
 * - *specificName points to a char[129], null-terminated string
 * - *message points to a char[70], null-terminated string

{
 /***** local variables *****/
 short int status = OK; /* DSN8EUDN run status */
 struct tm tmbuff; /* buffer for time.h tm struct */
 char *rc; /* gets strftime return code */

 char *tokPtr; /* string ptr for token parser */
 char workStr[11]; /* work copy of ISOdateIn parm */

 int yearInt; /* numeric copy of 4-digit yr */

```

```

char yearStr[05]; /* string copy of 4-digit year*/
int monthInt; /* numeric copy of month no. */
char monthStr[03]; /* string copy of month no. */
int dayInt; /* numeric copy of day no. */
char dayStr[03]; /* string copy of day no. */
int weekDayInt; /* week day no (0=Sun...6=Sat)*/
char weekDayStr[02]; /* string copy of week day no.*/
char *isoFormat; /* format of isoDate: */
= "%Y-%m-%d"; /* %Y = YYYY, %m = MM, %d = DD*/
char *weekDateFormat; /* format of weekday: */
= "%W"; /* %W = weekday */
char *weekDayLongNameFormat/* format of weekday long name*/
= "%A"; /* %A = weekday long name */

/***
* Verify that something has been passed in
***/
if(*niIS0dateIn != 0 || (strlen(IS0dateIn) == 0))
{
 status = NOT_OK;
 strcpy(message,
 "DSN8EUDN Error: No date entered");
 strcpy(sqlstate, "38601");
}

/***
* Verify that the input looks like a date
***/
if(status == OK)
{
 rc = strptime(IS0dateIn,isoFormat,&tmbuff);
 if(rc == NULL) /* Unable to convert IS0dateIn*/
 {
 status = NOT_OK;
 strcpy(message,
 "DSN8EUDN Error: Input date not valid "
 "or not in ISO format");
 strcpy(sqlstate, "38602");
 }
}

/***
* Parse the 4-digit year, the month no., and day no. from IS0dateIn
***/
if(status == OK)
{
 strcpy(workStr,IS0dateIn);

 tokPtr = strtok(workStr,"- ");
 strcpy(yearStr,tokPtr);
 yearInt = atoi(yearStr);

 tokPtr = strtok(NULL,"- ");
 strcpy(monthStr,tokPtr);
 monthInt = atoi(monthStr);

 tokPtr = strtok(NULL,"- ");
 strcpy(dayStr,tokPtr);
 dayInt = atoi(dayStr);
}

/***
* Get the weekday name of IS0dateIn
***/
if(status == OK)
{
 /* ****
 * Leap year allowance: Shift Jan and Feb to end of prev year
 */
 if(monthInt < 3)
 { monthInt += 12;
 yearInt--;
 }

 /* ****
 * Calculate weekday no. with Sunday basis
 */
 weekDayInt = ((13 * monthInt) + 3) / 5 /* xform months */
 + dayInt /* + days */
 + yearInt /* + years */
 + yearInt / 4 /* + leapyear/4 */

```

```

 - yearInt / 100 /* - leapyear/100 */
 + yearInt / 400 /* + leapyear/400 */
 + 1 /* + Sunday basis */
) % 7; /* % days per wk */

/***
* adjust for pre-gregorian calendar (September 1752) *
***/
if((yearInt < 1752) || (yearInt == 1752 && monthInt < 9))
{ if(weekDayInt > 3)
 weekDayInt = weekDayInt - 4;
else
 weekDayInt = weekDayInt + 3;
}

/***
* convert day of week from numeric to string *
**/
sprintf(weekDayStr,"%02d",weekDayInt);

/***
* Convert day of week from numeric string to day name *
**/
rc = strftime(weekDayStr,weekDayFormat,&tmbuff);
*rc = strftime(dayNameOut,10,weekDayLongNameFormat,&tmbuff);
}

/***
* If weekday name was obtained, clear the message buffer and sql- *
* state, and unset the SQL null indicator for dayNameOut. *
**/
if(status == OK)
{
 *niDayNameOut = 0;
 message[0] = NULLCHAR;
 strcpy(sqlstate,"00000");
}

/***
* If errors occurred, clear the dayNameOut buffer and set the SQL *
* NULL indicator. A diagnostic message and the SQLSTATE have been *
* set where the error was detected. *
**/
else
{
 dayNameOut[0] = NULLCHAR;
 *niDayNameOut = -1;
}

return;
} /* end DSN8EUDN */

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8EUMN

Devuelve el nombre de calendario del nombre del mes en el que una fecha en formato ISO (AAAA-MM-DD) falla.

```

/***
* Module name = DSN8EUMN (DB2 sample program) *
* *
* DESCRIPTIVE NAME = Query calendar month name (UDF) *
* *
* *
* LICENSED MATERIALS - PROPERTY OF IBM *
* 5675-DB2 *
* (C) COPYRIGHT 1998, 2000 IBM CORP. ALL RIGHTS RESERVED. *
* *
* STATUS = VERSION 7 *
* *
* *
* Function: Returns the calendar name of the month name in *
* which a given date in ISO format (YYYY-MM-DD) falls. *
* *
* Example invocation: *
* *

```

```

* EXEC SQL SET :monthname = MONTHNAME("2000-01-29");
* ==> monthname = January
* Notes:
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher
* Restrictions:
*
* Module type: C++ program
* Processor: IBM C/C++ for OS/390 V1R3 or higher
* Module size: See linkedit output
* Attributes: Re-entrant and re-usable
*
* Entry Point: DSN8EUMN
* Purpose: See Function
* Linkage: DB2SQL
* Invoked via SQL UDF call
*
* Input: Parameters explicitly passed to this function:
* - *ISOdateIn : pointer to a char[11], null-terminated
* string having a date in ISO
* format.
* - *niISOdateIn : pointer to a short integer having
* the null indicator variable for
* *ISOdateIn.
* - *fnName : pointer to a char[138], null-terminated
* string having the UDF family
* name of this function.
* - *specificName: pointer to a char[129], null-terminated
* string having the UDF specific
* name of this function.
*
* Output: Parameters explicitly passed by this function:
* - *monthNameOut: pointer to a char[10], null-terminated
* string to receive the month-
* name for ISOdateIn.
* - *niMonthNameOut: pointer to a short integer to re-
* ceive the null indicator variable
* for *monthNameOut.
* - *sqlstate : pointer to a char[06], null-termin-
* ated string to receive the SQLSTATE.
* - *message : pointer to a char[70], null-termin-
* ated string to receive a diagnostic
* message if one is generated by this
* function.
*
* Normal Exit: Return Code: SQLSTATE = 00000
* - Message: none
*
* Error Exit: Return Code: SQLSTATE = 38601
* - Message: DSN8EUMN Error: No date entered
* Return Code: SQLSTATE = 38602
* - Message: DSN8EUMN Error: Input date not valid
* or not in ISO format"
*
* External References:
* - Routines/Services:
* - strftime: Formatted time conversion routine
* - from IBM C/C++ for z/OS run-time library
* - strptime: Date and time conversion routine
* - from IBM C/C++ for z/OS run-time library
* - Data areas : None
* - Control blocks : None
*
*
* Pseudocode:
* DSN8EUMN:
* - Verify that a date was passed in:
* - if *ISOdateIn blank or niISOdateIn is not 0, no date passed:
* - issue SQLSTATE 38601 and a diagnostic message.
* - Use strptime to validate the entry and convert the date to tm
* - if *ISOdateIn is not a valid ISO date:
* - issue SQLSTATE 38602 and a diagnostic message
* - Use strftime to get the full monthname for the locale
* End DSN8EUMN
*
* Change log:
* 2004-02-25: Rewritten due to demise of IBM Open Class library
*
*****extern "C" void DSN8EUMN /* Establish linkage */
(char *ISOdateIn, /* / * in: date to look up */

```

```

char *monthNameOut, /* out: ISOdateIn's month name*/
short int *niISOdateIn, /* in: indic var, ISOdateIn */
short int *niMonthNameOut, /* out: indic var,monthNameOut*/
char *sqlstate, /* out: SQLSTATE */
char *fnName, /* in: family name of functions*/
char *specificName, /* in: specific name of func */
char *message /* out: diagnostic message */
);

#pragma linkage(DSN8EUMN,fetchable) /* Establish linkage */

/****** C library definitions *****/
#include <stdio.h>
#include <time.h>

/****** Equates *****/
#define NULLCHAR '\0' /* Null character */

#define MATCH 0 /* Comparison status: Equal */
#define NOT_OK 0 /* Run status indicator: Error*/
#define OK 1 /* Run status indicator: Good */

/****** DSN8EUMN functions *****/
/****** main routine *****/
void DSN8EUMN /* main routine */
(char *ISOdateIn, /* in: date to look up */
 char *monthNameOut, /* out: ISOdateIn's month name*/
 short int *niISOdateIn, /* in: indic var, ISOdateIn */
 short int *niMonthNameOut, /* out: indic var,monthNameOut*/
 char *sqlstate, /* out: SQLSTATE */
 char *fnName, /* in: family name of function*/
 char *specificName, /* in: specific name of func */
 char *message /* out: diagnostic message */
)
/****** */
* Returns the name of the month for the date in isoDate. *
*
* Assumptions:
* - *ISOdateIn points to a char[11], null-terminated string
* - *monthNameOut points to a char[10], null-terminated string
* - *niISOdateIn points to a short integer
* - *niMonthNameOut points to a short integer
* - *sqlstate points to a char[06], null-terminated string
* - *fnName points to a char[138], null-terminated string
* - *specificName points to a char[129], null-terminated string
* - *message points to a char[70], null-terminated string
***** */
{

/****** local variables *****/
short int status = OK; /* DSN8EUMN run status */
struct tm tmbuff; /* buffer for time.h tm struct*/
char *rc; /* gets strf/ptime return code*/
char *isoFormat; /* format of isoDate: */
= "%Y-%m-%d"; /* %Y = YYYY, %m = MM, %d = DD*/
char *fullMonthName; /* format of fullMonthName */
= "%B"; /* %B = full month name */

/****** */
* Verify that something has been passed in *
***** */
if(*niISOdateIn != 0 || (strlen(ISOdateIn) == 0))
{
 status = NOT_OK;
 strcpy(message,
 "DSN8EUMN Error: No date entered");
 strcpy(sqlstate, "38601");
}

/****** */
* Convert ISOdateIn to C tm format *
***** */
if(status == OK)
{
 rc = strptime(ISOdateIn,isoFormat,&tmbuff);
 if(rc == NULL) /* Unable to convert ISOdateIn*/
 {
 status = NOT_OK;
 strcpy(message,

```

```

 "DSN8EUMN Error: Input date not valid "
 "or not in ISO format");
strcpy(sqlstate, "38602");
}

/***
* Convert the date from C tm format to the locale's full monthname *
***/
if(status == OK)
{ *rc = strftime(monthNameOut,10,fullMonthName,&tmbuff);
}

/***
* If month name was obtained, clear the message buffer and sql- *
* state, and unset the SQL null indicator for monthNameOut. *
***/
if(status == OK)
{
 *niMonthNameOut = 0;
 message[0] = NULLCHAR;
 strcpy(sqlstate,"00000");
}

/***
* If errors occurred, clear the monthNameOut buffer and set the SQL*
* NULL indicator. A diagnostic message and the SQLSTATE have been *
* set where the error was detected.
***/
else
{
 monthNameOut[0] = NULLCHAR;
 *niMonthNameOut = -1;
}

return;
} /* end DSN8EUMN */

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8DLPL

Llena las columnas PSEG\_PHOTO (500K BLOB) y BMP\_PHOTO (100K BLOB) de la tabla de ejemplo EMP\_PHOTO\_RESUME con datos leídos de los conjuntos de datos secuenciales.

```

/***
* Module name = DSN8DLPL (DB2 sample program)
*
* DESCRIPTIVE NAME = Populate LOB columns that exceed 32K with data
* read from sequential data sets.
*
*
* LICENSED MATERIALS - PROPERTY OF IBM
* 5655-DB2
* (C) COPYRIGHT 1997 IBM CORP. ALL RIGHTS RESERVED.
*
* STATUS = VERSION 6
*
* Function: Populates the PSEG_PHOTO (500K BLOB) and BMP_PHOTO (100K
* BLOB) columns of the EMP_PHOTO_RESUME sample table with
* data read from sequential data sets.
*
* LOB locators are used to avoid having to contain all the
* data in the application's storage.
*
* Notes:
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher
*
* Restrictions:
*
* Module type: C program
* Processor: IBM C/C++ for OS/390 V1R3 or subsequent release
* Module size: See linkedit output
* Attributes: Re-entrant and re-usable
*
* Entry Point: CEESTART (Language Environment entry point)
* Purpose: See Function

```

```

* Linkage: Standard MVS program invocation, no parameters *
*
* Input: Symbolic label/name = PSEGINnn, where 00 <= nn <= 99 *
* Description = PSEG photo image data *
*
* Symbolic label/name = BMPINnn, where 00 <= nn <= 99 *
* Description = BMP photo image data *
*
* Output: Symbolic label/name = SYSPRINT *
* Description = Report and messages *
*
*
* Normal Exit: Return Code = 0000 *
* - Message: none *
*
* Error Exit: Return Code = 0008 *
* - Message: *** ERROR: DSN8DLPL DB2 Sample Program *
* Unable to open BMPINnn DD data *
* set. Processing terminated. *
*
* - Message: *** ERROR: DSN8DLPL DB2 Sample Program *
* Unexpected SQLCODE encountered *
* at location xxxx *
* Error detailed below *
* Processing terminated *
* (DSNTIAR-formatted message *
* follows). *
*
* External References: *
* - Routines/Services: DSNTIAR *
* - Data areas : DSNTIAR error_message *
* - Control blocks : None *
*
*
* Pseudocode: *
* DSN8DLPL:
* - Set DD counter (nn) to 00 *
* - Do while more PSEGINnn DD's to process *
* - Call openPSEGfile to open the data set associated with
* DD PSEGINnn *
* - Call getPSEGrec to read the first record of the data set
* - Extract the employee serial from this record
* - Call openBMPfile to open the data set associated with
* DD BMPINnn *
* - Call getBMPrec to read the first record of the data set
* - Call primeBLOBcols to:
* (a) UPDATE the PSEG_PHOTO and BMP_PHOTO columns of the
* employee's row in the EMP_PHOTO_RESUME table with the
* contents of these first records
* (b) SELECT the PSEG_PHOTO and BMP_PHOTO columns back into
* BLOB locators
* - Call getPSEGrec to read the next record from the PSEGINnn DD
* - Do while not end of file for the PSEGINnn DD
* - Call buildPSEGcol to append the current PSEGINnn record to
* the PSEG BLOB locator
* - Call getPSEGrec to read the next record from PSEGINnn
* - Call getBMPrec to read the next record from the BMPINnn DD
* - Do while not end of file for the BMPINnn DD
* - Call buildBMPcol to append the current BMPINnn record to
* the BMP BLOB locator
* - Call getBMPrec to read the next record from BMPINnn
* - Call updateBLOBcols to apply the BLOB locators to the
* PSEG_PHOTO and BMP_PHOTO columns of the employee's row in
* the EMP_PHOTO_RESUME table
* - If all went well, call commitWorkUnit to commit the changes
* - Else call rollbackWorkUnit to roll back the changes
* - Print a status line
* - Close the PSEGINnn and BMPINnn DD's
* - Increment DD counter (nn) by 1.
* - If an SQL error occurs, invoke the sql_error routine to generate
* and display message text
* End DSN8DLPL
*
* openPSEGfile:
* - Open the data set associated with the PSEGINnn DD
* - If the open fails, set validDD to false
* End openPSEGfile
*
* getPSEGrec:
* - Read a record from the data set associated with the PSEGINnn DD
* - If end of file, set morePSEGrecs to false
* End getPSEGrec

```

```

/*
 * openBMPfile:
 * - Open the data set associated with the BMPINnn DD
 * End openBMPfile
 *
 * getBMPrec:
 * - Read a record from the data set associated with the BMPINnn DD
 * - If end of file, set moreBMPrecs to false
 * End getBMPrec
 *
 * primeBLOBcols:
 * - extract the employee serial from bytes 10-15 of the PSEG
 * buffer.
 * - UPDATE the PSEG_PHOTO and BMP_PHOTO columns for the employee's
 * row in the EMP_PHOTO_RESUME table from the PSEG and BMP records
 * - SELECT the PSEG_PHOTO and BMP_PHOTO columns for the employee
 * into LOB locators b1PSEG1 and b1BMP1
 * End primeBLOBcols
 *
 * buildPSEGcol:
 * - append the contents of the PSEG input record to the PSEG BLOB
 * locator b1PSEG1 and assign to BLOB locator b1PSEG2
 * - free BLOB locator b1PSEG1
 * - set BLOB locator b1PSEG1 from BLOB locator b1PSEG2
 * - free BLOB locator b1PSEG2
 * End buildPSEGcol
 *
 * buildBMPcol:
 * - append the contents of the BMP input record to the BMP BLOB
 * locator b1BMP1 and assign to BLOB locator b1BMP2
 * - free BLOB locator b1BMP1
 * - set BLOB locator b1BMP1 from BLOB locator b1BMP2
 * - free BLOB locator b1BMP2
 * End buildBMPcol
 *
 * updateBLOBcols:
 * - UPDATE the PSEG_PHOTO and BMP_PHOTO columns for the employee's
 * row in the EMP_PHOTO_RESUME table from the PSEG and BMP BLOB
 * locators BMP b1PSEG1 and b1BMP1
 * End updateBLOBcols
 *
 * commitWorkUnit:
 * - commit the changes
 * End commitWorkUnit
 *
 * rollbackWorkUnit:
 * - roll back the changes
 * End rollbackWorkUnit
 *
 * sql_error:
 * - call DSNTIAR to format the unexpected SQLCODE.
 * End sql_error
 *
***** C library definitions *****/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/* Equates */
#define NO 0 /* False */
#define YES 1 /* True */

#define NOT_OK 0 /* Run status indicator: Error */
#define OK 1 /* Run status indicator: Good */

#define TIAR_DIM 10 /* Max no. of DSNTIAR msgs */
#define TIAR_LEN 80 /* Length of DSNTIAR messages */

/* Files */
FILE *BMPin; /* pointer to BMP input file */
FILE *PSEGin; /* pointer to PSEG input file */

/* Global Storage */
int status = OK; /* run status flag */

char PSEGinDD[12]; /* PSEGin DD template */
char BMPinDD[12]; /* BMPin DD template */
short int DDcounter = 0; /* DD allocation counter */

```

```

char DDnum[2]; /* DD number string template */
short int validDD = YES; /* unprocessed DD indicator */

short int morePSEGrecs = YES; /* eof indicator for PSEGINnn */
short int moreBMPrecs = YES; /* eof indicator for BMPINnn */

short int PSEGblkLen = 0; /* length of PSEG input block */
short int PSEGblkPos = 0; /* offset in PSEG input block */
short int PSEGrecLen = 0; /* length of PSEG input records */
short int PSEGrecPos = 0; /* offset in PSEG input records */
long int PSEGcolLen = 0; /* length of PSEG column data */

short int BMPblkLen = 0; /* length of BMP input block */
short int BMPblkPos = 0; /* offset in BMP input block */
short int BMPrecLen = 0; /* length of BMP input record */
short int BMPrecPos = 0; /* offset in BMP input record */
long int BMPcolLen = 0; /* length of BMP column data */

int byteIn; /* current incoming byte */

/*** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

/*** DB2 Tables *****/
EXEC SQL DECLARE EMP_PHOTO_RESUME TABLE
(
 EMPNO CHAR(06) NOT NULL,
 EMP_ROWID ROWID,
 PSEG_PHOTO BLOB(500K),
 BMP_PHOTO BLOB(100K),
 RESUME CLOB(5K)
);

/*** DB2 Host and Null Indicator Variables *****/
EXEC SQL BEGIN DECLARE SECTION;
char hvEMPNO[7]; /* Host var for employee no. */
SQL TYPE IS BLOB(8K) PSEGinRec; /* Area for PSEG input record */
short int niPSEG_PHOTO = 0; /* Null ind for PSEG photo col */

SQL TYPE IS BLOB(8K) BMPinRec; /* Area for BMP input record */
short int niBMP_PHOTO = 0; /* Null ind for BMP photo col */
EXEC SQL END DECLARE SECTION;

/*** DB2 LOB Locator Variables *****/
EXEC SQL BEGIN DECLARE SECTION;
SQL TYPE IS BLOB_LOCATOR b1PSEG1; /* BLOB loc for PSEG photo col */
SQL TYPE IS BLOB_LOCATOR b1PSEG2; /* BLOB loc for PSEG photo col */
SQL TYPE IS BLOB_LOCATOR b1BMP1; /* BLOB loc for BMP photo col */
SQL TYPE IS BLOB_LOCATOR b1BMP2; /* BLOB loc for BMP photo col */
EXEC SQL END DECLARE SECTION;

/*** DB2 Message Formatter *****/
struct error_struct { /* DSNTIAR message structure */
 short int error_len;
 char error_text[TIAR_DIM][TIAR_LEN];
} error_message = {TIAR_DIM * (TIAR_LEN)};

#pragma linkage(dsntiar, OS)

extern short int dsntiar(struct sqlca *sqlca,
 struct error_struct *msg,
 int *len);

/*** Global Functions *****/
int main(void); /* main routine */
void openPSEGfile(void); /* open PSEGINnn DD file */
void getPSEGrec(void); /* read PSEG image file */
void openBMPfile(void); /* open BMPINnn DD file */
void getBMPrec(void); /* read BMP image file */
void primeBLOBcols(void); /* set PSEG and BMP BLOB locs */
void buildPSEGcol(void); /* add to PSEG BLOB locator */
void buildBMPcol(void); /* add to BMP BLOB locator */
void updateBLOBcols(void); /* apply PSEG and BMP locs */
void commitWorkUnit(void); /* commit changes */
void rollbackWorkUnit(void); /* roll back changes */
void sql_error(char *locmsg); /* generate msg for SQL error */

```

```

***** main routine *****
Initialization
***** Write identification header *****
printf("*****\n");
printf("*****\n");
printf(" * DSN8DPL DB2 Sample Program\n");
printf("*****\n");
printf("*****\n");

***** Processing *****
***** Cycle through PSEGINnn and BMPINnn DD pairs, incrementing the DD *****
* counter, until all pairs have been processed. *
for(DDcounter=0; validDD == YES && status == OK; DDcounter++)
{

 * Fetch the PSEG data from the current PSEGINnn DD. The first *
 * record contains the serial number of the employee associated *
 * with the photo. *

 openPSEGfile();
 if(validDD == YES && status == OK)
 getPSEGrec();

 * Fetch the first record from the BMPINnn DD *

 if(validDD == YES && status == OK)
 openBMPfile();
 if(validDD == YES && status == OK)
 getBMPrec();

 * Init the PSEG and BMP BLOB table columns for the employee *

 if(validDD == YES && status == OK)
 primeBLOBcols();

 * Read the second records from the PSEG and BMP data sets *

 if(validDD == YES && status == OK)
 getPSEGrec();
 if(validDD == YES && status == OK)
 getBMPrec();

 * Append remaining PSEG recs to PSEG photo col using BLOB loc *

 while(morePSEGrecs == YES && validDD == YES && status == OK)
 {
 buildPSEGcol();
 if(status == OK)
 getPSEGrec();
 }

 * Append remaining BMP recs to BMP photo col using BLOB locator*

 while(moreBMPrecs == YES && validDD == YES && status == OK)
 {
 buildBMPcol();
 if(status == OK)
 getBMPrec();
 }

 * Apply the data associated with the PSEG and BMP BLOB locators*
 * to the table *
}

```

```

*****+
if(validDD == YES && status == OK)
 updateBLOBcols();

/*****
* If clear status, commit the work unit; otherwise, rollback *
*****+
if(validDD == YES)
 if(status == OK)
 commitWorkUnit();
 else
 rollbackWorkUnit();

/*****
* Print report line
*****+
if(validDD == YES && status == OK)
{
 printf(" * LOB population statistics for employee "
 " number %s follow:\n", hvEMPNO);
 printf(" * - PSEG photo bytes: %d\n", PSEGcolLen);
 printf(" * - BMP photo bytes: %d\n", BMPcolLen);
 printf("*****"
 "*****\n");
}

/*****
* Close data sets for current PSEGINnn and BMPINnn DDs
*****+
fclose(PSEGin);
fclose(BMPin);
} /* end for(DDcounter=0; validDD == YES && status == OK ... */

/*****
***** Cleanup *****
*****+
/* Set return code
 * Set return code
 */
if(status == OK)
 return(0);
else
 return(8);

} /* end main */

void openPSEGfile(void)
/*****
* Opens the data set associated with the PSEGINnn DD, where "nn" is
* the current setting of the DD counter from the main loop.
*
* If the DD cannot be allocated, then no further data sets remain to
* be processed so signal end of job.
*****+
{
 /*****
 * initialize work variables
 */
morePSEGrecs = YES;
PSEGblkPos = 0;
PSEGblkLen = 0;

/*****
* form the DD name for the next PSEG data set
*/
strcpy(PSEGinDD,"DD:PSEGIN\0"); /* init PSEGin DD template */
sprintf(DDnum,"%02d",DDcounter); /* convert DD cntr to string */
strcat(PSEGinDD,DDnum); /* form PSEGINnn DD name */

/*****
* open the PSEGINnn DD data set
*/
PSEGin = fopen(PSEGinDD,"rb,rcfm=u");

if(PSEGin == NULL) /* if no pointer returned */
 validDD = NO; /* .. no more data sets left */
} /* end openPSEGfile */

```

```

void getPSEGrec(void)
/*****
 * Called by the main routine to read the next record from the data *
 * set associated with the current PSEGINnn DD into a buffer, PSEGin- *
 * Rec. *
 *
 * If this is the first record from the PSEGINnn DD data set, it con- *
 * tains the serial number of an employee in bytes 10-15 and it will * *
 * be UPDATED into the PSEG_PHOTO column of that employee's row in * *
 * the sample EMP_PHOTO_RESUME table. This column and row will then * *
 * be SELECTed into a BLOB locator, b1PSEG1, to be used for accumu- * *
 * lating the remaining records from the current PSEGINnn DD data set * *
 * to form a complete PSEG_PHOTO entry for the current employee. *
 *
 * If this is not the first record from the PSEGINnn DD data set, it * *
 * will be appended to previously read records for this data set in * *
 * a DB2 data area associated with the BLOB locator, b1PSEG1. *
 *
 * When all records of the data set have been read and accumulated in * *
 * the locator area, the locator will be applied to the PSEG_PHOTO * *
 * column of the current employee's row in the EMP_PHOTO_RESUME table.* *
 *****/
 * Because the C language is not record-oriented in the sense of MVS *
 * data sets, it's necessary to treat the PSEG data set, which has a *
 * variable-blocked format, as an unformatted dataset in order to *
 * access the block descriptor word (BDW) of each input block and the *
 * record descriptor word (RDW) of each input record. *
 *
 * Each RDW provides the number of bytes of data in its record, *
 * including 4 bytes for itself. *
 *
 * Each BDW provides the number of bytes of data in its block, *
 * including 4 bytes for each RDW in the block and 4 bytes for *
 * itself. *
 *****/
{
 /*****
 * initialize work variables
 */
 PSEGrecLen = 0;
 PSEGrecPos = 0;
 PSEGinRec.length = 0;

 /*****
 * read the 1st byte of the record
 */
 byteIn = getc(PSEGin);
 /*****
 * get remaining bytes of the record if not EOF
 */
 if(byteIn != EOF)
 {
 /*****
 * if at end of block, read next BDW
 */
 if(PSEGblkPos >= PSEGblkLen && PSEGrecPos >= PSEGrecLen)
 {
 /*****
 * length of block = (16**2) * BDW[0]
 * + (16**0) * BDW[1]
 * - 4 (length of BDW)
 */
 PSEGblkLen = 256 * byteIn;
 byteIn = getc(PSEGin);
 PSEGblkLen = PSEGblkLen + byteIn - 4;

 /*****
 * skip remainder of BDW
 */
 byteIn = getc(PSEGin);
 byteIn = getc(PSEGin);
 PSEGblkPos = 0;

 /*****
 * read first byte of RDW
 */
 byteIn = getc(PSEGin);
 }
 /*****
 * process the RDW
 */
 * length of record = (16**2) * RDW[0]
 }
}

```

```

* + (16**0) * RDW[1] *
* - 4 (length of RDW) *
*****PSEGrecLen = 256 * byteIn;
byteIn = getc(PSEGin);
PSEGrecLen = PSEGrecLen + byteIn - 4;

/* skip remainder of RDW */
byteIn = getc(PSEGin);
byteIn = getc(PSEGin);
PSEGrecPos = 0;

/* update position in block */
PSEGblkPos = PSEGblkPos + PSEGrecLen + 4;
}

/* build the PSEG record according to the record length */
while(PSEGrecPos < PSEGrecLen && byteIn != EOF)
{
 byteIn = getc(PSEGin);
 PSEGinRec.data[PSEGinRec.length++] = byteIn;
 PSEGrecPos++;
}

/* signal end of file when applicable */
if(byteIn == EOF)
 morePSEGrecs = NO;

} /* end getPSEGrec */

void openBMPfile(void)
/*
* Opens the data set associated with the BMPINnn DD, where "nn" is
* the current setting of the DD counter from the main loop.
*
* If the DD cannot be allocated, then an error has occurred because
* each BMPINnn DD must be paired with a PSEGINnn data set.
*/
{
 /* initialize work variables */
moreBMPrecs = YES;
BMPblkPos = 0;
BMPblkLen = 0;

/* form the DD name for the next BMP data set */
strcpy(BMPinDD, "DD:BMPIN\0"); /* init BMPin DD template */
sprintf(DDnum,"%02d",DDcounter); /* convert DD cntr to string */
strcat(BMPinDD,DDnum); /* form BMPINnn DD name */

/* open the current BMPINnn DD data set */
BMPin = fopen(BMPinDD,"rb,recfm=u");

if(BMPin == NULL)
{
 printf("*****\n");
 printf("*** ERROR: DSN8DLPL DB2 Sample Program\n");
 printf("*** Unable to open BMPIN%ns DD data set\n",
 DDnum);
 printf("*** Processing terminated.\n");
 printf("*****\n");
 status = NOT_OK;
}
} /* end openBMPfile */

void getBMPrec(void)
/*

```

```

* Called by the main routine to read the next record from the data *
* set associated with the current BMPINnn DD into a buffer, BMPinRec.* *
*
* If this is the first record from the BMPINnn DD data set, it con- *
* tains the serial number of an employee in bytes 10-15 and it will *
* be UPDATED into the BMP_PHOTO column of that employee's row in *
* the sample EMP_PHOTO_RESUME table. This column and row will then *
* be SELECTed into a BLOB locator, blBMP1, to be used for accumulat- *
* ing the remaining records from the current BMPINnn DD data set to *
* form a complete BMP_PHOTO entry for the current employee. *
*
* If this is not the first record from the BMPINnn DD data set, it *
* will be appended to previously read records for this data set in *
* a DB2 data area associated with the BLOB locator, blBMP1. *
*
* When all records of the data set have been read and accumulated in *
* the locator area, the locator will be applied to the BMP_PHOTO *
* column of the current employee's row in the EMP_PHOTO_RESUME table.* *
***** ****
* Because the C language is not record-oriented in the sense of MVS *
* data sets, it's necessary to treat the BMP data set, which has a *
* variable-blocked format, as an unformatted dataset in order to *
* access the block descriptor word (BDW) of each input block and the *
* record descriptor word (RDW) of each input record. *
*
* Each RDW provides the number of bytes of data in its record, *
* including 4 bytes for itself. *
*
* Each BDW provides the number of bytes of data in its block, *
* including 4 bytes for each RDW in the block and 4 bytes for *
* itself. *
***** ****
{
 /*****
 * initialize work variables
 */
 BMPrecLen = 0;
 BMPrecPos = 0;
 BMPinRec.length = 0;

 /*****
 * read the 1st byte of the record
 */
 byteIn = getc(BMPin);
 /*****
 * get remaining bytes of the record if not EOF
 */
 if(byteIn != EOF)
 {
 /*****
 * if at end of block, read next BDW
 */
 if(BMPblkPos >= BMPblkLen)
 {
 /*****
 * length of block = (16**2) * BDW[0]
 * + (16**0) * BDW[1]
 * - 4 (length of BDW)
 */
 BMPblkLen = 256 * byteIn;
 byteIn = getc(BMPin);
 BMPblkLen = BMPblkLen + byteIn - 4;

 /*****
 * skip remainder of BDW
 */
 byteIn = getc(BMPin);
 byteIn = getc(BMPin);
 BMPblkPos = 0;

 /*****
 * read first byte of RDW
 */
 byteIn = getc(BMPin);
 }
 /*****
 * process the RDW
 */
 * length of record = (16**2) * RDW[0]
 * + (16**0) * RDW[1]
 * - 4 (length of RDW)
 */
 }
}

```

```

BMPrecLen = 256 * byteIn;
byteIn = getc(BMPin);
BMPrecLen = BMPrecLen + byteIn - 4;

/****** */
* skip remainder of RDW
*****/
byteIn = getc(BMPin);
byteIn = getc(BMPin);
BMPrecPos = 0;

/****** */
* update position in block
*****/
BMPblkPos = BMPblkPos + BMPrecLen + 4;
}

/****** */
* build the BMP record according to the record length
*****/
while(BMPrecPos < BMPrecLen && byteIn != EOF)
{
 byteIn = getc(BMPin);
 BMPinRec.data[BMPinRec.length++] = byteIn;
 BMPrecPos++;
}

/****** */
* signal end of file when applicable
*****/
if(byteIn == EOF)
 moreBMPrecs = NO;

} /* end getBMPrec */

void primeBLOBcols(void)
/****** */
* Called by the main routine to apply the first PSEG input record
* (from getPSEGrec) and the first BMP input record (from getBMPrec)
* to the PSEG_PHOTO and BMP_PHOTO BLOB columns, respectively, and
* then fetch those columns using BLOB locators.
*
* The PSEG BLOB locator will be used by the buildPSEGcol function
* to build a BLOB entity of up to 500K bytes from the remaining
* PSEGin records without consuming application workspace.
*
* The BMP BLOB locator will be used by the buildBMPcol function to
* build a BLOB entity of up to 500K bytes from the remaining BMPin
* records, again without consuming application workspace.
*
* When all PSEG and BMP records have been processed, the data will
* be applied from the BLOB locators to the EMP_PHOTO_RESUME table by
* the updateBLOBcols function.
*****/
{
 char *empser; /*

 /****** */
 * Extract the employee number from bytes 10-15 of the PSEG record
 *****/
 empser = &PSEGinRec.data[9];
 strncpy(hvEMPNO,empser,6);

 /****** */
 * Initialize the BLOB columns with data from the 1st input records
 *****/
 EXEC SQL UPDATE EMP_PHOTO_RESUME
 SET PSEG_PHOTO = :PSEGinRec,
 BMP_PHOTO = :BMPinRec
 WHERE EMPNO = :hvEMPNO;

 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("primeBLOBcols @ UPDATE");
 }

 /****** */
 * Select the initial BLOB data into locators
 *****/
 if(status == OK)

```

```

{
 EXEC SQL SELECT PSEG_PHOTO, BMP_PHOTO
 INTO :b1PSEG1 :niPSEG_PHOTO,
 :b1BMP1 :niBMP_PHOTO
 FROM EMP_PHOTO_RESUME
 WHERE EMPNO = :hvEMPNO;

 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("primeBL0Bcols @ SELECT");
 }
}

/*
 * Set initial lengths of PSEG_PHOTO and BMP_PHOTO columns
 */
PSEGcolLen = PSEGinRec.length;
BMPcolLen = BMPinRec.length;

} /* end primeBL0Bcols */

void buildPSEGcol(void)
/*
 * Called by the main routine to build a PSEG_PHOTO column entry for *
 * the current employee. *
 * *
 * This is done by appending the current record of the PSEG input file*
 * (from getPSEGrec) to the entity associated with b1PSEG1, the BLOB *
 * locator for the PSEG_PHOTO column. *
 * *
 * When all PSEG input records have been appended to this entity, the *
 * updateBL0Bcols function will be invoked to update the PSEG_PHOTO *
 * column in the EMP_PHOTO_RESUME table from b1PSEG1. *
 */
{
 /*
 * Generate a new BLOB locator that contains the current input *
 * record appended to the current PSEG_PHOTO locator *
 */
 EXEC SQL SET :b1PSEG2 = SUBSTR(:b1PSEG1,1,LENGTH(:b1PSEG1))
 || :PSEGinRec;

 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("buildPSEGcol @ SET LOCATOR #2");
 }

 /*
 * Regenerate the PSEG_PHOTO locator from the updated locator *
 */
 if(status == OK)
 {
 EXEC SQL FREE LOCATOR :b1PSEG1;
 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("buildPSEGcol @ FREE LOCATOR #1");
 }
 }

 if(status == OK)
 {
 EXEC SQL SET :b1PSEG1 = :b1PSEG2;
 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("buildPSEGcol @ SET LOCATOR #1");
 }
 }

 if(status == OK)
 {
 EXEC SQL FREE LOCATOR :b1PSEG2;
 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("buildPSEGcol @ FREE LOCATOR #2");
 }
 }
}

```

```

}

/*********************

 * Update length of PSEG_PHOTO column

if(status == OK)

 PSEGcolLen = PSEGcolLen + PSEginRec.length;

} /* end buildPSEGcol */

void buildBMPcol(void)
/*********************

 * Called by the main routine to build a BMP_PHOTO column entry for *
 * the current employee. *
 *
 * This is done by appending the current record of the BMP input file *
 * (from getBMPrec) to the entity associated with blBMP1, the BLOB *
 * locator for the BMP_PHOTO column. *
 *
 * When all BMP input records have been appended to this entity, the *
 * updateBLOBcols function will be invoked to update the BMP_PHOTO *
 * column in the EMP_PHOTO_RESUME table from blBMP1. *

{

 /*****

 * Generate a new BLOB locator that contains the current input *
 * record appended to the current BMP_PHOTO locator *

 EXEC SQL SET :blBMP2 = SUBSTR(:blBMP1,1,LENGTH(:blBMP1))

 || :BMPinRec;

 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("buildBMPcol @ SET LOCATOR #2");
 }

 /*****

 * Regenerate the BMP_PHOTO locator from the updated locator *

 if(status == OK)
 {
 EXEC SQL FREE LOCATOR :blBMP1;
 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("buildBMPcol @ FREE LOCATOR #1");
 }
 }

 if(status == OK)
 {
 EXEC SQL SET :blBMP1 = :blBMP2;
 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("buildBMPcol @ SET LOCATOR #1");
 }
 }

 if(status == OK)
 {
 EXEC SQL FREE LOCATOR :blBMP2;
 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("buildBMPcol @ FREE LOCATOR #2");
 }
 }

 /*****

 * Update length of BMP_PHOTO column *

 if(status == OK)
 BMPcolLen = BMPcolLen + BMPinRec.length;

} /* end buildBMPcol */

void updateBLOBcols(void)
/*********************
```

```

* Called by the main routine to apply the BLOB entities constructed *
* from the PSEGin and BMPin input files by the buildPSEGcol and *
* buildBMPcol functions and pointed to by the blPSEG1 and blBMP1 *
* BLOB locators to the PSEG_PHOTO and BMP_PHOTO columns of the *
* EMP_PHOTO_RESUME_TABLE. *
*****{*}
{
 EXEC SQL UPDATE EMP_PHOTO_RESUME
 SET PSEG_PHOTO = :blPSEG1,
 BMP_PHOTO = :blBMP1
 WHERE EMPNO = :hvEMPNO;

 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("updateBLOBcols @ UPDATE");
 }
} /* end updateBLOBcols */

void commitWorkUnit(void)
/*****
* Called by the main routine to commit the current unit of work, *
* which is composed of a fully-built PSEG entry and a fully-built *
* BMP entry for the current employee. *
*****{/}
{
 EXEC SQL COMMIT;

 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("commitWorkUnit @ COMMIT");
 }
} /* end commitWorkUnit */

void rollbackWorkUnit(void)
/*****
* Called by the main routine to rollback the current unit of work, *
* which is composed of a fully-built PSEG entry and a fully-built *
* BMP entry for the current employee. *
*****{/}
{
 EXEC SQL ROLLBACK;

 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("rollbackWorkUnit @ ROLLBACK");
 }
} /* end rollbackWorkUnit */

void sql_error(char *locmsg)
/*****
* SQL error handler
*****{/}
{
 short int rc; /* DSNTIAR Return code */
 int j,k; /* Loop control */
 static int lrecl = TIAR_LEN; /* Width of message lines */
 /* */

/* print the location message */
*****{/}
printf("*****\n");
printf("*** ERROR: DSN8DLPL DB2 Sample Program\n");
printf("*** Unexpected SQLCODE encountered at location\n");
printf("*** %.68s\n", locmsg);
printf("*** Error detailed below\n");
printf("*** Processing terminated\n");
printf("*****\n");

/* format and print the SQL message */
*****{/}
rc = dsntiar(&sqlca, &error_message, &lrecl);
if(rc == 0)
 for(j=0; j<TIAR_DIM; j++)
 {
 for(k=0; k<TIAR_LEN; k++)

```

```

 putchar(error_message.error_text[j][k]);
 putchar('\n');
 }
else
{
 printf(" *** ERROR: DSNTIAR could not format the message\n");
 printf(" *** SQLCODE is %d\n",SQLCODE);
 printf(" *** SQLERRM is \n");
 for(j=0; j<sqlca.sqlerrml; j++)
 printf("%c", sqlca.sqlerrmc[j]);
 printf("\n");
}
} /* end sql_error */

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

### DSN8DLRV

Solicita al usuario que elija un empleado, después recupera los datos del curriculum vitae para ese empleado de la columna RESUME (CLOB) de la tabla EMP\_PHOTO\_RESUME en un localizador CLOB, utiliza las funciones de manejo de localizador LOB para localizar y dividir elementos de datos y ponerlos en campos para que los visualice ISPF.

```

* Module name = DSN8DLRV (DB2 sample program)
*
* DESCRIPTIVE NAME = Display the resume of a specified employee
*
*
* LICENSED MATERIALS - PROPERTY OF IBM
* 5675-DB2
* (C) COPYRIGHT 1982, 2000 IBM CORP. ALL RIGHTS RESERVED.
*
* STATUS = VERSION 7
*
* Function: Prompts the user to choose an employee, then retrieves
* the resume data for that employee from the RESUME (CLOB)
* column of the EMP_PHOTO_RESUME table into a CLOB locator,
* uses LOB locator-handling functions to locate and break
* out data elements, and puts them in fields for display
* by ISPF.
*
* Notes:
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher
*
* Restrictions:
*
* Module type: C program
* Processor: IBM C/C++ for OS/390 V1R3 or subsequent release
* Module size: See linkedit output
* Attributes: Re-entrant and re-usable
*
* Entry Point: CEESTART (Language Environment entry point)
* Purpose: See Function
* Linkage: Standard MVS program invocation, no parameters
*
* Normal Exit: Return Code = 0000
* - Message: none
*
* Error Exit: Return Code = 0008
* - Message: *** ERROR: DSN8DLRV DB2 Sample Program
* Unexpected SQLCODE encountered
* at location xxxx
* Error detailed below
* Processing terminated
* (DSNTIAR-formatted message here)
*
* - Message: *** ERROR: DSN8DLRV DB2 Sample Program
* No entry in the Employee Photo/
* Resume table for employee with
* empno = xxxxxxx
* Processing terminated
*
* - Message: *** ERROR: DSN8DLRV DB2 Sample Program
*
```

```

*
* No resume data exists in *
* the Employee Photo/Resume table *
* for the employee with empno = *
* xxxxx. *
* Processing terminated *
*
*
* External References:
* - Routines/Services: DSNTIAR, ISPF
* - Data areas : DSNTIAR error_message
* - Control blocks : None
*
*
* Pseudocode:
* DSN8DLRV:
* - Call initISPFvars to establish ISPF variable sharing
* - Do until the user indicates termination
* - Call clearISPFvars to reset the ISPF shared variables
* - Call getEmplNum to request an employee id
* - Call getEmplResume to retrieve the resume
* - Call formatEmplResume to populate the ISPF display panel
* - Call showEmplResume to display the resume
* - Call freeISPFvars to terminate ISPF variable sharing
* End DSN8DLRV
*
* initISPFvars:
* - Establish ISPF variable sharing
* End initISPFvars
*
* clearISPFvars:
* - Set ISPF vars to blank if character type or 0 if numeric
* End clearISPFvars
*
* getEmplNum:
* - prompt user to select an employee whose resume is to be viewed
* End getEmplNum
*
* getEmplResume:
* - Fetch the specified employee's resume from DB2 using a CLOB
* locator
* End getEmplResume
*
* formatEmplResume:
* - call getPersonalData to extract personal data from the resume
* - call getDepartmentData to extract department data
* - call getEducationData to extract education data
* - call getWorkHistoryData to extract work history data
* End formatEmplResume
*
* showEmplResume:
* - Display the ISPF panel with the specified employee's resume
* End showEmplResume
*
* freeISPFvars:
* - Terminate variable sharing with ISPF
* End freeISPFvars
*
* getPersonalData:
* - Parse the employee's name, address, home telephone no.,
* birthdate, sex, marital status, height, and weight into ISPF
* display variables
* End getPersonalData
*
* getDepartmentData:
* - Parse the employee's department number, manager, job position,
* work telephone no., and hire date into ISPF display variables.
* End getDepartmentData
*
* getEducationData:
* - Parse the employee's degree dates, descriptions, and schools
* into ISPF display variables.
* End getEducationData
*
* getWorkHistoryData:
* - Parse the employee's job dates, titles, and descriptions into
* ISPF display variables.
* End getWorkHistoryData
*
* sql_error:
* - call DSNTIAR to format an unexpected SQLCODE.
* End sql_error
*

```

```

* Assumptions: *
* (1) Each employee has exactly 2 entries under "Education" *
* (2) Each employee has exactly 3 entries under "Work History" *
* (3) Each job description consists of a single sentence and that *
* sentence ends with a period and that period is the only *
* period in the sentence. *

/***** C Program Product Libraries *****/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/***** Equates *****/
#define NO 0 /* False */
#define YES 1 /* True */

#define NOT_OK 0 /* Run status indicator: Error */
#define OK 1 /* Run status indicator: Good */

#define TIAR_DIM 10 /* Max no. of DSNTIAR msgs */
#define TIAR_LEN 80 /* Length of DSNTIAR messages */

/***** Global Storage *****/
int keepViewing = YES; /* User status */
int status = OK; /* Run status */

short int ISPFrc; /* For ISPF return code */

/***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

/***** DB2 Message Formatter *****/
struct error_struct { /* DSNTIAR message structure */
 short int error_len;
 char error_text[TIAR_DIM][TIAR_LEN];
} error_message = {TIAR_DIM * (TIAR_LEN)};

#pragma linkage(dsntiar, OS)

extern short int dsntiar(struct sqlca *sqlca,
 struct error_struct *msg,
 int *len);

/***** DB2 Tables *****/
EXEC SQL DECLARE EMP_PHOTO_RESUME TABLE
(
 EMPNO CHAR(06) NOT NULL,
 EMP_ROWID ROWID,
 PSEG_PHOTO BLOB(500K),
 BMP_PHOTO BLOB(100K),
 RESUME CLOB(5K));
;

/***** DB2 Host and Null Indicator Variables *****/
EXEC SQL BEGIN DECLARE SECTION;
 char hvEMPNO[7]; /* host var for emp ser no. */
 long int begSection; /* ptr to beg of resume sec'n */
 char *begField; /* ptr to beg of fld in sec'n */
 long int endSection; /* ptr to end of resume sec'n */
 char *endField; /* ptr to end of fld in sec'n */

 SQL TYPE IS CLOB(5K) hvRESUME; /* host var for RESUME CLOB */
 char *phvRESUME; /* ptr to RESUME CLOB data */
 short int niRESUME = 0; /* indic var for RESUME CLOB */
EXEC SQL END DECLARE SECTION;

/***** DB2 LOB Locator Variables *****/
EXEC SQL BEGIN DECLARE SECTION;
 SQL TYPE IS CLOB_LOCATOR clRESUME; /* CLOB loc for RESUME column */
EXEC SQL END DECLARE SECTION;

/***** ISPF Linkage *****/
#pragma linkage(isplink,OS)

```

```

/********************* ISPF Syntax *****/
char CHAR[9] = "CHAR ";
char DISPLAY[9] = "DISPLAY ";
char VDEFINE[9] = "VDEFINE ";
char VGET[9] = "VGET ";
char VRESET[9] = "VRESET ";

/********************* ISPF Shared Variables *****/
char D8EMNAME[25]; /* employee's name */
char D8EMNUMB[7]; /* employee's serial number */
char D8EMADR1[25]; /* employee's address line 1 */
char D8EMDEPT[5]; /* employee's department */
char D8EMADR2[25]; /* employee's address line 2 */
char D8MGRNAM[22]; /* employee's manager's name */
char D8EMADR3[15]; /* employee's address line 3 */
char D8EMPOSN[22]; /* employee's job position */
char D8EMBORN[19]; /* employee's date of birth */
char D8EMPHON[15]; /* employee's home phone no. */
char D8EMSEX[7]; /* employee's gender */
char D8EMHIRE[11]; /* employee's hire date */
char D8EMHGT[6]; /* employee's height */
char D8EMWGT[9]; /* employee's weight */
char D8EMPMST[9]; /* employee's marital status */
char D8EMEDY1[5]; /* date of most recent degree */
char D8EMEDD1[35]; /* type of most recent degree */
char D8EMEDY2[5]; /* date of previous degree */
char D8EMEDD2[35]; /* type of previous degree */
char D8EMEDI1[35]; /* name of most recent school */
char D8EMEDI2[35]; /* name of previous school */
char D8EMWHD1[17]; /* dates of 1st previous job */
char D8EMWHJ1[63]; /* title of 1st previous job */
char D8EMWHT1[63]; /* descr. of 1st previous job */
char D8EMWHD2[17]; /* dates of 2nd previous job */
char D8EMWHJ2[63]; /* title of 2nd previous job */
char D8EMWHT2[63]; /* descr. of 2nd previous job */
char D8EMWHD3[17]; /* dates of 3rd previous job */
char D8EMWHJ3[63]; /* title of 3rd previous job */
char D8EMWHT3[63]; /* descr. of 3rd previous job */

/********************* Global Functions *****/
int main(void); /* main logic */
void initISPFvars(void); /* establish ISPF vars */
void clearISPFvars(void); /* blank/zero ISPF disp vars */
void getEmplNum(void); /* prompt for employee ser no */
void getEmplResume(void); /* get resume from database */
void formatEmplResume(void); /* build display panel */
void getPersonalData(void); /* get personal data from res */
void getDepartmentData(void); /* get dept data from resume */
void getEducationData(void); /* get educ data from resume */
void getWorkHistoryData(void); /* get job hist from resume */
void showEmplResume(void); /* display the ISPF panel */
void freeISPFvars(void); /* drop ISPF vars */
void sql_error(char *locmsg); /* generate SQL messages */

/********************* main routine *****/
***** main routine ****
***** main routine ****
int main(void)
{
 /* Establish variable sharing with ISPF */
 initISPFvars();

 /* Display employee resumes until user indicates completion */
 keepViewing = YES;
 while(keepViewing == YES)
 {
 clearISPFvars();
 /* prompt user to select employee whose resume is to be viewed */
 getEmplNum();

 if(keepViewing == YES && status == OK)

```

```

{
 /*****
 * retrieve the employee's resume from DB2
 */
 getEmplResume();
 /*****
 * if successful, format the resume on ISPF
 */
 if(status == OK)
 formatEmplResume();
 /*****
 * if successful, display the resume on ISPF
 */
 if(status == OK)
 showEmplResume();
 /*****
 * otherwise, exit this program
 */
 else
 keepViewing = NO;
}

/*
 * Terminate variable sharing with ISPF
 */
freeISPFvars();

} /* end main */

void initISPFvars(void)
/*****
 * Called by the main routine. Establishes variable sharing between
 * ISPF and this program.
 */
{
 ISP弗rc = isplink(VDEFINE, "D8EMNAME", D8EMNAME, CHAR, 24);
 ISP弗rc = isplink(VDEFINE, "D8EMNUMB", D8EMNUMB, CHAR, 6);
 ISP弗rc = isplink(VDEFINE, "D8EMADR1", D8EMADR1, CHAR, 24);
 ISP弗rc = isplink(VDEFINE, "D8EMDEPT", D8EMDEPT, CHAR, 4);
 ISP弗rc = isplink(VDEFINE, "D8EMADR2", D8EMADR2, CHAR, 24);
 ISP弗rc = isplink(VDEFINE, "D8MGRNAM", D8MGRNAM, CHAR, 21);
 ISP弗rc = isplink(VDEFINE, "D8EMADR3", D8EMADR3, CHAR, 14);
 ISP弗rc = isplink(VDEFINE, "D8EMPOSN", D8EMPOSN, CHAR, 21);
 ISP弗rc = isplink(VDEFINE, "D8EMBORN", D8EMBORN, CHAR, 18);
 ISP弗rc = isplink(VDEFINE, "D8EMPHON", D8EMPHON, CHAR, 14);
 ISP弗rc = isplink(VDEFINE, "D8EMSEX ", D8EMSEX , CHAR, 6);
 ISP弗rc = isplink(VDEFINE, "D8EMHIRE", D8EMHIRE, CHAR, 10);
 ISP弗rc = isplink(VDEFINE, "D8EMHGHT ", D8EMHGHT , CHAR, 5);
 ISP弗rc = isplink(VDEFINE, "D8EMWGT ", D8EMWGT , CHAR, 8);
 ISP弗rc = isplink(VDEFINE, "D8EMPMST", D8EMPMST, CHAR, 8);
 ISP弗rc = isplink(VDEFINE, "D8EMEDY1", D8EMEDY1, CHAR, 4);
 ISP弗rc = isplink(VDEFINE, "D8EMEDD1", D8EMEDD1, CHAR, 34);
 ISP弗rc = isplink(VDEFINE, "D8EMEDY2", D8EMEDY2, CHAR, 4);
 ISP弗rc = isplink(VDEFINE, "D8EMEDD2", D8EMEDD2, CHAR, 34);
 ISP弗rc = isplink(VDEFINE, "D8EMEDI1", D8EMEDI1, CHAR, 34);
 ISP弗rc = isplink(VDEFINE, "D8EMEDI2", D8EMEDI2, CHAR, 34);
 ISP弗rc = isplink(VDEFINE, "D8EMWHD1", D8EMWHD1, CHAR, 16);
 ISP弗rc = isplink(VDEFINE, "D8EMWHD1", D8EMWHD1, CHAR, 62);
 ISP弗rc = isplink(VDEFINE, "D8EMWHT1", D8EMWHT1, CHAR, 62);
 ISP弗rc = isplink(VDEFINE, "D8EMWHD2", D8EMWHD2, CHAR, 16);
 ISP弗rc = isplink(VDEFINE, "D8EMWHD2", D8EMWHD2, CHAR, 62);
 ISP弗rc = isplink(VDEFINE, "D8EMWHT2", D8EMWHT2, CHAR, 62);
 ISP弗rc = isplink(VDEFINE, "D8EMWHD3", D8EMWHD3, CHAR, 16);
 ISP弗rc = isplink(VDEFINE, "D8EMWHD3", D8EMWHD3, CHAR, 62);
 ISP弗rc = isplink(VDEFINE, "D8EMWHT3", D8EMWHT3, CHAR, 62);
}

/* end initISPFvars */

void clearISPFvars(void)
/*****
 * Called by the main routine. Blanks out the ISPF shared variables.
 */
{
 memset(D8EMNAME, 0, 25);
 memset(D8EMNUMB, 0, 7);
 memset(D8EMADR1, 0, 25);
 memset(D8EMDEPT, 0, 5);
 memset(D8EMADR2, 0, 25);
 memset(D8MGRNAM, 0, 22);
 memset(D8EMADR3, 0, 15);
}

```

```

memset(D8EMPOSN, 0, 22);
memset(D8EMBORN, 0, 19);
memset(D8EMPHON, 0, 15);
memset(D8EMSEX , 0, 7);
memset(D8EMHIRE, 0, 11);
memset(D8EMHGT , 0, 9);
memset(D8EMWGT , 0, 8);
memset(D8EMPMST, 0, 9);
memset(D8EMEDY1, 0, 5);
memset(D8EMEDD1, 0, 35);
memset(D8EMEDY2, 0, 5);
memset(D8EMEDD2, 0, 35);
memset(D8EMEDI1, 0, 35);
memset(D8EMEDI2, 0, 35);
memset(D8EMWHD1, 0, 17);
memset(D8EMWHJ1, 0, 63);
memset(D8EMWHT1, 0, 63);
memset(D8EMWHD2, 0, 17);
memset(D8EMWHJ2, 0, 63);
memset(D8EMWHT2, 0, 63);
memset(D8EMWHD3, 0, 17);
memset(D8EMWHJ3, 0, 63);
memset(D8EMWHT3, 0, 63);
} /* end clearISPFvars */

void getEmplNum(void)
/***
* Called by the main routine. Displays an ISPF panels to prompt the *
* user to select an employee whose resume is to be displayed. *
***/
{
 /*****
 * Display the prompt panel
 */
 ISPFrc = isplink("DISPLAY ", "DSN8SSE ");
 if(ISPFrc != 0)
 keepViewing = NO;

 /*****
 * Save off the value of the ISPF shared variable
 */
 strcpy(hvEMPNO,D8EMNUMB);

} /* end getEmplNum */

void getEmplResume(void)
/***
* Called by the main routine. Extracts a specified employee's *
* resume data from a CLOB column in the sample EMP_PHOTO_RESUME *
* table to a CLOB locator.
***/
{
 /*****
 * Establish a CLOB locator on the resume of the specified empno *
 */
 EXEC SQL SELECT RESUME
 INTO :clRESUME
 FROM EMP_PHOTO_RESUME
 WHERE EMPNO = :hvEMPNO;

 if(SQLCODE == 100)
 {
 status = NOT_OK;
 printf("*****\n");
 printf("*** ERROR: DSN8DLRV DB2 Sample Program\n");
 printf("*** No entry in the Employee Photo/Resume\n");
 printf("*** table for employee with empno = %s\n",
 hvEMPNO);
 printf("*** Processing terminated\n");
 printf("*****\n");
 }
 else if(SQLCODE == -305)
 {
 status = NOT_OK;
 printf("*****\n");
 printf("*** ERROR: DSN8DLRV DB2 Sample Program\n");
 printf("*** No resume data exists in the\n");
 printf("*** Employee Photo/Resume table for the\n");
 printf("*** employee with empno = %s\n",
 hvEMPNO);
 }
}

```

```

 printf("*** Processing terminated\n");
 printf("*****\n");
 }
else if(SQLCODE != 0)
{
 status = NOT_OK;
 sql_error("getEmplResume @ SELECT");
}

} /* end getEmplResume */

void formatEmplResume(void)
/***
* Called by the main routine. Calls routines to parse out the *
* contents of the resume into ISPF-shared variables. *
***/
{
 /***
 * Get the employee's name, address, and other personal information *
 ***/
 getPersonalData();

 /***
 * Get the employee's department no., manager, and other dept data *
 ***/
 if(status == OK)
 getDepartmentData();

 /***
 * Get the employee's education data *
 ***/
 if(status == OK)
 getEducationData();

 /***
 * Get the employee's employment history *
 ***/
 if(status == OK)
 getWorkHistoryData();

 /***
 * Free the CLOB locator for the resume *
 ***/
 if(status == OK)
 {
 EXEC SQL FREE LOCATOR :clRESUME;
 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("formatEmplResume @ FREE LOCATOR");
 }
 }

} /* end formatEmplResume */

void getPersonalData(void)
/***
* Called by the formatEmplResume routine to parse the CLOB locator *
* data for the employee's name, address, home telephone no., birth- *
* date, sex, marital status, height, and weight into ISPF variables. *
***/
{
 /***
 * Extract the Personal Data section from the CLOB locator *
 ***/
 EXEC SQL SET :begSection /* locate start of pers. data */
 = POSSTR(:clRESUME, 'Resume:');
 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("getPersonalData @ POSSTR 1");
 }
 if(status == OK)
 {
 EXEC SQL SET :endSection /* locate start of dept. data */
 = POSSTR(:clRESUME, 'Department Information');
 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("getPersonalData @ POSSTR 2");
 }
 }
}

```

```

 }

 if(status == OK)
 {
 EXEC SQL SET :hvRESUME /* extract what's in between */
 = SUBSTR(:clRESUME, :begSection, :endSection-:begSection);
 if(SQLCODE == 0)
 hvRESUME.data[hvRESUME.length] = '\0';
 else
 {
 status = NOT_OK;
 sql_error("getPersonalData @ SUBSTR");
 }
 }

/***** Get the employee's name *****/
* Get the employee's name
*****/if(status == OK)
{
 phvRESUME = &hvRESUME.data[0]; /* set pointer to the data */
 begField /* find Resume: label */
 = strstr(phvRESUME, " Resume: ");
 begField = begField + 11; /* skip past label */
 endField /* find Personal Inf... label */
 = strstr(phvRESUME, " Personal Information ");
 strncpy(D8EMNAME, /* get name from in between */
 begField,
 endField - begField);
}

/***** Get the employee's street address *****/
* Get the employee's street address
*****/if(status == OK)
{
 begField /* find Address: label */
 = strstr(phvRESUME, " Address: ");
 begField = begField + 22; /* skip past label */
 endField /* find end of street addr */
 = strstr(phvRESUME, " ");
 strncpy(D8EMADR1, /* get addr from in between */
 begField,
 endField - begField);
}

/***** Get the employee's city, state, and zipcode *****/
* Get the employee's city, state, and zipcode
*****/if(status == OK)
{
 begField = endField + 22; /* set loc to city/st/zip dat */
 endField /* find end of ciy/st/zip */
 = strstr(phvRESUME, " Phone: ");
 strncpy(D8EMADR2, /* get data from in between */
 begField,
 endField - begField);
}

/***** Get the employee's home telephone number *****/
* Get the employee's home telephone number
*****/if(status == OK)
{
 begField = endField + 22; /* set loc to home phone data */
 endField /* find end of home phone no. */
 = strstr(phvRESUME, " Birthdate: ");
 strncpy(D8EMADR3, /* get phone# from in between */
 begField,
 endField - begField);
}

/***** Get the employee's birthdate *****/
* Get the employee's birthdate
*****/if(status == OK)
{
 begField = endField + 22; /* set loc to birthdate data */
 endField /* find end of birthdate data */
 = strstr(phvRESUME, " Sex: ");
 strncpy(D8EMBORN, /* get birthdate from in betw */
 begField,
 endField - begField);
}

/***** Get the employee's sex *****/
* Get the employee's sex
*****/

```

```

if(status == OK)
{
 begField = endField + 22; /* set loc to sex data */
 endField /* find end of sex data */
 = strstr(phvRESUME, " Marital Status: ");
 strncpy(D8EMSEX, /* get sex data from in betw */
 begField,
 endField - begField);
}
/***
* Get the employee's marital status
***** */
if(status == OK)
{
 begField = endField + 22; /* set loc to marital status */
 endField /* find end of marital stat. */
 = strstr(phvRESUME, " Height: ");
 strncpy(D8EMPMST, /* get mar stat from in betw */
 begField,
 endField - begField);
}
/***
* Get the employee's height
***** */
if(status == OK)
{
 begField = endField + 22; /* set loc to height data */
 endField /* find end of height data */
 = strstr(phvRESUME, " Weight: ");
 strncpy(D8EMHGT, /* get height from in between */
 begField,
 endField - begField);
}
/***
* Get the employee's weight
***** */
if(status == OK)
{
 begField = endField + 22; /* set loc to weight data */
 strcpy(D8EMWGT, /* weight is at end of string */
 begField);
}
} /* end getPersonalData */

void getDepartmentData(void)
/***
* Called by the formatEmp1Resume routine to parse the CLOB locator
* data for the employee's department number, manager, job position,
* work telephone no., and hire date into ISPF variables.
***** */
{
 /* Extract the Department Data section from the CLOB locator */
 begSection = endSection; /* Locate start of Dept data */
 EXEC SQL SET :endSection /* Locate start of Educ data */
 = POSSTR(:clRESUME, ' Education');
 if(SQLCODE != 0)
 {
 status = NOT_OK;
 sql_error("getDepartmentData @ POSSTR");
 }
 if(status == OK)
 {
 EXEC SQL SET :hvRESUME /* extract what's in between */
 = SUBSTR(:clRESUME, :begSection, :endSection-:begSection);
 if(SQLCODE == 0)
 hvRESUME.data[hvRESUME.length] = '\0';
 else
 {
 status = NOT_OK;
 sql_error("getDepartmentData @ SUBSTR");
 }
 }
 /* Get the employee's department number */
 if(status == OK)
 {
 phvRESUME = &hvRESUME.data[0]; /* set pointer to the data */
 begField /* find Dept Number: label */

```

```

 = strstr(phvRESUME, " Dept Number: ");
begField = begField + 22; /* skip past label */
endField /* find end of dept. no. */
 = strstr(phvRESUME, " Manager: ");
strncpy(D8EMDEPT, /* get dept# from in between */
 begField,
 endField - begField);
}
/***
* Get the employee's manager's name
***/
if(status == OK)
{
 begField = endField + 22; /* set loc to manager data */
endField /* find end of manager */
 = strstr(phvRESUME, " Position: ");
strncpy(D8MGRNAM, /* get mgr name from in betw */
 begField,
 endField - begField);
}
/***
* Get the employee's job position
***/
if(status == OK)
{
 begField = endField + 22; /* set loc to position data */
phvRESUME = begField; /* skip ahead in buffer */
endField /* find end of position data */
 = strstr(phvRESUME, " Phone: ");
strncpy(D8EMPOSN, /* get position from in betw */
 begField,
 endField - begField);
}
/***
* Get the employee's work telephone number
***/
if(status == OK)
{
 begField = endField + 22; /* set loc to work phone data */
endField /* find end of work phone no. */
 = strstr(phvRESUME, " Hire Date: ");
strncpy(D8EMPHON, /* get work ph# from in betw */
 begField,
 endField - begField);
}
/***
* Get the employee's hire date
***/
if(status == OK)
{
 begField = endField + 22; /* set loc to hire date data */
strcpy(D8EMHIRE, /* hire data is at end of str */
 begField);
}
} /* end getDepartmentData */

void getEducationData(void)
/***
* Called by the formatEmplResume routine to parse the CLOB locator
* data for the employee's degree dates, descriptions, and schools
* into ISPF variables.
***/
{
/* Extract the Education Data section from the CLOB locator */
begSection = endSection; /* Locate start of Educ data */
EXEC SQL SET :endSection /* Locate start of Work Hist */
 = POSSTR(:clRESUME, ' Work History ');
if(SQLCODE != 0)
{
 status = NOT_OK;
 sql_error("getEducationData @ POSSTR");
}
if(status == OK)
{
 EXEC SQL SET :hvRESUME /* extract what's in between */
 = SUBSTR(:clRESUME, :begSection, :endSection-:begSection);
 if(SQLCODE == 0)
 hvRESUME.data[hvRESUME.length] = '\0';
 else
}
}

```

```

 {
 status = NOT_OK;
 sql_error("getEducationData @ SUBSTR");
 }
 }
/****** */
* Get year and description of employee's most recent degree *
***** */
if(status == OK)
{
 phvRESUME = &hvRESUME.data[0]; /* set pointer to the data */
 begField /* find Education label */
 = strstr(phvRESUME, " Education ");
 begField = begField + 16; /* skip past label */
 endField /* find end of dept. no. */
 = strstr(phvRESUME, " ");
 strncpy(D8EMEDY1, /* get dept# from in between */
 begField,
 endField - begField);
 begField = endField + 16; /* set loc to degree descript */
 endField /* find end of deg descr data */
 = strstr(phvRESUME, " ");
 strncpy(D8EMEDD1, /* get deg descr from in betw */
 begField,
 endField - begField);
}
/****** */
* Get institution that granted employee's most recent degree *
***** */
if(status == OK)
{
 begField = endField + 22; /* set loc to inst name data */
 phvRESUME = begField; /* point to beginning */
 endField /* find end of inst name data */
 = strstr(phvRESUME, " ");
 strncpy(D8EMEDI1, /* get inst name from in betw */
 begField,
 endField - begField);
}
/****** */
* Get year and description of employee's previous degree *
***** */
if(status == OK)
{
 begField = endField + 3; /* set loc to grad year data */
 endField /* find end of grad year data */
 = strstr(phvRESUME, " ");
 strncpy(D8EMEDY2, /* get hire data from in betw */
 begField,
 endField - begField);
 begField = endField + 16; /* set loc to degree descript */
 endField /* find end of deg descr data */
 = strstr(phvRESUME, " ");
 strncpy(D8EMEDD2, /* get deg descr from in betw */
 begField,
 endField - begField);
}
/****** */
* Get institution that granted employee's previous degree *
***** */
if(status == OK)
{
 begField = endField + 22; /* set loc to inst name data */
 phvRESUME = begField; /* reset starting point */
 strcpy(D8EMEDI2, /* inst name is at end of str */
 begField);
}
} /* end getEducationData */

void getWorkHistoryData(void)
/****** */
* Called by the formatEmplResume routine to parse the CLOB locator *
* data for the employee's job dates, titles, and descriptions into *
* ISPF variables. *
***** */
{
 /****** */
 * Extract the Work History Data section from the CLOB locator *
***** */
 begSection = endSection; /* Locate start of Work Hist */
 EXEC SQL SET :endSection /* Locate start of Interests */
}

```

```

 = POSSTR(:clRESUME, ' Interests ');
if(SQLCODE != 0)
{
 status = NOT_OK;
 sql_error("getWorkHistoryData @ POSSTR");
}
if(status == OK)
{
 EXEC SQL SET :hvRESUME /* extract what's in between */
 = SUBSTR(:clRESUME, :begSection, :endSection-:begSection);
if(SQLCODE == 0)
 hvRESUME.data[hvRESUME.length] = '\0';
else
{
 status = NOT_OK;
 sql_error("getWorkHistoryData @ SUBSTR");
}
}
/***
* Get dates and title of employee's most recent job

if(status == OK)
{
 phvRESUME = &hvRESUME.data[0]; /* set pointer to the data */
 begField /* find Work History label */
 = strstr(phvRESUME, " Work History ");
 begField = begField + 19; /* set loc to job 1 dates */
 phvRESUME = begField; /* reset starting point */
 strncpy(D8EMWHD1, /* job 1 dates, next 15 bytes */
 begField,
 15);
 begField = begField + 20; /* set loc to job 1 title */
 endField /* find end of job 1 title */
 = strstr(phvRESUME, " "
);
 strncpy(D8EMWHT1, /* get job 1 title from betw */
 begField,
 endField - begField);
}
/***
* Get description of employee's most recent job

if(status == OK)
{
 begField = endField + 22; /* set loc to job 1 descr. */
 phvRESUME = begField; /* reset starting point */
 endField /* find end of job 1 descr. */
 = strstr(phvRESUME, ". ");
if(endField - begField < 62) /* job 1 descr has 1 part */
 strncpy(D8EMWHJ1, /* get job 1 descr from betw */
 begField,
 endField - begField);
else /* job 1 descr has 2 parts */
{
 endField /* find 1st part of job descr */
 = strstr(phvRESUME, " "
);
 strncpy(D8EMWHJ1, /* get job 1 descr from betw */
 begField,
 endField - begField);
 begField = endField + 22; /* set loc to 2nd part job des*/
 endField /* find end of job 1 descr. */
 = strstr(phvRESUME, ". ");
 strncat(D8EMWHJ1, /* get rest of job 1 descr. */
 begField-1,
 endField - (begField-1));
}
}
/***
* Get dates and title of employee's previous job

if(status == OK)
{
 begField = endField + 4; /* set loc to job 2 dates */
 phvRESUME = begField; /* reset starting point */
 strncpy(D8EMWHD2, /* job 2 dates, next 15 bytes */
 begField,
 15);
 begField = begField + 20; /* set loc to job 2 title */
 endField /* find end of job 2 title */
 = strstr(phvRESUME, " "
);
 strncpy(D8EMWHT2, /* get job 2 title from betw */
 begField,
 endField - begField);
}

```

```

 }

/*
 * Get description of employee's previous job
 */
if(status == OK)
{
 begField = endField + 22; /* set loc to job 2 descr. */
 phvRESUME = begField; /* reset starting point */
 endField /* find end of job 2 descr. */
 = strstr(phvRESUME,". ");
 if(endField - begField < 62) /* job 2 descr has 1 part */
 strncpy(D8EMWHJ2, /* get job 2 title from betw */
 begField,
 endField - begField);
 else /* job 2 descr has 2 parts */
 {
 endField /* find 1st part of job descr */
 = strstr(phvRESUME," ");
 strncpy(D8EMWHJ2, /* get job 2 descr from betw */
 begField,
 endField - begField);
 begField = endField + 22; /* set loc to 2nd part job des*/
 endField /* find end of job 2 descr. */
 = strstr(phvRESUME,". ");
 strncat(D8EMWHJ2, /* get rest of job 2 descr. */
 begField-1,
 endField - (begField-1));
 }
}

/*
 * Get dates and title of employee's other previous job
 */
if(status == OK)
{
 begField = endField + 4; /* set loc to job 3 dates */
 phvRESUME = begField; /* reset starting point */
 strncpy(D8EMWHD3, /* job 3 dates, next 15 bytes */
 begField,
 15);
 begField = begField + 20; /* set loc to job 3 title */
 endField /* find end of job 3 title */
 = strstr(phvRESUME," ");
 strncpy(D8EMWHT3, /* get job 3 title from betw */
 begField,
 endField - begField);
}

/*
 * Get description of employee's other previous job
 */
if(status == OK)
{
 begField = endField + 22; /* set loc to job 3 descr. */
 phvRESUME = begField; /* reset starting point */
 begField = phvRESUME; /* reset starting point */
 endField /* find end of job 3 descr. */
 = strstr(phvRESUME,". ");
 if(endField - begField < 62) /* job 3 descr has 1 part */
 strncpy(D8EMWHJ3, /* get job 3 title from betw */
 begField,
 endField - begField);
 else /* job 3 descr has 2 parts */
 {
 endField /* find 1st part of job descr */
 = strstr(phvRESUME," ");
 strncpy(D8EMWHJ3, /* get job 3 descr from betw */
 begField,
 endField - begField);
 begField = endField + 22; /* set loc to 2nd part job des*/
 endField /* find end of job 3 descr. */
 = strstr(phvRESUME,". ");
 strncat(D8EMWHJ3, /* get rest of job 3 descr. */
 begField-1,
 endField - (begField-1));
 }
}

/* end getWorkHistoryData */

void showEmplResume(void)
/*
 * Called by the main routine. Displays an ISPF panel that is for-
 * matted with the resume data for the employee specified.
 */

```

```

{
 ISPFRc = isplink("DISPLAY ", "DSN8SSR ");
} /* end showEmplResume */

void freeISPFvars(void)
/*****
 * Called by the main routine. Frees the ISPF variables that were *
 * established for running this application. *
 *****/
{
 ISPFRc = isplink(VRESET);
} /* end freeISPFvars */

void sql_error(char *locmsg)
/*****
 * SQL error handler
 *****/
{
 short int rc; /* DSNTIAR Return code */
 int j,k; /* Loop control */
 static int lrecl = TIAR_LEN; /* Width of message lines */
 /* print the location message */
 printf("*****\n");
 printf("*** ERROR: DSN8DLRV DB2 Sample Program\n");
 printf("*** Unexpected SQLCODE encountered at location\n");
 printf("*** %.68s\n", locmsg);
 printf("*** Error detailed below\n");
 printf("*** Processing terminated\n");
 printf("*****\n");

 /* format and print the SQL message */
 rc = dsntiar(&sqlca, &error_message, &lrecl);
 if(rc == 0)
 for(j=0; j<TIAR_DIM; j++)
 {
 for(k=0; k<TIAR_LEN; k++)
 putchar(error_message.error_text[j][k]);
 putchar('\n');
 }
 else
 {
 printf(" *** ERROR: DSNTIAR could not format the message\n");
 printf(" *** SQLCODE is %d\n",SQLCODE);
 printf(" *** SQLERRM is \n");
 for(j=0; j<sqlca.sqlerrml; j++)
 printf("%c", sqlca.sqlerrmc[j]);
 printf("\n");
 }
} /* end sql_error */

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSN8DLPV

Solicita al usuario que elija un empleado, después recupera la imagen de foto PSEG para ese empleado de la columna PSEG\_- PHOTO de la tabla EMP\_PHOTO\_RESUME y la pasa a GDDM para formatearla y mostrarla.

```

 * Module name = DSN8DLPV (DB2 sample program) *
 * DEScriptive NAME = Display PSEG photo image of a specified employee*

```

```

*
*
* LICENSED MATERIALS - PROPERTY OF IBM
* 5655-DB2
* (C) COPYRIGHT 1997 IBM CORP. ALL RIGHTS RESERVED.
*
* STATUS = VERSION 6
*
* Function: Prompts the user to choose an employee, then retrieves
* the PSEG photo image for that employee from the PSEG-
* PHOTO column of the EMP_PHOTO_RESUME table and passes it
* to GDDM for formatting and display.
*
* Notes:
* Dependencies: Requires IBM C/C++ for OS/390 V1R3 or higher
* Requires IBM Graphical Data Display Manager (GDDM)
* V3R1 or higher
*
* Restrictions:
*
* Module type: C program
* Processor: IBM C/C++ for OS/390 V1R3 or subsequent release
* Module size: See linkedit output
* Attributes: Re-entrant and re-usable
*
* Entry Point: CEESTART (Language Environment entry point)
* Purpose: See Function
* Linkage: Standard MVS program invocation, no parameters
*
* Normal Exit: Return Code = 0000
* - Message: none
*
* Error Exit: Return Code = 0008
* - Message: *** ERROR: DSN8DLPV DB2 Sample Program
* Unexpected SQLCODE encountered
* at location xxx
* Error detailed below
* Processing terminated
* (DSNTIAR-formatted message here)*
*
* - Message: *** ERROR: DSN8DLPV DB2 Sample Program
* No entry in the Employee Photo/
* Resume table for employee with
* empno = xxxxxxx
* Processing terminated
*
* - Message: *** ERROR: DSN8DLPV DB2 Sample Program
* No PSEG photo image exists in
* the Employee Photo/Resume table
* for the employee with empno =
* xxxxxxx.
* Processing terminated
*
* External References:
* - Routines/Services: DSNTIAR, GDDM, ISPF
* - Data areas : DSNTIAR error_message
* - Control blocks : None
*
* Pseudocode:
* DSN8DLPV:
* - Do until the user indicates termination
* - Call getEmplNum to request an employee id
* - Call getEmplPhoto to retrieve the PSEG photo image
* - Call showEmplPhoto to display the photo
* End DSN8DLRV
*
* getEmplNum:
* - prompt user to select an employee whose photo is to be viewed
*
* getEmplPhoto:
* - Fetch the specified employee's PSEG photo image from DB2
* - call sql_error for unexpected SQLCODEs
* End getEmplPhoto:
*
* showEmplPhoto:
* - Use GDDM calls to format and display the PSEG photo image
*
* sql_error:
* - call DSNTIAR to format the unexpected SQLCODE.
*

```

```

***** C Program Product Libraries *****/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

***** GDDM Program Product Libraries (Reentrant Versions) *****/
#include <ADMUCIRA>
#include <ADMTSTRC>
#include <ADMUCIRF>
#include <ADMUCIRG>
#include <ADMUCIRI>

***** Equates *****/
#define NO 0 /* Boolean: False */
#define YES 1 /* Boolean: True */

#define NOT_OK 0 /* Run status indicator: Error */
#define OK 1 /* Run status indicator: Good */

#define TIAR_DIM 10 /* Max no. of DSNTIAR msgs */
#define TIAR_LEN 80 /* Length of DSNTIAR messages */

***** Global Storage *****/
int keepViewing = YES; /* */
int status = OK; /* run status */

short int ISPFrc; /* For ISPF return code */

***** DB2 SQL Communication Area *****/
EXEC SQL INCLUDE SQLCA;

***** DB2 Message Formatter *****/
struct error_struct { /* DSNTIAR message structure */
 short int error_len;
 char error_text[TIAR_DIM][TIAR_LEN];
} error_message = {TIAR_DIM * (TIAR_LEN)};

#pragma linkage(dsntiar, OS)

extern short int dsntiar(struct sqlca *sqlca,
 struct error_struct *msg,
 int *len);

***** DB2 Tables *****/
EXEC SQL DECLARE EMP_PHOTO_RESUME TABLE
(
 EMPNO CHAR(06) NOT NULL,
 EMP_ROWID ROWID,
 PSEG_PHOTO BLOB(500K),
 BMP_PHOTO BLOB(100K),
 RESUME CLOB(5K));
;

***** DB2 Host and Null Indicator Variables *****/
EXEC SQL BEGIN DECLARE SECTION;
char hvEMPNO[7]; /* */
SQL TYPE IS BLOB(500K) hvPSEG_PHOTO; /* */
short int n1PSEG_PHOTO = 0; /* */
EXEC SQL END DECLARE SECTION;

***** GDDM Variables *****/
union{ Admaab AABtag;
 char AABstr[8];
 } AAB;
int appl_id; /* id for application image */

int ih_pixels = 800; /* -horiz size in # of pixels */
int iv_pixels = 750; /* -vert size in # of pixels */
int iim_type = 1; /* -pixel type (1=bi-level) */
int ires_type = 1; /* -defined resolution indic. */
int ires_unit = 0; /* -resolut'n units (0=inches) */
float ih_res = 100.00; /* -horizontal resolution */
float iv_res = 100.00; /* -vertical resolution */

```

```

int PSEGformat = -3; /* PSEG/GDDM convers'n factors*/
int PSEGcompression = 4; /* indicates PSEG format */
int attype; /* type of attn/interrupt key */
int attval; /* value of attn/interrupt key */
int count; /* number of fields modified */

/****** ISPF Linkage *****/
#pragma linkage(isplink,OS)

/****** ISPF Syntax *****/
char CHAR[9] = "CHAR ";
char CONTROL[9] = "CONTROL ";
char DISPLAY[9] = "DISPLAY ";
char LINE[9] = "LINE ";
char VDEFINE[9] = "VDEFINE ";
char VGET[9] = "VGET ";
char VRESET[9] = "VRESET ";

/****** ISPF Shared Variables *****/
char D8EMNUMB[7]; /* */

/****** Global Functions *****/
int main(void); /* */
void getEmplNum(void); /* */
void getEmplPhoto(void); /* */
void showEmplPhoto(void); /* */
void sql_error(char *locmsg); /* */

/****** main routine *****/
int main(void)
{
 /* Display employee photos until user indicates completion */
 keepViewing = YES;
 while(keepViewing == YES)
 {
 /* prompt user to select an employee whose photo is to be viewed*/
 getEmplNum();

 if(keepViewing == YES && status == OK)
 {
 /* extract the employee's PSEG photo image from BLOB storage*/
 getEmplPhoto();
 /* if okay, convert PSEG image to GDDM format and display it*/
 if(status == OK)
 showEmplPhoto();
 /* otherwise, exit this program */
 else
 keepViewing = NO;
 }
 }

} /* end main */

void getEmplNum(void)
/* Called by the main routine. Displays an ISPF panels to prompt the */
/* user to select an employee whose photo image is to be displayed. */
{
 /* Share the ISPF var having the employee number */
}

```

```

*****+
ISPFrc = isplink(VDEFINE, "D8EMNUMB", D8EMNUMB, CHAR, 6);
strcpy(D8EMNUMB," ");

/*****+
* Display the prompt panel
*****+
ISPFrc = isplink(DISPLAY,"DSN8SSE ");
if(ISPFrc != 0)
 keepViewing = NO;

/*****+
* Save off the value of the ISPF shared variable
*****+
strcpy(hvEMPNO,D8EMNUMB);

/*****+
* And release it
*****+
ISPFrc = isplink(VRESET);
} /* end getEmplNum */

void getEmplPhoto(void)
/*****+
* Called by the main routine. Extracts a specified employee's PSEG *
* photo image from a BLOB column in the sample EMP_PHOTO_RESUME. *
* This image will be converted to GDDM format and displayed by the *
* rotuien showEmplPhoto. *
*****+
{
 EXEC SQL SELECT PSEG_PHOTO
 INTO :hvPSEG_PHOTO
 FROM EMP_PHOTO_RESUME
 WHERE EMPNO = :hvEMPNO;

 if(SQLCODE == 0)
 hvPSEG_PHOTO.data[hvPSEG_PHOTO.length] = '\n';
 else if(SQLCODE == 100)
 {
 status = NOT_OK;
 printf("*****\n");
 printf("*** ERROR: DSN8DLPV DB2 Sample Program\n");
 printf("*** No entry in the Employee Photo/Resume\n");
 printf("*** table for employee with empno = %s\n",
 hvEMPNO);
 printf("*** Processing terminated\n");
 printf("*****\n");
 }
 else if(SQLCODE == -305)
 {
 status = NOT_OK;
 printf("*****\n");
 printf("*** ERROR: DSN8DLPV DB2 Sample Program\n");
 printf("*** No PSEG photo image exists in the\n");
 printf("*** Employee Photo/Resume table for the\n");
 printf("*** employee with empno = %s\n",
 hvEMPNO);
 printf("*** Processing terminated\n");
 printf("*****\n");
 }
 else
 {
 status = NOT_OK;
 sql_error("getEmplPhoto @1");
 }
} /* end getEmplPhoto */

void showEmplPhoto(void)
/*****+
* Called by the main routine. Converts the employee's photo from *
* PSEG format to a GDDM image and then displays it until the user *
* depresses any PF key or the <enter> key. *
*****+
{
 /*****+
 * Signal ISPF to full-screen refresh when GDDM session terminates *
 *****+
 isplink(CONTROL,DISPLAY,LINE);
}

```

```

*****+
* Initialize GDDM
*****+
fsinit(AAB.AABstr); /* GDDM anchor block */

*****+
* Obtain a GDDM application image id
*****+
imagid(AAB.AABstr, /* GDDM anchor block */
 &appl_id); /* application id for image */

*****+
* Create a GDDM application image to receive the employee photo
*****+
imacrt(AAB.AABstr, /* GDDM anchor block */
 appl_id, /* target: application image */
 ih_pixels, /* horiz size in # of pixels */
 iv_pixels, /* vert size in # of pixels */
 iim_type, /* pixel type (1=bi-level) */
 ires_type, /* defined resolution indic. */
 ires_unit, /* resolut'n units (0=inches) */
 ih_res, /* horizontal resolution */
 iv_res); /* vertical resolution */

*****+
* Set up conversion of photo from PSEG format to GDDM format
*****+
imaps(AAB.AABstr, /* GDDM anchor block */
 appl_id, /* target: application image */
 0, /* GDDM proj. id (0=identity) */
 PSEGformat, /* source format (PSEG) */
 PSEGcompression); /* source compression (3800) */

*****+
* Perform conversion
*****+
imapt(AAB.AABstr, /* GDDM anchor block */
 appl_id, /* target: application image */
 hvPSEG_PHOTO.length, /* source length */
 hvPSEG_PHOTO.data); /* source: employee PSEG photo */

*****+
* Terminate conversion
*****+
imape(AAB.AABstr, /* GDDM anchor block */
 appl_id); /* target: application image */

*****+
* Transfer the GDDM application image to the display
*****+
imxfer(AAB.AABstr, /* GDDM anchor block */
 appl_id, /* source: application image */
 0, /* target: 0=display */
 0); /* GDDM proj. id (0=identity) */

*****+
* Disable user updates to the image on the display
*****+
fsenab(AAB.AABstr, /* GDDM anchor block */
 1, /* type of input (1=alphanum) */
 0); /* type of control (0=disable)*/

fsenab(AAB.AABstr,
 2, /* type of input (2=graphic) */
 0); /* type of control (0=disable)*/

fsenab(AAB.AABstr,
 3, /* type of input (3=image) */
 0); /* type of control (0=disable)*/

*****+
* Display the image until attn or interrupt key depressed
*****+
asread(AAB.AABstr, /* GDDM anchor block */
 &attype, /* type of attn/interrupt key */
 &attval, /* value of attn/interrupt key */
 &count); /* number of fields modified */

*****+
* Delete the GDDM application image
*****+

```

```

imadel(AAB.AABstr, /* GDDM anchor block */
 appl_id); /* target: application image */

/* **** Terminate GDDM ****
fsterm(AAB.AABstr); /* GDDM anchor block */

} /* end showEmplPhoto */

void sql_error(char *locmsg)
/* **** SQL error handler ****
{
 short int rc; /* DSNTIAR Return code */
 int j,k; /* Loop control */
 static int lrecl = TIAR_LEN; /* Width of message lines */

/* **** print the location message ****
printf("*****\n");
printf("*** ERROR: DSN8DLPV DB2 Sample Program\n");
printf("*** Unexpected SQLCODE encountered at location\n");
printf("*** %.68s\n", locmsg);
printf("*** Error detailed below\n");
printf("*** Processing terminated\n");
printf("*****\n");

/* **** format and print the SQL message ****
rc = dsntiar(&sqlca, &error_message, &lrecl);
if(rc == 0)
 for(j=0; j<TIAR_DIM; j++)
 {
 for(k=0; k<TIAR_LEN; k++)
 putchar(error_message.error_text[j][k]);
 putchar('\n');
 }
else
{
 printf(" *** ERROR: DSNTIAR could not format the message\n");
 printf(" *** SQLCODE is %d\n",SQLCODE);
 printf(" *** SQLERRM is \n");
 for(j=0; j<sqlca.sqlerrml; j++)
 printf("%c", sqlca.sqlerrmc[j]);
 printf("\n");
}

} /* end sql_error */

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ2C

ESTE JCL REALIZA LA CONFIGURACIÓN DE COBOL DE LA FASE 2 PARA LAS APLICACIONES DE MUESTRA.

```

//*****
//** NAME = DSNTEJ2C
//**
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 2
//** COBOL
//**
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
//**
//** STATUS = VERSION 12
//**
//** FUNCTION = THIS JCL PERFORMS THE PHASE 2 COBOL SETUP FOR THE
//** SAMPLE APPLICATIONS. IT PREPARES AND EXECUTES

```

```

//* COBOL BATCH PROGRAMS.
//*
//* THIS JOB IS RUN AFTER PHASE 1.
//*
//*
//* CHANGE ACTIVITY =
//* 08/18/2014 Single-phase migration s21938_inst1 s21938
//*
//***** ****
//*
//JOBLIB DD DISP=SHR,DSN=DSN!!0.SDSNEXIT
// DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
// DD DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
//*
//* STEP 1: CREATE COPY FILE TABLE DESCRIPTIONS (DCLGEN)
//PH02CS01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*,DCB=(RECFM=F,LRECL=200,BLKSIZE=200)
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCDP)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCEM)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCDM)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCAD)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCA2)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCCS)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCOV)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCDT)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MCED)'
DSN SYSTEM(DSN)
DCLGEN TABLE(VDEPT) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCDP') +
 ACTION(ADD) APOST +
 LANGUAGE(IBMCOB) +
 STRUCTURE(PDEPT)
DCLGEN TABLE(VEMP) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCEM') +
 ACTION(ADD) APOST +
 LANGUAGE(IBMCOB) +
 STRUCTURE(PEMP)
DCLGEN TABLE(VDEPMG1) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCDM') +
 ACTION(ADD) APOST +
 LANGUAGE(IBMCOB) +
 STRUCTURE(PDEPMGR)
DCLGEN TABLE(VASTRDE1) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCAD') +
 ACTION(ADD) APOST +
 LANGUAGE(IBMCOB) +
 STRUCTURE(PASTRDET)
DCLGEN TABLE(VASTRDE2) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCA2') +
 ACTION(ADD) APOST +
 LANGUAGE(IBMCOB) +
 NAMES(ADE2) +
 STRUCTURE(PASTRDE2)
DCLGEN TABLE(VCONA) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCCS') +
 ACTION(ADD) APOST +
 LANGUAGE(IBMCOB) +
 STRUCTURE(PCONA)
DCLGEN TABLE(VOPTVAL) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCOV') +
 ACTION(ADD) APOST +
 LANGUAGE(IBMCOB) +
 STRUCTURE(POPTVAL)
DCLGEN TABLE(VDSPTXT) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCDT') +
 ACTION(ADD) APOST +
 LANGUAGE(IBMCOB) +
 STRUCTURE(PDSPTXT)
DCLGEN TABLE(VEMPDPT1) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MCED') +
 ACTION(ADD) APOST +

```

```

LANGUAGE(IBMCOB) +
STRUCTURE(PEMPDPT1)
END
//*
//* STEP 2: PREPARE ERROR MESSAGE ROUTINE
//PH02CS02 EXEC DSNHICOB, MEM=DSN8MCG,
// COND=(4,LT),
// PARM,PC='HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
// NOXREF,'SQL(DB2)', 'DEC(31)'),
// PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)'),
// PARM.LKED='LIST,XREF,MAP,RENT'
//PC.DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA(DSN8MCG),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8MCG),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8MCG),
// DISP=SHR
//*
//* STEP 3: PREPARE COBOL PHONE PROGRAM
//PH02CS03 EXEC DSNHICOB, MEM=DSN8BC3,
// COND=(4,LT),
// PARM,PC='HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
// NOXREF,'SQL(DB2)', 'DEC(31)'),
// PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')
//PC.DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA(DSN8BC3),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8BC3),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8BC3),
// DISP=SHR
//LKED.RUNLIB DD DSN=DSN!!0.RUNLIB.LOAD,
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(DSNELI)
// INCLUDE RUNLIB(DSN8MCG)
//*
//* STEP 4: BIND AND RUN PROGRAMS
//PH02CS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//REPORT DD SYSOUT=*
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT BIND, EXECUTE ON PLAN DSN8BH!!
TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE(DSN8BH!!) MEMBER(DSN8BC3) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8BH!!) PKLIST(DSN8BH!!.*.) +
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
RUN PROGRAM(DSN8BC3) PLAN(DSN8BH!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END
//CARDIN DD *
L*
LJ0%
L%SON
LSMITH
LBROWN ALAN
LBROWN DAVID
U 0002304265
//*

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ2D

ESTE JCL REALIZA LA CONFIGURACIÓN DE IDIOMAS DE LA FASE 2 C PARA LAS APLICACIONES DE MUESTRA.

```
//*****
/* NAME = DSNTEJ2D
/*
/* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
/* PHASE 2
/* C
/*
/* LICENSED MATERIALS - PROPERTY OF IBM
/* 5650-DB2
/* (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
/*
/* STATUS = VERSION 12
/*
/* FUNCTION = THIS JCL PERFORMS THE PHASE 2 C LANGUAGE SETUP FOR
/* THE SAMPLE APPLICATIONS. IT PREPARES AND EXECUTES
/* C BATCH PROGRAMS.
/*
/* NOTES = ENSURE THAT LINE NUMBER SEQUENCING IS SET 'ON' IF
/* THIS JOB IS SUBMITTED FROM AN ISPF EDIT SESSION
/*
/* THIS JOB IS RUN AFTER PHASE 1.
/*
/* CHANGE ACTIVITY =
/* 08/18/2014 Single-phase migration s21938_inst1 s21938
/*
//*****
/*
//JOBLIB DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
// DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
/*
/* STEP 1 : PREPARE ERROR MESSAGE ROUTINE
//PH02DS01 EXEC DSNHC,MEM=DSN8MDG,
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF',
// PARM.C='SOURCE XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM.LKED='NCAL,MAP,AMODE=31,RMODE=ANY'
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8MDG),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8MDG),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB LOAD(DSN8MDG),
// DISP=SHR
/*
/* STEP 2 : PREPARE C PHONE PROGRAM
//PH02DS02 EXEC DSNHC,MEM=DSN8BD3,
// COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF',
// PARM.C='SOURCE LIST MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM.LKED='AMODE=31,RMODE=ANY,MAP'
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8BD3),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8BD3),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB LOAD(DSN8BD3),
// DISP=SHR
//LKED.RUNLIB DD DSN=DSN!!0.RUNLIB LOAD,
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNELI)
INCLUDE RUNLIB(DSN8MDG)
/*
/* STEP 3 : BIND AND RUN PROGRAMS
//PH02DS03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLMLIB DD DISP=SHR,DSN=DSN!!0.DBRLMLIB.DATA
//SYSTSPRT DD SYSOUT=*

//SYSPRINT DD SYSOUT=*

//CEEDUMP DD SYSOUT=*

//SYSUDUMP DD SYSOUT=*

//SYSOUT DD SYSOUT=*

//REPORT DD SYSOUT=*
```

```

//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT BIND, EXECUTE ON PLAN DSN8BD!!
 TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE(DSN8BD!!) MEMBER(DSN8BD3) APPLCOMPAT(V!!R1) +
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8BD!!) PKLIST(DSN8BD!!.*)
 ACTION(REPLACE) RETAIN +
 ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
 LIB('DSN!!0.RUNLIB LOAD')
RUN PROGRAM(DSN8BD3) PLAN(DSN8BD!!) -
 LIB('DSN!!0.RUNLIB LOAD')
END
//CARDIN DD *
L*
LJO%
L%SON
LSMITH
LBROWN ALAN
LBROWN DAVID
U 0002304265
//*

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

### DSNTEJ2E

ESTE JCL REALIZA LA CONFIGURACIÓN DEL LENGUAJE C++ DE LA FASE 2 PARA LAS APLICACIONES DE MUESTRA.

```

//***** ****
//** NAME = DSNTEJ2E
//**
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 2
//** C++
//**
//** Licensed Materials - Property of IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM Corp. All Rights Reserved.
//**
//** STATUS = Version 12
//**
//** FUNCTION = THIS JCL PERFORMS THE PHASE 2 C++ LANGUAGE SETUP FOR
//** THE SAMPLE APPLICATIONS. IT PREPARES AND EXECUTES
//** C++ BATCH PROGRAMS.
//**
//** NOTES = ENSURE THAT LINE NUMBER SEQUENCING IS SET 'ON' IF
//** THIS JOB IS SUBMITTED FROM AN ISPF EDIT SESSION
//**
//** THIS JOB IS RUN AFTER PHASE 1.
//**
//** CHANGE ACTIVITY =
//** 10/16/2013 Don't use prelinker by default PI13612 DM1812
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//**
//***** ****
//** JOBLIB DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
//** DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
//**
//** STEP 1 : PREPARE ERROR MESSAGE ROUTINE
//**PH02ES01 EXEC DSNHC,MEM=DSN8MDG,
//** PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
//** SOURCE,XREF),
//** PARM.C='SOURCE XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
//** PARM.LKED='NCAL,MAP,AMODE=31,RMODE=ANY'
//**PC.DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA(DSN8MDG),
//** DISP=SHR
//**PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
//** DISP=SHR
//**PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8MDG),
//** DISP=SHR

```

```

//C.SYSLIN DD DSN=&&LOADSET2,
// DISP=(MOD,PASS),
// UNIT=SYSDA,SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//LKED.SYSLIN DD DSN=&&LOADSET2,DISP=(OLD,PASS)
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8MDG),
// DISP=SHR
///*
//** STEP 2 : PREPARE CLASSES USED BY C++ PHONE PROGRAM
//PH02ES02 EXEC DSNHCPP,MEM=DSN8BECL,COND=(4,LT),
// PARM.PC='HOST(CPP),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.CP='/'CXX SOURCE XREF OPTFILE(DD:CCOPTS)',
// 'LANGLVL(EXTENDED)'),
// PARM.LKED='NCAL,MAP,AMODE=31,RMODE=ANY'
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8BE3),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSENSAMP(DSN8BECL),
// DISP=SHR
//CP.USERLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//CP.SYSLIN DD DSN=&&LOADSET1,
// DISP=(MOD,PASS),
// UNIT=SYSDA,SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//LKED.SYSLIN DD DSN=&&LOADSET1,DISP=(OLD,PASS)
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8BECL),
// DISP=SHR
//LKED.RUNLIB DD DSN=DSN!!0.RUNLIB.LOAD,
// DISP=SHR
///*
//** STEP 3 : PREPARE C++ PHONE PROGRAM
//PH02ES03 EXEC DSNHCPP,MEM=DSN8BE3,
// COND=(4,LT),
// PARM.PC='HOST(CPP),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.CP='/'CXX SOURCE XREF OPTFILE(DD:CCOPTS)',
// 'LANGLVL(EXTENDED)'),
// PARM.LKED='AMODE=31,RMODE=ANY,MAP,UPCASE'
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DUMMY),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSENSAMP(DSN8BE3),
// DISP=SHR
//CP.USERLIB DD DSN=DSN!!0.SDSENSAMP,
// DISP=SHR
//LKED.SYSLIN DD DSN=&&LOADSET,DISP=(OLD,DELETE)
// DD DSN=&&LOADSET1,DISP=(OLD,DELETE)
// DD DSN=&&LOADSET2,DISP=(OLD,DELETE)
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8BE3),
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(DSNELI)
// INCLUDE SYSLMOD(DSN8MDG)
// NAME DSN8BE3(R)
///*
//** STEP 4 : BIND AND RUN PROGRAMS
//PH02ES04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLIB DD DISP=SHR,DSN=DSN!!0.DBRLIB.DATA
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//REPORT DD SYSOUT=*
//SYSIN DD *
//SET CURRENT SQLID = 'SYSADM';
//GRANT BIND, EXECUTE ON PLAN DSN8BE!!
//TO PUBLIC;
//SYSTSIN DD *
//DSN SYSTEM(DSN)
BIND PACKAGE(DSN8BE!!) MEMBER(DSN8BE3) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8BE!!) PKLIST(DSN8BE!!.*)
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
RUN PROGRAM(DSN8BE3) PLAN(DSN8BE!!) -

```

```

LIB('DSN!!0.RUNLIB LOAD')
END
//CARDIN DD *
L*
LJO%
L%SON
LSMITH
LBROWN ALAN
LBROWN DAVID
U 0002304265
/*
```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO" en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ2P

ESTE JCL REALIZA LA CONFIGURACIÓN DE LA FASE 2 PARA LAS APLICACIONES DE MUESTRA EN SITIOS CON PL/I.

```

//***** ****
//** NAME = DSNTEJ2P
//*
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 2
//** PL/I
//*
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
//*
//** STATUS = VERSION 12
//*
//** FUNCTION = THIS JCL PERFORMS THE PHASE 2 SETUP FOR THE SAMPLE
//** APPLICATIONS AT SITES WITH PL/I. IT PREPARES AND
//** EXECUTES THE PL/I BATCH PROGRAM.
//*
//** THIS JOB IS RUN AFTER PHASE 1.
//*
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//*
//***** ****
//** JOBLIB DD DISP=SHR,DSN=DSN!!0.SDSNEXIT
//** DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
//** DD DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
//** STEP 1: CREATE COPY FILE TABLE DESCRIPTIONS (DCLGEN)
//PH02PS01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*,DCB=(RECFM=F,LRECL=200,BLKSIZE=200)
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPAC)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPDP)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPEM)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPPJ)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPPA)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPEP)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPDM)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPAD)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPA2)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPPR)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPPD)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPP2)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPSA)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPS2)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPCS)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MP0V)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPDT)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPED)'
DELETE 'DSN!!0.SRCLIB.DATA(DSN8MPFP)'
DSN SYSTEM(DSN)
 DCLGEN TABLE(VACT) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPAC)') +
 ACTION(ADD) +
 LANGUAGE(PLI) +
```

```

STRUCTURE(PACT)
DCLGEN TABLE(VDEPT) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPDP)') +
ACTION(ADD) +
LANGUAGE(PLI) +
STRUCTURE(PDEPT)
DCLGEN TABLE(VEMP) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPEM)') +
ACTION(ADD) +
LANGUAGE(PLI) +
STRUCTURE(PEMP)
DCLGEN TABLE(VPROJ) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPPJ)') +
ACTION(ADD) +
LANGUAGE(PLI) +
STRUCTURE(PPROJ)
DCLGEN TABLE(VPROJECT) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPPA)') +
ACTION(ADD) +
LANGUAGE(PLI) +
STRUCTURE(PPROJECT)
DCLGEN TABLE(VEMPPROJECT) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPEP)') +
ACTION(ADD) +
LANGUAGE(PLI) +
STRUCTURE(PEMPPROJECT)
DCLGEN TABLE(VDEPMG1) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPDM)') +
ACTION(ADD) +
LANGUAGE(PLI) +
STRUCTURE(PDEPMGR)
DCLGEN TABLE(VASTRDE1) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPAD)') +
ACTION(ADD) +
LANGUAGE(PLI) +
STRUCTURE(PASTRDET)
DCLGEN TABLE(VASTRDE2) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPA2)') +
ACTION(ADD) +
LANGUAGE(PLI) +
NAMES(ASTD) +
STRUCTURE(PASTRDE2)
DCLGEN TABLE(VPROJRE1) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPPR)') +
ACTION(ADD) +
LANGUAGE(PLI) +
STRUCTURE(PPROJRES)
DCLGEN TABLE(VPSTRDE1) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPPD)') +
ACTION(ADD) +
LANGUAGE(PLI) +
STRUCTURE(PPSTRDET)
DCLGEN TABLE(VPSTRDE2) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPP2)') +
ACTION(ADD) +
LANGUAGE(PLI) +
NAMES(PSTD) +
STRUCTURE(PPSTRDE2)
DCLGEN TABLE(VSTAFAC1) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPSA)') +
ACTION(ADD) +
LANGUAGE(PLI) +
NAMES(STAF) +
STRUCTURE(PSTAFAC1)
DCLGEN TABLE(VSTAFAC2) +
OWNER(DSN8!!0) +
LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPS2)') +
ACTION(ADD) +
LANGUAGE(PLI) +
STRUCTURE(PSTAFACT)

```

```

DCLGEN TABLE(VCONA) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPCS)') +
 ACTION(ADD) +
 LANGUAGE(PLI) +
 STRUCTURE(PCONA)
DCLGEN TABLE(VOPTVAL) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPOV)') +
 ACTION(ADD) +
 LANGUAGE(PLI) +
 STRUCTURE(POPTVAL)
DCLGEN TABLE(VDSPXTXT) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPDT)') +
 ACTION(ADD) +
 LANGUAGE(PLI) +
 STRUCTURE(PDSPXTXT)
DCLGEN TABLE(VEMPDP1) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPED)') +
 ACTION(ADD) +
 LANGUAGE(PLI) +
 STRUCTURE(PEMPDP1)
DCLGEN TABLE(VFORPLA) +
 OWNER(DSN8!!0) +
 LIBRARY('DSN!!0.SRCLIB.DATA(DSN8MPFP)') +
 ACTION(ADD) +
 LANGUAGE(PLI) +
 STRUCTURE(PFORPLA)
END
///*
//** STEP 2: PREPARE PLI MESSAGE ROUTINE
//PH02PS02 EXEC DSNHPLI,MEM=DSN8MPG,
// COND=(4,LT),
// PARM.PC='HOST(PLI),CCSID(37),SOURCE,XREF,STDSQL(NO)',
// PARM.PLI='(NOPT,SOURCE,OBJECT,MARGINS(2,72,0),NORENT',
// 'LIMITS(EXTNAME(7)),OPTIONS',
// PARM.LKED='NCAL,LIST,XREF'
//PPLI.SYSIN DD DSN=DSN!!0.SSDNSAMP(DSN8MPG),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8MPG),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8MPG),
// DISP=SHR
///*
//** STEP 3: PREPARE TELEPHONE PROGRAM
//PH02PS03 EXEC DSNHPLI,MEM=DSN8BP3,
// COND=(4,LT),
// PARM.PC='HOST(PLI),CCSID(37),SOURCE,XREF,STDSQL(NO)',
// PARM.PLI='(NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',
// 'LIMITS(EXTNAME(7)),OPTIONS'
//PPLI.SYSIN DD DSN=DSN!!0.SSDNSAMP(DSN8BP3),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8BP3),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8BP3),
// DISP=SHR
//LKED.RUNLIB DD DSN=DSN!!0.RUNLIB.LOAD,
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(DSNELI)
// INCLUDE RUNLIB(DSN8MPG)
///*
//** STEP 4: BIND PROGRAMS
//PH02PS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLIB DD DSN=DSN!!0.DBRLIB.DATA,
// DISP=SHR
//SYSTSPPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
// SET CURRENT SQLID = 'SYSADM';
// GRANT BIND, EXECUTE ON PLAN DSN8BP!!
// TO PUBLIC;
//SYSTSIN DD *
// DSN SYSTEM(DSN)
// BIND PACKAGE(DSN8BP!!) MEMBER(DSN8BP3) APPLCOMPAT(V!!R1) +

```

```

 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8BP!!) PKLIST(DSN8BP!!.*)
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END
/*
//** STEP 5: RUN PROGRAMS
//PH02PS05 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//REPORT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//CARDIN DD *
L*
LJ0%
L%SON
LSMITH
LBROWN ALAN
LBROWN DAVID
U 0002304265
//SYSTSIN DD *
DSN SYSTEM(DSN)
RUN PROGRAM(DSN8BP3) PLAN(DSN8BP!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END
/*

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ2F

ESTE JCL REALIZA LA CONFIGURACIÓN DE LA FASE 2 PARA LAS APLICACIONES DE MUESTRA EN SITIOS CON FORTRAN.

```

//*****
//** NAME = DSNTEJ2F
//*
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 2
//** FORTRAN
//*
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
//*
//** STATUS = VERSION 12
//*
//** FUNCTION = THIS JCL PERFORMS THE PHASE 2 SETUP FOR THE SAMPLE
//** APPLICATIONS AT SITES WITH FORTRAN. IT PREPARES
//** AND EXECUTES THE FORTRAN BATCH PROGRAM.
//*
//** THIS JOB IS RUN AFTER PHASE 1.
//*
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//*
//*****
//*
//JOBLIB DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
// DD DSN=SYS1.VSF2FORT,DISP=SHR
//*
//** STEP 1: PREPARE DSNTIR ROUTINE
//PH02FS01 EXEC DSNHASM,MEM=DSNTIR,
// PARM.PC='HOST(ASM),STDSQL(NO)',
// PARM.ASM='RENT,OBJECT,NODECK',
// PARM.LKED='XREF,NCAL'
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSNTIR),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SSDNSAMP,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SSDNSAMP(DSNTIR),
// DISP=SHR
//ASML.SYSLIB DD DSN=DSN!!0.SSDNSAMP,

```

```

// DISP=SHR
// DD DSN=SYS1.MACLIB,DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSNTIR),
// DISP=SHR
//LKED.SYSIN DD *
// ENTRY DSNTIR
// NAME DSNTIR(R)
///*
//** STEP 2: PREPARE TELEPHONE PROGRAM
//PH02FS02 EXEC DSNHFOR,MEM=DSN8BF3,
// COND=(4,LT),
// PARM.PC='HOST(FORTRAN),SOURCE,XREF,STDSQL(NO)',
// PARM.FORT='MAP,GOSTMT,SOURCE,XREF'
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8BF3),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8BF3),
// DISP=SHR
//LKED.SYSLIB DD
// DD
// DD DSN=DSN!!0.RUNLIB.LOAD,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8BF3),
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(DSNHFT)
///*
//** STEP 3: BIND AND RUN PROGRAM
//PH02FS03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//FT06F001 DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
// SET CURRENT SQLID = 'SYSADM';
// GRANT BIND, EXECUTE ON PLAN DSN8BF!!
// TO PUBLIC;
//SYSTSIN DD *
// DSN SYSTEM(DSN)
BIND PACKAGE(DSN8BF!!) MEMBER(DSN8BF3) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8BF!!) PKLIST(DSN8BF!!.*)
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSN8BF3) PLAN(DSN8BF!!) -
LIB('DSN!!0.RUNLIB.LOAD')
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END
//FT05F001 DD *
L*
LJ0%
L%SON
LSMITH
LBROWN ALAN
LBROWN DAVID
U 0002304265
//*

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

### DSNTEJ3C

ESTE JCL REALIZA LA CONFIGURACIÓN DE LA FASE 3 PARA LAS APLICACIONES DE MUESTRA EN SITIOS CON COBOL.

```

//*****NAME = DSNTEJ3C*****
//*
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 3
//** COBOL, ISPF, CAF
//**
//** LICENSED MATERIALS - PROPERTY OF IBM

```

```

//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM Corp. All Rights Reserved.
//**
//** STATUS = Version 12
//**
//** FUNCTION = THIS JCL PERFORMS THE PHASE 3 SETUP FOR THE SAMPLE
//** APPLICATIONS AT SITES WITH COBOL. IT PREPARES THE
//** COBOL ISPF CAF TELEPHONE APPLICATION AND THE REMOTE
//** COBOL ORGANIZATION APPLICATION.
//**
//** NOTE: DDF MUST BE UP FOR STEP PH03CS06 TO EXECUTE
//**
//** RUN THIS JOB ANYTIME AFTER PHASE 2.
//**
//**
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//** ****
//** JOBLIB DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
//**
//**
//** STEP 1: PREPARE THE COBOL CAF INTERFACE
//**
//PH03CS01 EXEC DSNHICOB,MEM=DSN8CC,
// COND=(4,LT),
// PARM.PC='HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
// NOXREF,'SQL(DB2)','DEC(31)'),
// PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')
//PC.DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA(DSN8CC),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CC),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CC),
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(DSNALI)
//**
//** STEP 2: PREPARE THE CONNECTION MANAGER
//**
//PH03CS02 EXEC DSNHICOB,MEM=DSN8SCM,
// COND=(4,LT),
// PARM.PC='HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
// NOXREF,'SQL(DB2)','DEC(31)'),
// PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')
//PC.DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA(DSN8SCM),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8SCM),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8SCM),
// DISP=SHR
//**
//** STEP 3: PREPARE THE TELEPHONE APPLICATION
//**
//PH03CS03 EXEC DSNHICOB,MEM=DSN8SC3,
// COND=(4,LT),
// PARM.PC='HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
// NOXREF,'SQL(DB2)','DEC(31)'),
// PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')
//PC.DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA(DSN8SC3),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8SC3),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8SC3),
// DISP=SHR
//LKED.RUNLIB DD DSN=DSN!!0.RUNLIB.LOAD,
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(DSNALI)
// INCLUDE RUNLIB(DSN8MCG)
//**
//** STEP 4: PREPARE THE REMOTE ORGANIZATION APPLICATION
//**
//PH03CS04 EXEC DSNHICOB,MEM=DSN8HC3,

```

```

// COND=(4,LT),
// PARM.PC='HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
// NOXREF,'SQL(DB2)', 'DEC(31)'),
// PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')
//PC.DBMLLIB DD DSN=DSN!!0.DBMLLIB.DATA(DSN8HC3),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8HC3),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8HC3),
// DISP=SHR
//LKED.RUNLIB DD DSN=DSN!!0.RUNLIB.LOAD,
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(DSNALI)
// INCLUDE RUNLIB(DSN8MCG)
///*
//** STEP 5: BIND THE TELEPHONE APPLICATION PROGRAM
///*
//PH03CS05 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBMLLIB DD DSN=DSN!!0.DBMLLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT BIND, EXECUTE ON PLAN DSN8SC!!
TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE(DSN8SC!!) MEMBER(DSN8SC3) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8SC!!) PKLIST(DSN8SC!!.*)
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
///*
//** STEP 6: BIND THE REMOTE ORGANIZATION APPLICATION
///*
//PH03CS06 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBMLLIB DD DSN=DSN!!0.DBMLLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT BIND, EXECUTE ON PLAN DSN8HC!!
TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE(DSN8HC!!) MEMBER (DSN8HC3) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(SAMPLC.DSN8HC!!) MEMBER(DSN8HC3) +
APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8HC!!) -
PKLIST(DSN8HC!!.* ,SAMPLC.DSN8HC!!.*) -
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ6

Ejecutar este trabajo en la ubicación remota para actualizar la ubicación de la muestra en la tabla de departamentos. Ejecutar este trabajo en cualquier momento después de la fase 3.

```

//***** 00010000
//** NAME = DSNTEJ6 00020000

```

```

//*
//** DESCRIPTIVE NAME = DB2 REMOTE UNIT OF WORK SAMPLE APPLICATION 00030000
//** PHASE 6 00040000
//*
//** LICENSED MATERIALS - PROPERTY OF IBM 00050000
//** 5615-DB2 00060000
//** (C) COPYRIGHT 1982, 2013 IBM CORP. ALL RIGHTS RESERVED. 00070000
//*
//** STATUS = VERSION 11 00080001
//*
//** FUNCTION = RUN THIS JOB AT THE REMOTE LOCATION TO UPDATE THE 00090001
//** SAMPLE LOCATION IN DEPARTMENT TABLE 00100000
//*
//** RUN THIS JOB ANYTIME AFTER PHASE 3. 00110001
//*
//** CHANGE ACTIVITY = 00120000
//** 11/07/2012 ADD SET CURRENT SQLID DN1651_INST1 / DN1651 00130000
//** 05/17/2013 FIX COPYRIGHT STATEMENT 49779_077_724 00140000
//*
//***** 00150000
//** RUN THIS JOB ANYTIME AFTER PHASE 3. 00160000
//*
//** CHANGE ACTIVITY = 00170000
//** 00180000
//** 11/07/2012 ADD SET CURRENT SQLID DN1651_INST1 / DN1651 00180100
//** 05/17/2013 FIX COPYRIGHT STATEMENT 49779_077_724 00180201
//*
//***** 00181000
//***** 00190000
//*
//** JOBLIB DD DSN=DSN!!0.SDSNLOAD,DISP=SHR 00200000
//*
//** STEP 1: UPDATE SAMPLE LOCATIONS IN DEPARTMENT TABLE 00210000
//*
//** PH06S01 EXEC PGM=IKJEFT01,DYNAMNBR=20 00220000
//** SYSTSPRT DD SYSOUT=*
//** SPRINT DD SYSOUT=*
//** SYSUDUMP DD SYSOUT=*
//** SYSOUT DD SYSOUT=*
//** SYSTSIN DD *
//** DSN SYSTEM(DSN) 00230000
//** RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) - 00240000
//** LIB('DSN!!0.RUNLIB.LOAD') 00250000
//** SYSIN DD *
//** SET CURRENT SQLID = 'SYSADM';
//** UPDATE DEPT SET LOCATION = 'SAMPLOC' WHERE DEPTNO = 'F22'; 00260000
//** UPDATE DEPT SET LOCATION = 'THISLOCN' WHERE LOCATION = ' ' ; 00270000
//*
//** 00280000
//** 00290000
//** 00300000
//** 00310000
//** 00320000
//** 00330000
//** 00340000
//** 00350000
//** 00360000
//** 00370000
//*

```

## Referencia relacionada

["Ejemplos de aplicaciones en TSO" en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ3P

ESTE JCL REALIZA LA CONFIGURACIÓN DE LA FASE 3 PARA LAS APLICACIONES DE MUESTRA EN SITIOS CON PL/I.

```

//*****
//** NAME = DSNTEJ3P
//*
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 3
//** PL/I, ISPF, CAF
//*
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
//*
//** STATUS = VERSION 12
//*
//** FUNCTION = THIS JCL PERFORMS THE PHASE 3 SETUP FOR THE SAMPLE
//** APPLICATIONS AT SITES WITH PL/I. IT PREPARES THE
//** PL/I ISPF CAF TELEPHONE PROGRAM.
//*
//** RUN THIS JOB ANYTIME AFTER PHASE 2.
//*
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//*
//*****
//*
//** JOBLIB DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
//** DD DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
//*
//** STEP 1: PREPARE THE ISPF CAF CONNECTION MANAGER
//*

```

```

//PH03PS01 EXEC DSNHPLI,MEM=DSN8SPM,
// COND=(4,LT),
// PARM.PC='HOST(PLI),CCSID(37),SOURCE,XREF,STDSQL(NO)',
// PARM.PLI='(NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',
// 'LIMITS(EXTNAME(7)),OPTIONS')
//PPLI.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8SPM),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8SPM),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8SPM),
// DISP=SHR
//LKED.SYSIN DD *
// ENTRY CEESTART
///*
//** STEP 2: PREPARE THE ISPF CAF TELEPHONE APPLICATION
///*
//PH03PS02 EXEC DSNHPLI,MEM=DSN8SP3,
// COND=(4,LT),
// PARM.PC='HOST(PLI),CCSID(37),SOURCE,XREF,STDSQL(NO)',
// PARM.PLI='(NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',
// 'LIMITS(EXTNAME(7)),OPTIONS')
//PPLI.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8SP3),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8SP3),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8SP3),
// DISP=SHR
//LKED.RUNLIB DD DSN=DSN!!0.RUNLIB.LOAD,
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(DSNALI)
// INCLUDE RUNLIB(DSN8MPG)
///*
//** STEP 3: BIND THE ISPF CAF TELEPHONE APPLICATION
///*
//PH03PS03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLIB DD DISP=SHR,DSN=DSN!!0.DBRLIB.DATA
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
// SET CURRENT SQLID = 'SYSADM';
// GRANT BIND, EXECUTE ON PLAN DSN8SP!!
// TO PUBLIC;
//SYSTSIN DD *
//DSN SYSTEM(DSN)
BIND PACKAGE(DSN8SP!!) MEMBER(DSN8SP3) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8SP!!) PKLIST(DSN8SP!!.*)
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END
//*/

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ2A

ESTE JCL REALIZA LA CONFIGURACIÓN DE LA FASE 2 PARA LAS APLICACIONES DE MUESTRA.

```

//*****
//* NAME = DSNTEJ2A
//*
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 2
//** ASSEMBLER
//*
//** Licensed Materials - Property of IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM Corp. All Rights Reserved.

```

```

//*
//** STATUS = Version 12
//*
//** FUNCTION = THIS JCL PERFORMS THE PHASE 2 SETUP FOR THE SAMPLE
//** APPLICATIONS. IT PREPARES AND RUNS THE SAMPLE
//** ASSEMBLER BATCH TABLE UNLOAD PROGRAM
//*
//** THIS JOB IS RUN AFTER PHASE 1.
//*
//** NOTICE =
//** THIS SAMPLE JOB USES DB2 UTILITIES. SOME UTILITY FUNCTIONS ARE
//** ELEMENTS OF SEPARATELY ORDERABLE PRODUCTS. SUCCESSFUL USE OF
//** A PARTICULAR SAMPLE JOB MAY BE DEPENDENT UPON THE OPTIONAL
//** PRODUCT BEING LICENSED AND INSTALLED IN YOUR ENVIRONMENT.
//*
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//*
//*****=====
//*
//JOBLIB DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
//*
//** PRECOMPILE, ASSEMBLE, AND LINK EDIT THE UNLOAD PROGRAM
//*
//PREPUNL EXEC DSNHASM,MEM=DSNTIAUL,
// PARM.PC='HOST(ASM),STDSQL(NO),VERSION(AUTO)',
// PARM.ASM='RENT,OBJECT,NODECK',
// PARM.LKED='RENT,XREF,AMODE=ANY,RMODE=24'
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSNTIAUL),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSNTIAUL),
// DISP=SHR
//ASM.SYSLIB DD
// DD DSN=DSN!!0.SDSNMACS,
// DISP=SHR
// DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSNTIAUL),
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(DSNELI)
// NAME DSNTIAUL(R)
//*
//** BIND THE UNLOAD PROGRAM AND GRANT EXECUTE AUTHORITY
//*
//BINDUNL EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLIB DD DSN=DSN!!0.DBRLIB.DATA,
// DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE(DSNTIB!!) MEM(DSNTIAUL) APPLCOMPAT(V!!R1) +
CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC) +
LIB('DSN!!0.DBRLIB.DATA')
BIND PLAN(DSNTIB!!) PKLIST(DSNTIB!!.*)
ACTION(REPLACE) RETAIN +
CURRENTDATA(NO) ISO(CS) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) +
LIB('DSN!!0.RUNLIB.LOAD')
END
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT EXECUTE ON PLAN DSNTIB!!
TO PUBLIC;
//*
//** DELETE DATA SETS, DROP TABLES TO ALLOW RERUNS
//*
//DELETE EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DELETE 'DSN!!0.DSN8UNLD.SYSREC00'
DELETE 'DSN!!0.DSN8UNLD.SYSREC01'
DELETE 'DSN!!0.DSN8UNLD.SYSPUNCH'
DSN SYSTEM(DSN)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) PARM('RC0') -
LIB('DSN!!0.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*

```

```

//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 DROP TABLE DSN8!!0.NEWDEPT ;
 DROP TABLE DSN8!!0.NEWPHONE ;
 DROP DATABASE DSN8D!!U ;
 DROP STOGROUP DSN8G!!U ;
 COMMIT;
/*
/* CREATE NEW TABLES
/*
//CREATE EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
 LIB('DSN!!0.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 CREATE STOGROUP DSN8G!!U
 VOLUMES (DSNV01)
 VCAT DSNC!!0;

 CREATE DATABASE DSN8D!!U
 STOGROUP DSN8G!!U
 CCSID EBCDIC;

 CREATE TABLE DSN8!!0.NEWDEPT
 (DEPTNO CHAR(3) NOT NULL,
 DEPTNAME VARCHAR(36) NOT NULL,
 MGRNO CHAR(6) ,
 ADMRDEPT CHAR(3) NOT NULL,
 LOCATION CHAR(16)) NOT NULL,
 IN DATABASE DSN8D!!U
 CCSID EBCDIC;

 CREATE TABLE DSN8!!0.NEWPHONE
 (LASTNAME VARCHAR(15) NOT NULL,
 FIRSTNAME VARCHAR(12) NOT NULL,
 MIDDLEINITIAL CHAR(1) NOT NULL,
 PHONENUMBER CHAR(4) ,
 EMPLOYEENUMBER CHAR(6) NOT NULL,
 DEPTNUMBER CHAR(3) NOT NULL,
 DEPTNAME VARCHAR(36) NOT NULL)
 IN DATABASE DSN8D!!U
 CCSID EBCDIC;

/*
/* RUN UNLOAD PROGRAM
/*
//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB!!) PARM('SQL') -
 LIB('DSN!!0.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DSN=DSN!!0.DSN8UNLD.SYSREC00,
// DISP=(,CATLG),
// UNIT=SYSDA,
// SPACE=(1024,(10,10))
//SYSREC01 DD DSN=DSN!!0.DSN8UNLD.SYSREC01,
// DISP=(,CATLG),
// UNIT=SYSDA,
// SPACE=(1024,(10,10))
//SYSPUNCH DD DSN=DSN!!0.DSN8UNLD.SYSPUNCH,
// DISP=(,CATLG),
// UNIT=SYSDA,
// SPACE=(800,(15,15))
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 LOCK TABLE DSN8!!0.DEPT IN SHARE MODE;
 SELECT * FROM DSN8!!0.DEPT;
 SELECT * FROM DSN8!!0.VPHONE;

/*
/* EDIT THE OUTPUT FROM THE PROGRAM
/*
//EDIT EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=((4,LT),(4,LE,UNLOAD))
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 EDIT 'DSN!!0.DSN8UNLD.SYSPUNCH' DATA NONUM

```

```

CHANGE * 30 /DSN8!!0.DEPT/DSN8!!0.NEWDEPT/
CHANGE * 30 /DSN8!!0.VPHONE/DSN8!!0.NEWPHONE/
TOP
LIST * 999
END SAVE
/***
/** RUN LOAD UTILITY TO LOAD TABLES
/***
//LOAD EXEC DSNUPROC,PARM='DSN,DSNTEX',
// COND=((4,LT),(4,LE,UNLOAD))
//DSNTRACE DD SYSOUT=*
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
//SORTWK01 DD UNIT=SYSDA,SPACE=(4000,(20,20),,ROUND)
//SORTWK02 DD UNIT=SYSDA,SPACE=(4000,(20,20),,ROUND)
//SORTWK03 DD UNIT=SYSDA,SPACE=(4000,(20,20),,ROUND)
//SORTWK04 DD UNIT=SYSDA,SPACE=(4000,(20,20),,ROUND)
//SYSRECO0 DD DSN=DSN!!0.DSN8UNLD.SYSREC00,
// DISP=(OLD,KEEP)
//SYSREC01 DD DSN=DSN!!0.DSN8UNLD.SYSREC01,
// DISP=(OLD,KEEP)
//SYSUT1 DD UNIT=SYSDA,SPACE=(4000,(20,20),,ROUND)
//SYSIN DD DSN=DSN!!0.DSN8UNLD.SYSPUNCH,
// DISP=(OLD,KEEP)
/***

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ1P

ESTE JCL REALIZA LA CONFIGURACIÓN DE LA FASE 1 PARA APLICACIONES DE MUESTRA EN SITIOS CON PL/I.

```

//***** ****
//* NAME = DSNTEJ1P
//*
//* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//* PHASE 1
//* PL/I
//*
//* Licensed Materials - Property of IBM
//* 5650-DB2
//* (C) COPYRIGHT 1982, 2016 IBM Corp. All Rights Reserved.
//*
//* STATUS = Version 12
//*
//* FUNCTION = THIS JCL PERFORMS THE PHASE 1 SETUP FOR SAMPLE
//* APPLICATIONS AT SITES WITH PL/I.
//*
//* THIS JOB IS RUN AFTER DSNTEJ1.
//*
//* CHANGE ACTIVITY =
//* 08/18/2014 Single-phase migration s21938_inst1 s21938
//***** ****
//JOBLIB DD DISP=SHR,DSN=DSN!!0.SDSNSLOAD
// DD DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
//*
//* STEP 1 : PREPARE DSNTEP2 FOR EXECUTION
//PH01PS01 EXEC DSNHPLI,MEM=DSNTEP2,
// PARM.PC=('HOST(PLI),CCSID(37),STDSQL(NO),CONNECT(2) ' ,
// 'TWOPASS,'VERSION(AUTO)'),
// PARM.PLI=(NOPT,'MAR(2,72,0)',GS,OBJ,S,
// 'LIMITS(FIXEDBIN(31,63))','LANGLVL(SPROG)',OFFSET)
//PPLI.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSNTEP2),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSNTEP2),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSNTEP2),
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(DSNELI)
//*
//* STEP 2 : BIND AND RUN PROGRAM DSNTEP2, TO
//* PRINT THE TABLES

```

```

//PH01PS02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA,
// DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 BIND PACKAGE (DSNTEP2) MEMBER(DSNTEP2) APPLCOMPAT(V!!R1) +
 CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC)
 BIND PLAN(DSNTEP!!) PKLIST(DSNTEP2.*) +
 ACTION(REPLACE) RETAIN +
 CURRENTDATA(NO) ISO(CS) ENCODING(EBCDIC) SQLRULES(DB2)
 RUN PROGRAM(DSNTEP2) PLAN(DSNTEP!!) +
 LIB('DSN!!0.RUNLIB.LOAD') +
 PARMS('/ALIGN(MID)')
 END
/*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 GRANT EXECUTE, BIND ON PLAN DSNTEP!!
 TO PUBLIC;
 SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME,
 WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,
 SEX, BIRTHDATE, SALARY, BONUS, COMM,
 SALARY+BONUS+COMM AS TOTAL_SALARY
 FROM EMP
 ORDER BY TOTAL_SALARY;
 SELECT * FROM DEPT;
 SELECT * FROM ACT;
 SELECT * FROM EMPPROJECT;
 SELECT * FROM PROJ;
 SELECT * FROM PROJECT;
/*
/*
// STEP 3 : PREPARE DSNTEP4 FOR EXECUTION
//PH01PS03 EXEC DSNHPLI,MEM=DSNTEP4,COND=(4,LT),
// PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO),CONNECT(2)',
// TWOPASS,'VERSION(AUTO)'),
// PARM.PLI=(NOPT,'MAR(2,72,0)',GS,OBJ,S,
// 'LIMITS(FIXEDBIN(31,63))','LANGLVL(SPROG)',OFFSET)
//PPLI.SYSIN DD DSN=DSN!!0.SDSDNSAMP(DSNTEP4),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRMLIB.DATA(DSNTEP4),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSNTEP4),
// DISP=SHR
//LKED.SYSIN DD *
 INCLUDE SYSLIB(DSNELI)
/*
/*
// STEP 4 : BIND AND RUN PROGRAM DSNTEP4, TO
// PRINT THE TABLES
//PH01PS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA,
// DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 BIND PACKAGE (DSNTEP4) MEMBER(DSNTEP4) APPLCOMPAT(V!!R1) +
 CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC)
 BIND PLAN(DSNTP4!!) PKLIST(DSNTEP4.*) +
 ACTION(REPLACE) RETAIN +
 CURRENTDATA(NO) ISO(CS) ENCODING(EBCDIC) SQLRULES(DB2)
 RUN PROGRAM(DSNTEP4) PLAN(DSNTP4!!) +
 LIB('DSN!!0.RUNLIB.LOAD') +
 PARMS('/ALIGN(MID)')
 END
/*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 GRANT EXECUTE, BIND ON PLAN DSNTPE4!!
 TO PUBLIC;
 SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME,
 WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,
 SEX, BIRTHDATE, SALARY, BONUS, COMM,
 SALARY+BONUS+COMM AS TOTAL_SALARY
 FROM EMP
 ORDER BY TOTAL_SALARY;

```

```

SELECT * FROM DEPT;
SELECT * FROM ACT;
SELECT * FROM EMPPROJECT;
SELECT * FROM PROJ;
SELECT * FROM PROJECT;
/*

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ1L

ESTE JCL CREA EL MÓDULO DE CARGA DSNTEP2 DESDE LA BARaja DE OBJETOS ENVIADOS, DSNTEP2L, Y ENLAZA EL PAQUETE Y EL PLAN PARA ESTA VERSIÓN DE DSNTEP2.

```

//*****
//** NAME = DSNTEJ1L
//**
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 1
//** L/E
//**
//** Licensed Materials - Property of IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM Corp. All Rights Reserved.
//**
//** STATUS = Version 12
//**
//** FUNCTION = THIS JCL CREATES THE DSNTEP2 LOAD MODULE FROM THE
//** SHIPPED OBJECT DECK, DSNTEP2L, AND LINKS THE
//** PACKAGE AND PLAN FOR THIS VERSION OF DSNTEP2.
//**
//** = THIS JCL ALSO CREATES THE DSNTEP4 LOAD MODULE FROM
//** THE SHIPPED OBJECT DECK, DSNTEP4L, AND LINKS THE
//** PACKAGE AND PLAN FOR THIS VERSION OF DSNTEP4.
//**
//** THIS JOB IS RUN AFTER DSNTEJ1.
//**
//** NOTE: IF YOU RUN THIS JOB, YOU DO NOT NEED TO RUN THE SAMPLE
//** JOB DSNTEJ1P EXCEPT TO PREPARE CUSTOMIZED VERSIONS OF
//** THE DSNTEP2 AND DSNTEP4 SOURCE CODE (YOU NEED A PL/I
//** COMPILER TO RUN DSNTEJ1P SUCCESSFULLY).
//**
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//*****
//JOBLIB DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
// DD DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
//**
//** STEP 1 : CREATE DSNTEP2 LOADMOD FROM DSNTEP2L OBJECT DECK
//**
//PH01LS01 EXEC PGM=IEWL,PARM='XREF'
//SYSLIB DD DISP=SHR,DSN=CEE.V!R!M!.SCEELKED
// DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
//SDSNSAMP DD DISP=SHR,DSN=DSN!!0.SDSNSAMP(DSNTEP2L)
//SYSLMOD DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD(DSNTEP2)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(50,50))
//SYSLIN DD *
// INCLUDE SDSNSAMP(DSNTEP2L)
// INCLUDE SYSLIB(DSNELI)
// NAME DSNTEP2(R)
//**
//** STEP 2 : BIND AND RUN PROGRAM DSNTEP2, TO
//** PRINT THE TABLES
//**
//PH01LS02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB DD DISP=SHR,DSN=DSN!!0.SDSNSAMP
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSTIN DD *
// DSN SYSTEM(DSN)
BIND PACKAGE (DSNTEP2) MEMBER(DSN@EP2L) APPLCOMPAT(V!!R1) +
CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC)

```

```

BIND PLAN(DSNTEP!!) PKLIST(DSNTEP2.*) +
ACTION(REPLACE) RETAIN +
CURRENTDATA(NO) ISO(CS) ENCODING(EBCDIC) SQLRULES(DB2)
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP!!) +
LIB('DSN!!0.RUNLIB LOAD') +
PARMS('/ALIGN(MID)')
END
/*
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT EXECUTE, BIND ON PLAN DSNTEP!!
TO PUBLIC;
SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,
WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,
SEX, BIRTHDATE, SALARY, BONUS, COMM,
SALARY+BONUS+COMM AS TOTAL_SALARY
FROM EMP
ORDER BY TOTAL_SALARY;
SELECT * FROM DEPT;
SELECT * FROM ACT;
SELECT * FROM EMPPROJECT;
SELECT * FROM PROJ;
SELECT * FROM PROJECT;
/*
/* STEP 3 : CREATE DSNTEP4 LOADMOD FROM DSNTEP4L OBJECT DECK
/*
//PH01LS03 EXEC PGM=IEWL,COND=(4,LT),PARM='XREF'
//SYSLIB DD DISP=SHR,DSN=CEE.V!R!M!.SCEELKED
// DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
//SDSNSAMP DD DISP=SHR,DSN=DSN!!0.SDSNSAMP(DSNTEP4L)
//SYSLMOD DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD(DSNTEP4)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(50,50))
//SYSLIN DD *
 INCLUDE SDSNSAMP(DSNTEP4L)
 INCLUDE SYSLIB(DSNELI)
 NAME DSNTEP4(R)
/*
/* STEP 4 : BIND AND RUN PROGRAM DSNTEP4, TO
/* PRINT THE TABLES
/*
//PH01LS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB DD DISP=SHR,DSN=DSN!!0.SDSNSAMP
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE (DSNTEP4) MEMBER(DSN@EP4L) APPLCOMPAT(V!!R1) +
CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC)
BIND PLAN(DSNTP4!!) PKLIST(DSNTEP4.*) +
ACTION(REPLACE) RETAIN +
CURRENTDATA(NO) ISO(CS) ENCODING(EBCDIC) SQLRULES(DB2)
RUN PROGRAM(DSNTEP4) PLAN(DSNTP4!!) +
LIB('DSN!!0.RUNLIB LOAD') +
PARMS('/ALIGN(MID)')
END
/*
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT EXECUTE, BIND ON PLAN DSNTPE4!!
TO PUBLIC;
SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,
WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,
SEX, BIRTHDATE, SALARY, BONUS, COMM,
SALARY+BONUS+COMM AS TOTAL_SALARY
FROM EMP
ORDER BY TOTAL_SALARY;
SELECT * FROM DEPT;
SELECT * FROM ACT;
SELECT * FROM EMPPROJECT;
SELECT * FROM PROJ;
SELECT * FROM PROJECT;
/*

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO" en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ6P

ESTE JCL EJECUTA EL EJEMPLO DE APLICACIÓN DEL PROCEDIMIENTO ALMACENADO DE LA FASE 6.

```
//*****
/* NAME = DSNTEJ6P
/*
/* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
/*
/* PHASE 6
/*
/* LICENSED MATERIALS - PROPERTY OF IBM
/* 5650-DB2
/* (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
/*
/* STATUS = VERSION 12
/*
/* FUNCTION = THIS JCL EXECUTES THE PHASE 6 STORED PROCEDURE
/* SAMPLE APPLICATION.
/*
/* DEPENDENCIES:
/* (1) RUN SAMPLE JOB DSNTEJ6S AT THE SERVER SITE BEFORE RUNNING THIS
/* JOB; DSNTEJ6S PREPARES THE SAMPLE STORED PROC W/O RESULT SET
/* (2) RUN THIS JOB AT THE CLIENT SITE
/*
/* CHANGE ACTIVITY =
/* 08/18/2014 Single-phase migration s21938_inst1 s21938
/*
//*****
//JOBLIB DD DISP=SHR,DSN=DSN!!0.SDSNEXIT
// DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
// DD DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
// DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD
/*
//*****
/* STEP 1: PRE-COMPILE, COMPILE, AND LINK-EDIT THE CALLING PROGRAM
//*****
//PH06PS01 EXEC DSNHPLI,MEM=DSN8EP1,
// PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO),CONNECT(2)'
//PPLI.SYSIN DD DSN=DSN!!0.SSDNSAMP(DSN8EP1),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8EP1),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8EP1),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNELI)
INCLUDE SYSLIB(DSNTIAR)
//*****
/* STEP 2: BIND THE CALLING PROGRAM PACKAGE
//*****
//PH06PS02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLIB DD DSN=DSN!!0.DBRLIB.DATA,
// DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT BIND, EXECUTE ON PLAN DSN8EP1
TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE(DSN8EP!!) MEMBER(DSN8EP1) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(SAMPLOC.DSN8EP!!) APPLCOMPAT(V!!R1) +
MEMBER(DSN8EP1) ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8EP1) -
PKLIST(DSN8EP!!..DSN8EP1, SAMPLOC.DSN8EP!!..DSN8EP1) -
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
//*****
/* STEP 3: EXECUTE THE STORED PROCEDURE
//*****
//PH06PS03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
```

```

//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSN8EP1) PLAN(DSN8EP1) PARMS('/SAMPLOC')
 END
//SYSIN DD *
 -DISPLAY ARCHIVE;
 -DISPLAY THREAD(*) DETAIL;
//*

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ6S

ESTE JCL PREPARA EL PROCEDIMIENTO DE MUESTRAS ALMACENADAS.

```

//***** ****
//** NAME = DSNTEJ6S
///*
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
///*
//** PHASE 6
///*
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
///*
//** STATUS = VERSION 12
///*
//** FUNCTION = THIS JCL PREPARES THE SAMPLE STORED PROCEDURE.
///*
//** DEPENDENCIES:
//** (1) RUN THIS JOB AT THE SERVER SITE BEFORE RUNNING SAMPLE JOB
//** DSNTEJ6P AT THE CLIENT SITE
///*
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
///*
//***** ****
//JOBLIB DD DISP=SHR,DSN=DSN!0.SDSNEXIT
// DD DISP=SHR,DSN=DSN!0.SDSNLOAD
///*
//***** ****
//** STEP 1: DROP ANY EXISTING STORED PROCEDURE CALLED DSN.DSN8EP2
//***** ****
//PH06SS01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
 LIB('DSN!!0.RUNLIB.LOAD') -
 PARM('RC0')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';

 DROP PROCEDURE DSN8.DSN8EP2 RESTRICT;

///*
//***** ****
//** STEP 2: CREATE SAMPLE STORED PROCEDURE DSN8.DSN8EP2
//***** ****
//PH06SS02 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
 LIB('DSN!!0.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';

CREATE PROCEDURE

```

```

DSN8.DSN8EP2(
 IN VARCHAR(4096) CCSID EBCDIC,
 OUT INTEGER,
 OUT INTEGER,
 OUT INTEGER,
 OUT VARCHAR(8320) CCSID EBCDIC)
LANGUAGE PLI
DETERMINISTIC
NO SQL
EXTERNAL NAME DSN8EP2
PARAMETER STYLE GENERAL WITH NULLS
COLLID DSN8EP!!
WLM ENVIRONMENT WLMENV
ASUTIME LIMIT 5
STAY RESIDENT NO
PROGRAM TYPE MAIN
SECURITY DB2
NO DBINFO
RESULT SET 0
COMMIT ON RETURN NO;
/*
//***** STEP 3: PRE-COMPILE, COMPILE, AND LINK-EDIT THE STORED PROCEDURE
//***** STEP 3: PRE-COMPILE, COMPILE, AND LINK-EDIT THE STORED PROCEDURE
//***** STEP 3: PRE-COMPILE, COMPILE, AND LINK-EDIT THE STORED PROCEDURE
//PH06SS03 EXEC DSNHPLI,MEM=DSN8EP2,COND=(4,LT),
// PARM.PC='HOST(PLI),CCSID(37),STDSQL(N0),CONNECT(2)'
//PPLI.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8EP2),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8EP2),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8EP2),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
INCLUDE SYSLIB(DSNTIAR)
/*
//***** STEP 4: BIND THE STORED PROCEDURE PACKAGE
//** NOTE: THIS STEP IS COMMENTED OUT FOR THE STORED
//** PROCEDURE SAMPLE APPLICATION BECAUSE IT CONTAINS
//** NO SQL STATEMENTS. IF YOUR STORED PROCEDURE
//** CONTAINS SQL STATEMENTS, YOU MUST BIND IT AS
//** A PACKAGE.
//***** STEP 4: BIND THE STORED PROCEDURE PACKAGE
//** NOTE: THIS STEP IS COMMENTED OUT FOR THE STORED
//** PROCEDURE SAMPLE APPLICATION BECAUSE IT CONTAINS
//** NO SQL STATEMENTS. IF YOUR STORED PROCEDURE
//** CONTAINS SQL STATEMENTS, YOU MUST BIND IT AS
//** A PACKAGE.
//PH06SS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLIB DD DSN=DSN!!0.DBRLIB.DATA,
// DISP=SHR
//SYSTSPPRT DD SYSOUT=*
//SYSTSIN DD *
// DSN SYSTEM(DSN)
// BIND PACKAGE(DSN8EP!!) MEMBER(DSN8EP2) APPLCOMPAT(V!!R1) +
// ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
//*/

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

### DSNTEJ6D

ESTE JCL PREPARA Y EJECUTA UN PROGRAMA DE APLICACIÓN DE MUESTRA, DSN8ED1, QUE DEMUESTRA CÓMO LLAMAR A UN PROCEDIMIENTO ALMACENADO (Db2 ) QUE DEVUELVE UN CONJUNTO DE RESULTADOS.

```

//***** NAME = DSNTEJ6D
//*
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 6
//** SAMPLE CALLER: STORED PROCEDURE WITH RESULT SET
//** C LANGUAGE
//*
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2

```

```

//** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
//*
//** STATUS = VERSION 12
//*
//** FUNCTION = THIS JCL PREPARES AND EXECUTES A SAMPLE APPLICATION
//** PROGRAM, DSN8ED1, THAT DEMONSTRATES HOW TO CALL A DB2
//** STORED PROCEDURE THAT RETURNS A RESULT SET.
//*
//** DSN8ED1 ACCEPTS A DB2 COMMAND FROM STANDARD INPUT
//** (SYSIN) AND PASSES IT AS A PARAMETER TO THE STORED
//** PROCEDURE WHICH RUNS ON A REMOTE DB2 SUBSYSTEM (SEE
//** DSNTEJ6T FOR DETAILS). THE STORED PROCEDURE PLACES THE
//** RESPONSES IN A RESULT SET AND DSN8ED1 EXTRACTS THEM AND
//** PRINTS THEM TO STANDARD OUTPUT (SYSPRINT).
//*
//** DEPENDENCIES:
//** (1) RUN SAMPLE JOB DSNTEJ6T AT THE SERVER SITE BEFORE RUNNING THIS
//** JOB; DSNTEJ6T PREPARES THE SAMPLE STORED PROC W/ RESULT SET
//** (2) RUN THIS JOB AT THE CLIENT SITE
//*
//*
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//**
//***** ****
//JOBLIB DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
// DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
// DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
// DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
//*
//***** ****
//** STEP 1: PRE-COMPILe, COMPILE, AND LINK-EDIT THE CALLING PROGRAM
//***** ****
//PH06DS01 EXEC DSNHC,MEM=DSN8ED1,
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE LIST MAR(1,72) NORENT OPTFILE(DD:CCOPTS)',
// PARM.LKED='AMODE=31,RMODE=ANY,MAP,NORENT'
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8ED1),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8ED1),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED1),
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(DSNELI)
// INCLUDE SYSLIB(DSNTIAR)
//*
//***** ****
//** STEP 2: BIND THE CALLING PROGRAM PACKAGE
//***** ****
//PH06DS02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLIB DD DSN=DSN!!0.DBRLIB.DATA,
// DISP=SHR
//SYSTSPPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT BIND, EXECUTE ON PLAN DSN8ED1
TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED1) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(SAMPLOC.DSN8ED!!) -
APPLCOMPAT(V!!R1) +
MEMBER(DSN8ED1) ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8ED1) -
PKLIST(DSN8ED!!..DSN8ED1, -
SAMPLOC.DSN8ED!!..DSN8ED1 -
SAMPLOC.DSN8ED!!..DSN8ED2) -
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
//*
//***** ****
//** STEP 3: EXECUTE THE STORED PROCEDURE
//***** ****

```

```

//PH06DS03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSN8ED1) PLAN(DSN8ED1) PARMS('/SAMPLOC')
END
//SYSIN DD *
 -DISPLAY ARCHIVE;
 -DISPLAY THREAD(*) DETAIL;
/*

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO" en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ6T

ESTE JCL PREPARA Y EJECUTA UN PROGRAMA DE APLICACIÓN DE MUESTRA, DSN8ED2, QUE DEMUESTRA UN PROCEDIMIENTO ALMACENADO (Db2) QUE DEVUELVE UN CONJUNTO DE RESULTADOS.

```

//*****
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 6
//** SAMPLE STORED PROCEDURE WITH RESULT SET
//** C LANGUAGE
//**
//**
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
//**
//** STATUS = VERSION 12
//**
//** FUNCTION = THIS JCL PREPARES AND EXECUTES A SAMPLE APPLICATION
//** PROGRAM, DSN8ED2, THAT DEMONSTRATES A DB2 STORED
//** PROCEDURE THAT RETURNS A RESULT SET.
//**
//** DSN8ED2 ACCEPTS A DB2 COMMAND PASSED AS AN INPUT
//** PARAMETER FROM A CLIENT PROGRAM ON A REMOTE DB2
//** SUBSYSTEM. IT CALLS THE IFI UTILITY TO PROCESS THE
//** COMMAND AND PLACES THE RESPONSES IN A TEMPORARY DB2
//** TABLE SO THEY CAN BE RETURNED AS A RESULT SET TO THE
//** CLIENT.
//**
//** DEPENDENCIES:
//** (1) RUN THIS JOB AT THE SERVER SITE BEFORE RUNNING SAMPLE JOB
//** DSNTEJ6D AT THE CLIENT SITE
//**
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//**
//*****
//JOBLIB DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
// DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
// DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
// DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
//**
//*****
//** STEP 1: DROP OBJECTS CREATED BY ANY PREVIOUS RUN OF DSNTEJ6T:
//** - SAMPLE STORED PROCEDURE DSN8.DSN8ED2
//** - GLOBAL TEMPORARY TABLE DSN8.DSN8ED2_RS_TBL
//**
//*****
//PH06TS01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
 LIB('DSN!!0.RUNLIB.LOAD') -
 PARM('RC0')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';

```

```

DROP PROCEDURE DSN8.DSN8ED2 RESTRICT;
COMMIT;

DROP TABLE DSN8.DSN8ED2_RS_TBL;
COMMIT;

<��
<
```

```

DSN SYSTEM(DSN)
BIND PACKAGE(DSN8ED!!) APPLCOMPAT(V!!R1) +
 MEMBER(DSN8ED2) ACT(REP) -
 ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
/**/

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ61

Este JCL crea una aplicación de muestra, DSN8EC1, que muestra un procedimiento almacenado de Db2 para IMS ODBA.

```

//***** *****
//** Name = DSNTEJ61
//**
//** Descriptive Name = DB2 Sample Application
//** Phase 6
//** Sample Stored Procedure for IMS ODBA
//** Cobol Language
//**
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM Corp. All Rights Reserved.
//**
//** STATUS = Version 12
//**
//** Function = This JCL creates a sample application, DSN8EC1, that
//** demonstrates a DB2 stored procedure for IMS ODBA.
//**
//** DSN8EC1 can be used to insert, retrieve, update,
//** and delete rows in the IMS IVP telephone directory
//** database, DFSIVD1.
//**
//** DSN8EC1 has one input-only parm, five input/output
//** parms, and three output-only parms.
//** - Input only:
//** (1) TDBCTLID : ID of IMS subsystem where data resides
//** - Input/Output:
//** (2) COMMAND : Action to be taken, or action taken
//** - ADD: Add an entry
//** - DEL: Delete an entry
//** - DIS: Retrieve an entry
//** - UPD: Update an entry
//** (3) LAST_NAME : Operand for, or result of, COMMAND
//** (4) FIRST_NAME: " " " "
//** (5) EXTENSION : " " " "
//** (6) ZIP-CODE : " " " "
//** - Output only:
//** (7) AIBRETRN : Return code from IMS AIB
//** (8) AIBREASN : Reason code from IMS AIB
//** (9) ERROR-CALL: DL/I command executed
//**
//** Dependencies:
//** (1) Run this job at the server site before running sample job
//** DSNTEJ62 at the client site
//** (2) The server site must have an IMS subsystem running IMS/ESA V6
//** or a subsequent release
//** (3) This IMS subsystem must have the following IMS IVP parts
//** available
//** (A) DFSIVD1, the IMS IVP telephone directory database
//** (B) DFSIVP64, the IMS IVP Cobol PSB for BMP access to DFSIVD1
//** (4) Specify the id for this IMS subsystem in DB2 sample job
//** DSNTEJ62, step PH062S03
//** (5) The server site must also have a WLM environment started by
//** a proc that references the IMS reslib in both the STEPLIB DD
//** and the DFSRESLB DD. See the DB2 Installation Guide for more
//** information.
//** (6) Before running this job, verify that this WLM environment is
//** the one specified in the CREATE PROCEDURE statement in step
//** PH061S01.
//**
//** Change Activity =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938

```

```

/*
*****JOBLIB DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
// DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
// DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
// DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
*/
/*
*****STEP 1: Drop the sample ODBA stored procedure, DSN8.DSN8EC1
*****PH061S01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
 LIB('DSN!!0.RUNLIB.LOAD') -
 PARM('RC0')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';

 DROP PROCEDURE DSN8.DSN8EC1 RESTRICT;

/*
*****STEP 2: Create the sample ODBA stored procedure, DSN8.DSN8EC1
*****PH061S02 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
 LIB('DSN!!0.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';

CREATE PROCEDURE
 DSN8.DSN8EC1(
 IN CHAR(8) CCSID EBCDIC,
 INOUT CHAR(8) CCSID EBCDIC,
 INOUT CHAR(10) CCSID EBCDIC,
 INOUT CHAR(10) CCSID EBCDIC,
 INOUT CHAR(10) CCSID EBCDIC,
 INOUT CHAR(7) CCSID EBCDIC,
 OUT INT,
 OUT INT,
 OUT CHAR(4) CCSID EBCDIC)
FENCED
RESULT SETS 0
EXTERNAL NAME DSN8EC1
LANGUAGE COBOL
PARAMETER STYLE GENERAL
NOT DETERMINISTIC
NO SQL
NO DBINFO
NO COLLID
WLM ENVIRONMENT WLMENV
ASUTIME LIMIT 50
STAY RESIDENT NO
PROGRAM TYPE MAIN
SECURITY DB2
RUN OPTIONS 'TRAP(OFF),RPTOPTS(OFF),TERMTHDAC((QUIET),NONOVR)'
 COMMIT ON RETURN NO;
*/
/*
** Step 3: Pre-compile, compile, and link-edit the stored procedure
*****PH061S03 EXEC DSNHICOB,MEM=DSN8EC1,
// COND=(4,LT),
// PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
// NOXREF,'SQL(DB2)','DEC(31)'),
// PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8EC1),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8EC1),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8EC1),

```

```

// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
NAME DSN8EC1(R)
/*
***** Step 4: Bind the stored procedure package
/* Note: This step is commented out for sample stored
/* procedure DSN8EC1 because it contains no SQL
/* statements. If your stored procedure contains
/* SQL statements, you must bind it as a package.

//**PH061S04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//**DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA,
//** DISP=SHR
//**SYSTSPRT DD SYSOUT=*
//**SYSPRINT DD SYSOUT=*
//**SYSUDUMP DD SYSOUT=*
//**SYSTSIN DD *
//** DSN SYSTEM(DSN)
//** BIND PACKAGE(DSN8EC!!) MEMBER(DSN8EC1) APPLCOMPAT(V!!R1) +
//** ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
/*

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ62

Este JCL prepara y ejecuta un programa de aplicación de muestra, DSN8EC2, que demuestra cómo llamar a un procedimiento almacenado de Db2 para IMS ODBA.

```

***** Name = DSNTEJ62
/*
/* Descriptive Name = DB2 Sample Application
/* Phase 6
/* Sample Client: Stored procedure for IMS ODBA
/* Cobol Language
/*
/*
/* LICENSED MATERIALS - PROPERTY OF IBM
/* 5650-DB2
/* (C) COPYRIGHT 1982, 2016 IBM Corp. All Rights Reserved.
/*
/* STATUS = Version 12
/*
/* Function = This JCL prepares and executes a sample application
/* program, DSN8EC2, that demonstrates how to call a DB2
/* stored procedure for IMS ODBA. The results are
/* directed to the SYSOUT DD.
/*
/* DSN8EC2 accepts a runtime parameter in step PH062S03
/* that specifies -both- the DB2 server location name
/* where the stored procedure is registered -and- the id
/* of the IMS subsystem where the ODBA activity is to
/* occur. You must modify this job to provide the IMS
/* subsystem id. See step PH062S03 for details.
/*
/* Dependencies:
/* (1) Run sample job DSNTEJ61 at the server site before running this
/* job; DSNTEJ61 prepares the sample stored proc for IMS ODBA
/* (2) Modify this job as directed in step PH062S03
/* (3) Run this job at the client site
/*
/*
/* Change activity =
/* 08/18/2014 Single-phase migration s21938_inst1 s21938
/*
/*
//JOBLIB DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
// DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
// DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
// DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
/*
/*
/* Step 1: Pre-compile, compile, and link-edit the client program

```

```

//*****
//PH062S01 EXEC DSNHICOB,MEM=DSN8EC2,
// COND=(4,LT),
// PARM.PC='HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
// NOXREF,'SQL(DB2)','DEC(31)'),
// PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)'),
// PARM.LKED='AMODE=31,RMODE=ANY,MAP'
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8EC2),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8EC2),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8EC2),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNELI)
INCLUDE SYSLIB(DSNTIAR)
//*
//*****
//* Step 2: Bind the client program package and plan
//*****
//PH062S02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA,
// DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE(DSN8EC!!) MEMBER(DSN8EC2) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(SAMPLOC.DSN8EC!!) APPLCOMPAT(V!!R1) +
MEMBER(DSN8EC2) ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8EC2) -
PKLIST(DSN8EC!!..DSN8EC2, -
SAMPLOC.DSN8EC!!..DSN8EC2) -
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT BIND, EXECUTE ON PLAN DSN8EC2
TO PUBLIC;
//*
//*****
//** STEP 3: Invoke the client for the IMS ODBA stored procedure
//** Note: The PARMS keyword in the RUN statement below accepts a
//** single argument that specifies the DB2 server location
//** name -and- the IMS subsystem id, in that order and
//** separated by a single blank character.
//**
//** Example: PARMS('SANTA_TERESA_LAB IMSP')
//**
//** Verify that the PARMS keyword below specifies the name
//** of the DB2 server location name where you ran DSNTEJ61.
//**
//** Change the string ?IMSID? to the id of the IMS subsystem
//** where you want the ODBA-directed activity to occur. This
//** subsystem must reside on the same server as the DB2
//** server and must be running IMS/ESA V6 or a subsequent
//** release.
//*
//*****
//PH062S03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
 RUN PROGRAM(DSN8EC2) -
 PLAN(DSN8EC2) -
 PARMS('SAMPLOC ?IMSID?')
END
//*

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ63

Este JCL prepara DSN8ES1, un procedimiento SQL de ejemplo que acepta un número de departamento y devuelve los datos de salario y bonificación de los empleados de ese departamento de la base de datos de ejemplo Db2 .

```
//*****
//** Name = DSNTEJ63
//**
//** Descriptive Name =
//** DB2 Sample Application
//** Phase 6
//** Sample SQL Procedure
//** SQL Procedure Language
//**
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM Corp. All Rights Reserved.
//**
//** STATUS = Version 12
//**
//** Function =
//** This JCL prepares DSN8ES1, a sample SQL procedure that
//** accepts a department number and returns salary and bonus data
//** for employees in that department from the DB2 sample database.
//**
//** Pseudocode =
//** PH063S01 Step Drop objects created by prior runs of this job
//** PH063S02 Step Prepare DSN8ES1 load module from DSN8ES1 src
//** PH063S03 Step Bind DSN8ES1 package in collection DSN8ES!!
//** Register the stored procedure using generated
//** DDL from the precompiler
//** Create the global temporary table required by
//** the result set
//**
//** Dependencies =
//** (1) This job requires the DB2-provided JCL procedure DSNHSQL
//** (2) Run this job prior to running the client job DSNTEJ64
//**
//** Notes =
//**
//** Change Activity =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//**
//*****
//**JOBLIB DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
//** DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
//** DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
//** DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
//**
//**
//***** STEP 1: Drop any pre-existing entries for stored proc DSN8ES1
//** and the global temporary table for its result set
//*****
//**PH063S01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//**SYSTSPRT DD SYSOUT=*
//**SYSTSIN DD *
//** DSN SYSTEM(DSN)
//** RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
//** LIB('DSN!!0.RUNLIB.LOAD') -
//** PARM('RC0')
//**SYSPRINT DD SYSOUT=*
//**SYSUDUMP DD SYSOUT=*
//**SYSIN DD *
//** SET CURRENT SQLID = 'SYSADM';
//
// DROP PROCEDURE DSN8.DSN8ES1 RESTRICT;
// COMMIT;
//
// DROP TABLE DSN8.DSN8ES1_RS_TBL;
// COMMIT;
//
//**
//*****
```

```

//** Step 2: Pre-compile, compile, and link-edit the stored procedure
//***** ****
//PH063S02 EXEC DSNHSQL,MEM=DSN8ES1,
// COND=(4,LT),
// PARM.PC=('HOST(SQL),SOURCE,XREF,MAR(1,72),CCSID(37) ' ,
// 'STDSQL(NO)'),
// PARM.PCC=('HOST(C),SOURCE,XREF,MAR(1,80),CCSID(37) ' ,
// 'TWOPASS,STDSQL(NO)'),
// PARM.C='SOURCE LIST MARGINS(1,80) NOSEQ LO RENT
// LOCALE("SAA") OPTFILE(DD:CCOPTS)'
// PARM.LKED='AMODE=31,RMODE=ANY,MAP,RENT'
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.NEW.SDSNSAMP(DSN8ES1),
// DISP=SHR
//PC.SYSUT2 DD DSN=&&SPDDL,DISP=(,PASS),
// UNIT=SYSDA,SPACE=(TRK,1),
// DCB=(RECFM=FB,LRECL=80)
//PCC.DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8ES1),
// DISP=SHR
//PCC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ES1),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
NAME DSN8ES1(R)
//*
//***** ****
//** STEP 3: Create the global temp table for DSN8ES1's result set
//** Register DSN8ES1 in SYSIBM.SYSROUTINES
//** Bind the package for DSN8ES1
//***** ****
//PH063S03 EXEC PGM=IKJEFT01,DYNAMNBR=20,
// COND=(4,LT)
//DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8ES1),
// DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD') -
PARM('SQLTERM(%)')
BIND PACKAGE(DSN8ES!!) APPLCOMPAT(V!!R1) +
QUALIFIER(DSN8!!0) -
MEMBER(DSN8ES1) -
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
END
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM'
%
CREATE GLOBAL TEMPORARY TABLE
DSN8.DSN8ES1_RS_TBL
(RS_SEQUENCE INTEGER NOT NULL,
RS_EMPLNO CHAR(6) NOT NULL,
RS_FIRSTNAME CHAR(12) NOT NULL,
RS_LASTNAME CHAR(15) NOT NULL,
RS_SALARY DECIMAL(9, 2) NOT NULL,
RS_BONUS DECIMAL(9, 2) NOT NULL)
CCSID EBCDIC
%
// DD DSN=&&SPDDL,DISP=(OLD,DELETE) <- From preceding step
//*

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO" en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

### DSNTEJ64

Este JCL prepara y ejecuta DSN8ED3, una llamada de muestra para el procedimiento SQL de muestra DSN8ES1.

```

//***** ****
//** Name = DSNTEJ64
//**

```

```

/** Descriptive Name =
/** DB2 Sample Application
/** Phase 6
/** Sample Caller for sample SQL Procedure DSN8ES1
/** C Language
/** LICENSED MATERIALS - PROPERTY OF IBM
/** 5650-DB2
/** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
/** Status = VERSION 12
/** Function =
/** This JCL prepares and executes DSN8ED3, a sample caller for the
/** sample SQL procedure DSN8ES1. DSN8ED3 passes a sample
/** department number to DSN8ES1, then processes what is returned:
/** - Parameters containing:
/** - The total earnings (salaries and bonuses) of employees in
/** that department
/** - The number of employees who got a bonus
/** - A result set containing a row of data (serial no, first and
/** last name, salary, and bonus) for each employee who got a
/** bonus
/** Pseudocode =
/** PH064S01 Step Prepare DSN8ED3 load module from DSN8ED3 src
/** PH064S02 Step Bind DSN8ED3 package in collection DSN8ED!!
/** Bind DSN8ED3 plan from DSN8ED!! and DSN8ES!!
/** collection ids
/** PH064S03 Step Run DSN8ED3 to call stored procedure DSN8ES1
/** Dependencies =
/** (1) Run sample job DSNTEJ63 prior to running this job
/** (2) This job requires the DB2-provided JCL procedure DSNHC
/** Notes =
/** Change Activity =
/** 10/16/2013 Don't use prelinker by default PI13612 DM1812
/** 08/18/2014 Single-phase migration s21938_inst1 s21938
/** ****
//JOBLIB DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
// DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
// DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
// DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
/** ****
/** Step 1: Pre-compile, compile, and link-edit DSN8ED3
/** ****
//PH064S01 EXEC DSNHC,MEM=DSN8ED3,
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE LIST MAR(1,72) LO RENT OPTFILE(DD:CCOPTS)',
// PARM.LKED='AMODE=31,RMODE=ANY,MAP,NORENT,UPCASE'
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8ED3),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8ED3),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED3),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNELI)
INCLUDE SYSLIB(DSNTIAR)
/** ****
/** Step 2: Bind DSN8ED3's PLAN from its package and DSN8ES1's pkg
/** ****
//PH064S02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA,
// DISP=SHR
//SYSTSPPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT BIND, EXECUTE ON PLAN DSN8ED3
 TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DSN)

```

```

BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED3) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(SAMPLOC.DSN8ED!!) APPLCOMPAT(V!!R1) +
MEMBER(DSN8ED3) ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8ED3) -
PKLIST(DSN8ED!!..DSN8ED3, -
SAMPLLOC.DSN8ED!!..DSN8ED3, -
SAMPLLOC.DSN8ES!!..DSN8ES1) -
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
/*
//*****STEP 3: Get Bonus and Salary report for department D11
//*****PH064S03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
RUN PROGRAM(DSN8ED3) PLAN(DSN8ED3) PARMS('/D11 SAMPLOC')
END
/*

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ65

Este JCL hace lo siguiente.

```

//*****Name = DSNTEJ65
//*
//** Descriptive Name =
//** DB2 Sample Application
//** Phase 6
//** - Sample C Caller for DB2 SQL Procedures Processor (DSNTPSMP)
//** - Sample SQL Procedure for DSNTPSMP to prepare
//** - Sample C Caller for SQL Procedure prepared by DSNTPSMP
//*
//*
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
//*
//** STATUS = VERSION 12
//*
//** Function =
//** This JCL does the following:
//** (1) Prepares and binds DSN8ED4, a sample caller for DSNTPSMP,
//** the DB2 Stored Procedures Processor.
//** (2) Invokes DSN8ED4 to prequalify that the server has DSNTPSMP
//** at the proper interface level supported by the this client
//** (3) Invokes DSN8ED4 to pass a sample SQL Procedure, DSN8.DSN8ES2,
//** to DSNTPSMP for preparation
//** (4) Prepares, binds, and executes DSN8ED5, a sample caller for
//** DSN8.DSN8ES2
//*
//** Pseudocode =
//** PH065S01 Step Prepare DSN8ED4 (sample caller of DSNTPSMP)
//** PH065S02 Step Bind DSN8ED4
//** PH065S03 Step Call DSN8ED4 to request DSNTPSMP QUERYLEVEL
//** PH065S04 Step Call DSN8ED4 to pass DSN8.DSN8ES2 to DSNTPSMP
//** PH065S05 Step Prepare DSN8ED5 (sample caller of DSN8.DSN8ES2)
//** PH065S06 Step Bind DSN8ED5
//** PH065S07 Step Call DSN8ED5 to call DSN8.DSN8ES2
//*
//** Dependencies =
//** (1) Sample program requires DSNTPSMP (the DB2 SQL Procedures
//** Processor)
//*
//** Notes =
//*
//** Change Activity =

```

```

//** 10/16/2013 Don't use prelinker by default PI13612 DM1812
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//**
//***** ****
//JOBLIB DD DISP=SHR,DSN=DSN!!0.SDSNEXIT
// DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
// DD DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
// DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD
//**
//***** ****
//** Step 1: Prepare DSN8ED4, caller for DSNTPSMP
//***** ****
//PH065S01 EXEC DSNHC,MEM=DSN8ED4,
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE LIST MAR(1,72) LO RENT OPTFILE(DD:CCOPTS)',
// PARM.LKED='AMODE=31,RMODE=ANY,MAP,NORENT,UPCASE'
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8ED4),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8ED4),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED4),
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(DSNELI)
// INCLUDE SYSLIB(DSNTIAR)
//**
//***** ****
//** Step 2: Bind DSN8ED4's PLAN
//***** ****
//PH065S02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLMLIB DD DISP=SHR,DSN=DSN!!0.DBRLMLIB.DATA
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
// SET CURRENT SQLID = 'SYSADM';
// GRANT BIND, EXECUTE ON PLAN DSN8ED4
// TO PUBLIC;
//SYSTSIN DD *
// DSN SYSTEM(DSN)
// BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED4) APPLCOMPAT(V!!R1) +
// ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
// BIND PACKAGE(SAMPLOC.DSN8ED!!) MEMBER(DSN8ED4) -
// APPLCOMPAT(V!!R1) +
// ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
// BIND PLAN(DSN8ED4) -
// PKLIST(DSN8ED!!..DSN8ED4, -
// SAMPLOC.DSN8ED!!..DSN8ED4, -
// SAMPLOC.DSNREXCS.DSNREXX) -
// ACTION(REPLACE) RETAIN +
// ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
// RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
// LIB('DSN!!0.RUNLIB.LOAD')
// END
//**
//***** ****
//** STEP 3: Invoke DSN8ED4 to pass a QUERYLEVEL request to DSNTPSMP.
//** This is a prequalification that the server has a DSNTPSMP
//** at the correct Interface level for our DSN8ED4 client.
//** Parms: (*) used as place holders)
//** (1) QUERYLEVEL
//** (2) *
//** (3) *
//** (4) (optional) name of server where DSNTPSMP is to be run
//** Note: DSN8ED4 requires all the same definitions be present
//** as on a BUILD request, even though only the function
//** request QUERYLEVEL is passed to DSNTPSMP.
//***** ****
//PH065S03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
// DSN SYSTEM(DSN)
// RUN PROGRAM(DSN8ED4) PLAN(DSN8ED4) -
// PARMS('QUERYLEVEL * * SAMPLOC')
// END
//PCOPTS DD *
//COPTS DD *

```

```

//PLKDOPTS DD *
//LKEDOPTS DD *
//BINDOPTS DD *
//SQLIN DD *
///*
//REPORT01 DD SYSOUT=*,DCB=(RECFM=FBA)
//REPORT02 DD SYSOUT=*
//REPORT03 DD SYSOUT=*
//REPORT04 DD SYSOUT=*
//REPORT05 DD SYSOUT=*
//REPORT06 DD SYSOUT=*
///*
//***** STEP 4: Invoke DSN8ED4 to pass sample SQL Procedure DSN8.DSN8ES2
//* to DSNTPSMP
//* Parms:
//* (1) operation to be performed by DSNTPSMP
//* (2) schema.name of SQL proc to be prepared by DSNTPSMP
//* (3) SQL authid to be used when calling DSNTPSMP
//* (4) (optional) name of server where DSNTPSMP is to be run
//* Note: Options passed in the PCOPTS, COPTS, PLKDOPTS, and
//* BINDOPTS DDS can span more than one input record.
//* Do not use continuation characters (+ or -) to
//* continue BIND options onto the next BINDOPTS record
//*****
//PH065S04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTGIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSN8ED4) PLAN(DSN8ED4) -
 PARMS('REBUILD DSN8.DSN8ES2 AUTHID SAMPLOC')
 END
//PCOPTS DD *
 SOURCE,XREF,MAR(1,80),STDSQL(NO)
//COPTS DD *
 SOURCE LIST MAR(1,80) NOSEQ LO RENT
//PLKDOPTS DD *
//LKEDOPTS DD *
 AMODE=31,RMODE=ANY,MAP,RENT
//BINDOPTS DD *
 PACKAGE(DSN8ES!!)
 QUALIFIER(DSN8!!0) ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
//SQLIN DD DSN=DSN!!0.NEW.SDSNSAMP(DSN8ES2),
// DISP=SHR
///*
//REPORT01 DD SYSOUT=*,DCB=(RECFM=FBA)
//REPORT02 DD SYSOUT=*
//REPORT03 DD SYSOUT=*
//REPORT04 DD SYSOUT=*
//REPORT05 DD SYSOUT=*
//REPORT06 DD SYSOUT=*
///*
//***** Step 5: Prepare DSN8ED5, sample caller of DSN8.DSN8ES2
//*****
//PH065S05 EXEC DSNHC,MEM=DSN8ED5,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF,
// PARM.C='SOURCE LIST MAR(1,72) LO RENT OPTFILE(DD:CCOPTS)',
// PARM.LKED='AMODE=31,RMODE=ANY,MAP,NORENT,UPCASE'
//PC.DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA(DSN8ED5),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8ED5),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED5),
// DISP=SHR
//LKED.SYSIN DD *
 INCLUDE SYSLIB(DSNELI)
 INCLUDE SYSLIB(DSNTIAR)
///*
//***** Bind DSN8ED5's PLAN
//*****
//PH065S06 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB DD DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*

```

```

//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT BIND, EXECUTE ON PLAN DSN8ED5
TO PUBLIC;
//SYSTGIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED5) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(SAMPLOC.DSN8ED!!) MEMBER(DSN8ED5) -
APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8ED5) -
PKLIST(DSN8ED!!..DSN8ED5, -
SAMPLOC.DSN8ED!!..DSN8ED5, -
SAMPLOC.DSN8ES!!..*) -
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END
/*
//***** STEP 7: Invoke DSN8ED5 to call sample SQL Procedure DSN8.DSN8ES2
//***** STEP 7: Invoke DSN8ED5 to call sample SQL Procedure DSN8.DSN8ES2
//PH065S07 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=**
//SYSTGIN DD *
DSN SYSTEM(DSN)
RUN PROGRAM(DSN8ED5) PLAN(DSN8ED5) -
PARMS('1500.00 SAMPLOC')
END

```

## Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ6W

Este JCL hace lo siguiente.

```

//*
//* DB2 Sample Application
//* Phase 6
//* Sample caller for Stored Procedure WLM_REFRESH
//* IBM C/C++ for z/OS
//*
//* Licensed Materials - Property of IBM
//* 5650-DB2
//* (C) COPYRIGHT 1982, 2016 IBM Corp. All Rights Reserved.
//*
//* STATUS = Version 12
//*
//* Function =
//* This JCL does the following:
//* * Prepares and executes DSN8ED6, a sample caller of the
//* WLM_REFRESH stored procedure. WLM_REFRESH refreshes a
//* WLM environment specified as an input parameter using an
//* authorization ID also specified as an input parameter. The
//* authorization ID must have READ access on a resource profile
//* called !DSN!.WLM_REFRESH.!WLMENV!
//* Use job DSNTIJRA job step DSNTWR to create and permit access
//* to this resource profile.
//* * Optional: Prepares DSNTWR and DSNTWRE, external modules for
//* WLM_REFRESH. These modules are provided in SDSNLOAD so
//* preparing them is required only if you maintain a customized
//* copy of DSNTWRS, the sample source code for DSNTWR, or
//* DSNTWRE, the sample source code for DSNTWRE.
//*
//* Pseudocode =
//* PH06WS00 Step Optional: Prepare DSNTWRE, a program for
//* getting the DB2 Environment Info Block (EIB)
//* -> Uncomment and run this step if you want to
//* override the DB2-supplied DSNTWRE module
//* PH06WS01 Step Optional: Prepare DSNTWR, the external module
//* for SYSPROC.WLM_REFRESH
//* -> Uncomment and run this step if you want to

```

```

//** override the DB2-supplied DSNTWR module
//** PH06WS02 Step Optional: Bind the package for DSNTWR
//** -> Uncomment and run this step only if you
//** also uncomment and run the step PH06WS01
//** PH06WS03 Step Prepare DSN8ED6
//** PH06WS04 Step Bind the plan and package for DSN8ED6
//** PH06WS05 Step Invoke DSN8ED6
//**
//** Dependencies =
//** (1) This job requires the DB2-provided JCL procedures DSNHASM and
//** DSNHC
//** (2) Run this job only after running jobs DSNTIJTM and DSNTIJRT
//** (3) The DSN8ED6 program receives parameters that contain the name
//** of the WLM environment to be refreshed and the authorization
//** ID to be used for the request. The authorization ID must have
//** READ access an a resource profile called
//** !DSN!.WLM_REFRESH.!WLMPENV!
//** Use job DSNTIJRA job step DSNTWR to create and permit access
//** to this resource profile.
//**
//** Notes =
//**
//** Change Activity =
//** 10/16/2013 Don't use prelinker by default PI13612 DM1812
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//**
//***** ****
//JOBLIB DD DISP=SHR,DSN=DSN!!0.SDSNEXIT
// DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
// DD DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
//**
//** /**
//** //** Step 0 (Optional): Prepare DSNTWRE, a program that gets
//** //** the DB2 group attach name
//** //**
//** //PH06WS00 EXEC DSNHASM,COND=(4,LT),
//** // MEM=DSNTWRE,
//** // PARM.PC='HOST(ASM),STDSQL(NO)',
//** // PARM.ASM='RENT,OBJECT,NODECK',
//** // PARM.LKED='LIST,XREF,AMODE=31,RMODE=ANY,RENT'
//** //PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSNTWRE),
//** // DISP=SHR
//** //PC.SYSLIB DD DSN=DSN!!0.SDSNSAMP,
//** // DISP=SHR
//** //PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSNTWRE),
//** // DISP=SHR
//** //ASM.SYSLIB DD DSN=SYS1.MACLIB,
//** // DISP=SHR
//** // DD DSN=CEE.V!R!M!.SCEEMAC,
//** // DISP=SHR
//** // DD DSN=DSN!!0.SDSNMACS,
//** // DISP=SHR
//** // DD DSN=DSN!!0.SDSNSAMP,
//** // DISP=SHR
//** //LKED.SYSLMOD DD DSN=DSN!!0.SDSNEXIT(DSNTWRE),
//** // DISP=SHR
//** //LKED.SYSLIB DD DSN=CEE.V!R!M!.SCEELKED,
//** // DISP=SHR
//** // DD DSN=CEE.V!R!M!.SCEERUN,
//** // DISP=SHR
//** // DD DSN=CEE.V!R!M!.SCEESPC,
//** // DISP=SHR
//** // DD DSN=CEE.V!R!M!.SCEESPCO,
//** // DISP=SHR
//** // DD DSN=DSN!!0.SDSNLOAD,
//** // DISP=SHR
//** //LKED.SYSIN DD *
//** INCLUDE SYSLIB(DSNRLI)
//** NAME DSNTWRE(R)
//** /*
//** /**
//** //** Step 1 (Optional): Prepare DSNTWR, the external module for
//** //** WLM_REFRESH
//** //**
//** //PH06WS01 EXEC DSNHASM,COND=(4,LT),
//** // MEM=DSNTWR,
//** // PARM.PC='HOST(ASM),STDSQL(NO)',
//** // PARM.ASM='RENT,OBJECT,NODECK',
//** // PARM.LKED='LIST,XREF,AMODE=31,RMODE=ANY,RENT'
//** //PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSNTWR),
//** // DISP=SHR

```

```

//*/ //PC.SYSLIB DD DSN=DSN!!0.SDSNSAMP,
//*/ //
//*/ //PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSNTWRS),
//*/ //
//*/ //ASM.SYSLIB DD DSN=SYS1.MACLIB,
//*/ //
//*/ //DISP=SHR
//*/ //DD DSN=CEE.V!R!M!.SCEEMAC,
//*/ //
//*/ //DISP=SHR
//*/ //DD DSN=DSN!!0.SDSNMACS,
//*/ //
//*/ //DISP=SHR
//*/ //DD DSN=DSN!!0.SDSNSAMP,
//*/ //
//*/ //DISP=SHR
//*/ //DD DSN=CEE.V!R!M!.SCEELKED,
//*/ //
//*/ //DISP=SHR
//*/ //DD DSN=CEE.V!R!M!.SCEERUN,
//*/ //
//*/ //DISP=SHR
//*/ //DD DSN=CEE.V!R!M!.SCEESPC,
//*/ //
//*/ //DISP=SHR
//*/ //DD DSN=CEE.V!R!M!.SCEESPCO,
//*/ //
//*/ //DISP=SHR
//*/ //DD DSN=DSN!!0.SDSNLOAD,
//*/ //
//*/ //DISP=SHR
//*/ //LKED.SYSIN DD *
//*/ INCLUDE SYSLIB(DSNRLI)
//*/ SETCODE AC(1)
//*/ NAME DSNTWR(R)
//*/ /*
//*/ /**
//*/ // Step 2 (Optional): Bind the package for DSNTWR
//*/ /**
//*/ //PH06WS02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//*/ //DBRMLIB DD DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA
//*/ //SYSTSPRT DD SYSOUT=*
//*/ //SYSPRINT DD SYSOUT=*
//*/ //SYSUDUMP DD SYSOUT=*
//*/ //SYSTSIN DD *
//*/ DSN SYSTEM(DSN)
//*/ BIND PACKAGE(DSNTWR) MEMBER(DSNTWR) APPLCOMPAT(V!!R1) +
//*/ ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
//*/ END
//*/ /*
//*/ /**
//*/ // Step 3: Prepare DSN8ED6, sample caller of WLM_REFRESH
//*/ /**
//*/ //PH06WS03 EXEC DSNHC,MEM=DSN8ED6,COND=(4,LT),
//*/ // PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
//*/ // SOURCE,XREF',
//*/ // PARM.C='SOURCE LIST MAR(1,72) LO RENT OPTFILE(DD:CCOPTS)'
//*/ // PARM.LKED='AMODE=31,RMODE=ANY,MAP,RENT,UPCASE'
//*/ //PC.DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA(DSN8ED6),
//*/ // DISP=SHR
//*/ //PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
//*/ // DISP=SHR
//*/ //PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8ED6),
//*/ // DISP=SHR
//*/ //LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED6),
//*/ // DISP=SHR
//*/ //LKED.SYSIN DD *
//*/ INCLUDE SYSLIB(DSNELI)
//*/ INCLUDE SYSLIB(DSNTIAR)
//*/ /*
//*/ /**
//*/ // Step 4: Bind the package and plan for DSN8ED6
//*/ /**
//*/ //PH06WS04 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//*/ //DBRMLIB DD DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA
//*/ //SYSTSPRT DD SYSOUT=*
//*/ //SYSPRINT DD SYSOUT=*
//*/ //SYSUDUMP DD SYSOUT=*
//*/ //SYSTSIN DD *
//*/ DSN SYSTEM(DSN)
//*/ BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED6) APPLCOMPAT(V!!R1) +
//*/ ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
//*/ BIND PLAN(DSN8ED6) -
//*/ PKLIST(DSN8ED!!..DSN8ED6, -
//*/ DSNTWR.DSNTWR) -
//*/ ACTION(REPLACE) RETAIN +
//*/ ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
//*/ RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
//*/ LIB('DSN!!0.RUNLIB.LOAD')
//*/ END

```

```

//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 GRANT BIND, EXECUTE ON PLAN DSN8ED6
 TO PUBLIC;
/*
/* Step 5: Invoke DSN8ED6
/*
//PH06WS05 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSN8ED6) PLAN(DSN8ED6) -
 LIB('DSN!!0.RUNLIB.LOAD') -
 PARMS('!WLMENV! !DSN! !ID!')
END
/*

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ6Z

Este JCL prepara y ejecuta DSN8ED7, una llamada de muestra de ADMIN\_INFO\_SYSPARM, un procedimiento almacenado proporcionado por Db2 que devuelve la configuración actual de los parámetros del subsistema Db2 .

```

/*
/** DB2 Sample Application
/** Phase 6
/** Sample Caller of Stored Procedure - ADMIN_INFO_SYSPARM
/** C language
/**
/** LICENSED MATERIALS - PROPERTY OF IBM
/** 5650-DB2
/** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
/**
/** Status = VERSION 12
/**
/** Function =
/** This JCL prepares and executes DSN8ED7, a sample caller of
/** ADMIN_INFO_SYSPARM, a DB2-provided stored procedure that returns
/** the current settings of your DB2 subsystem parameters. After
/** calling ADMIN_INFO_SYSPARM, DSN8ED7 formats the results in a
/** report format.
/**
/** Pseudocode =
/** PH06ZS01 Step Prepare DSN8ED7
/** PH06ZS02 Step Bind the plan and package for DSN8ED7
/** PH06ZS03 Step Invoke DSN8ED7
/**
/** Dependencies =
/** - This job requires the DB2-provided JCL procedure DSNHC
/**
/** Notes =
/**
/** Change Activity =
/** 10/16/2013 Don't use prelinker by default PI13612 DM1812
/** 08/18/2014 Single-phase migration s21938_inst1 s21938
/**
//***** ****
/**
//JOBLIB DD DISP=SHR,DSN=DSN!!0.SDSNEXIT
// DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
// DD DISP=SHR,DSN=CEE.V!R!M!.SCEERUN
/**
/** Step 1: Prepare DSN8ED7, sample caller of ADMIN_INFO_SYSPARM
/**
//PH06ZS01 EXEC DSNHC,MEM=DSN8ED7,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF',
// PARM.C='SOURCE LIST MAR(1,72) LO_RENT OPTFILE(DD:CCOPTS)',
// PARM.LKED='AMODE=31,RMODE=ANY,MAP,RENT,REUS,UPCASE'
//PC.DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8ED7),

```

```

// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8ED7),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED7),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNELI)
INCLUDE SYSLIB(DSNTIAR)
/*
/* Step 2: Bind the package and plan for DSN8ED7
/*
//PH06ZS02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB DD DISP=SHR,DSN=DSN!!0.DBRMLIB.DATA
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED7) APPLCOMPAT(V!!R1) +
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
 BIND PLAN(DSN8ED7) -
 PKLIST(DSN8ED!!..DSN8ED7) -
 ACTION(REPLACE) RETAIN +
 ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
 LIB('DSN!!0.RUNLIB.LOAD')
END
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 GRANT BIND, EXECUTE ON PLAN DSN8ED7
 TO PUBLIC;
/*
/* Step 3: Invoke DSN8ED7
/*
//PH06ZS03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSN8ED7) PLAN(DSN8ED7) -
 LIB('DSN!!0.RUNLIB.LOAD')
END
/*

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ66

Este JCL hace lo siguiente.

```

//*****
//** Name = DSNTEJ66
//**
//** Descriptive Name = DB2 Sample Application - Native SQL Procedure
//** Phase 6
//**
//**
//** Licensed Materials - Property of IBM
//** 5650-DB2
//** (C) COPYRIGHT 2006, 2016 IBM Corp. All Rights Reserved.
//**
//** STATUS = Version 12
//**
//** Function = This JCL does the following:
//** - Creates a sample native SQL procedure called
//** DSN8.DSN8ES3 that generates and returns (by result
//** set) a CREATE PROCEDURE statement for a given stored
//** procedure.
//** - Prepares and executes a sample caller of DSN8ES3
//** called DSN8ED9.
//** - Shows how to use ALTER PROCEDURE ... ADD VERSION to
//** create a version V2 of DSN8ES3 that does the same
//** thing as the original version but also adds a
//**

```

```

/*
 terminating semicolon at the end of the generated
 CREATE PROCEDURE statement
 - Shows how to ALTER ACTIVATE version V2 to make it
 the active version of DSN8ES3
 - Shows how to DEPLOY DSN8ES3 at a remote site
*/
/*
 Restrictions =
 As part of the setup to DEPLOY DSN8ES3, the DSNTEP2 application
 needs to be able to connect to the remote site.
*/
/*
 Notice =
*/
/*
 Pseudocode =
 PH066S01 Step Drop objects created by prior runs of this job
 PH066S02 Step Create the global temporary table for
 DSN8.DSN8ES3
 PH066S03 Step Prepare DSN8ES3 as a native SQL procedure
 -> Also generates a package called
 DSN8.DSN8ES3
 PH066S04 Step Prepare DSN8ED9, sample caller for the DSN8ES3
 SQL proc
 PH066S05 Step Bind the plan for DSN8ED9
 PH066S06 Step Execute DSN8ED9 to request a CREATE PROC
 statement for the stored procedure
 SYSPROC.DSNUTILS
 PH066S07 Step Create a work copy of the DSN8ES3 source code
 PH066S08 Step Use TSO edit to modify the work copy into an
 ALTER PROCEDURE that will make a trivial
 change to DSN8ES3 as VERSION V2
 -> The generated CREATE PROC statement will be
 terminated by a semicolon
 PH066S09 Step Save the work copy as DSN8ES3 in
 DSN!!0.NEW.SDSNSAMP
 PH066S10 Step Process the ALTER PROCEDURE DSN8ES3 to ADD
 VERSION V2
 -> Also generates a package called
 DSN8.DSN8ES3 (VERSION V2)
 PH066S11 Step Activate V2 as the current version of DSN8ES3
 PH066S12 Step Execute DSN8ED9 to request a CREATE PROC
 statement for SYSPROC.DSNUTILU
 -> When using DSN8ES3 V2, it's terminated by a
 semicolon
 PH066S13 Step Setup to DEPLOY DSN8ES3: Create a global
 temporary table on the remote server
 -> To rerun this step, uncomment the DROP
 and COMMIT statements
 PH066S14 Step DEPLOY DSN8ES3 on the remote server
 PH066S15 Step Bind the plan for DSN8ED9 on the remote server
 PH066S16 Step Execute DSN8ED9 to request a CREATE PROC
 statement for SYSPROC.DSNUTILS at the remote
 site
*/
/*
 Change Activity =
 10/16/2013 Don't use prelinker by default PI13612 DM1812
 08/18/2014 Single-phase migration s21938_inst1 s21938
*/
//*****
//*
//JOBLIB DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
// DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
// DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
//*
//** Step 1: Drop objects created by prior runs of this job
//**
//PH066S01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) +
 PLAN(DSNTIA!!) +
 LIB('DSN!!0.RUNLIB.LOAD') +
 PARM('RC0')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 DROP PROCEDURE DSN8.DSN8ES3;
 COMMIT;
 DROP TABLE DSN8.DSN8ES3_RS_TBL;
 COMMIT;
//WORKCOPY DD DSN=DSN!!0.DSN8.DSN8ES3.WORKCOPY,
// DISP=(MOD,DELETE),

```

```

// UNIT=SYSDA,SPACE=(TRK,0)
///*
//** Step 2: Create the global temporary table for DSN8.DSN8ES3
//*/
//PH066S02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) +
 PLAN(DSNTIA!!) +
 LIB('DSN!!0.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 CREATE GLOBAL TEMPORARY TABLE
 DSN8.DSN8ES3_RS_TBL
 (RS_SEQUENCE INTEGER NOT NULL,
 RS_LINE CHAR(80) NOT NULL)
 CCSID EBCDIC
///*
//** Step 3: Prepare DSN8ES3 as a native SQL procedure
//** -> Also generates a package called DSN8.DSN8ES3
//*/
//PH066S03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTEP2) PLAN(DSNTEP!!) +
 LIB('DSN!!0.RUNLIB.LOAD') PARMS('/SQLTERM(%)')
END
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM'
%
// DD DISP=SHR,
// DSN=DSN!!0.SDSNSAMP(DSN8ES3)
// DD *
%
///*
//** Step 4: Prepare DSN8ED9, sample caller for the DSN8ES3 SQL proc
//*/
//PH066S04 EXEC DSNHC,MEM=DSN8ED9,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF',
// PARM.C='SOURCE LIST MAR(1,72) LO RENT OPTFILE(DD:CCOPTS)',
// PARM.LKED='AMODE=31,RMODE=ANY,MAP,NORENT,UPCASE'
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8ED9),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8ED9),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8ED9),
// DISP=SHR
//LKED.SYSIN DD *
 INCLUDE SYSLIB(DSNELI)
 INCLUDE SYSLIB(DSNTIAR)
///*
//** Step 5: Bind the plan for DSN8ED9
//*/
//PH066S05 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA,
// DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 GRANT BIND, EXECUTE ON PLAN DSN8ED9
 TO PUBLIC;
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 BIND PACKAGE(DSN8ED!!) MEMBER(DSN8ED9) APPLCOMPAT(V!!R1) +
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
 BIND PLAN(DSN8ED9) +
 PKLIST(DSN8ED!!..DSN8ED9) +
 ACTION(REPLACE) RETAIN +
 ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) +
 LIB('DSN!!0.RUNLIB.LOAD')

```

```

/*
/* Step 6: Execute DSN8ED9 to request a CREATE PROC statement
/* for the stored procedure named SYSPROC.DSNUTILS
*/
//PH066S06 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
 RUN PROGRAM(DSN8ED9) PLAN(DSN8ED9) +
 LIB('DSN!!0.RUNLIB.LOAD') +
 PARMS('/SYSPROC DSNUTILS')
END
/*
/* Step 7: Create a work copy of the DSN8ES3 source code
*/
//PH066S07 EXEC PGM=IEBGENER,COND=(4,LT)
//SYSIN DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DISP=SHR,
// DSN=DSN!!0.SDSENSAMP(DSN8ES3)
//SYSUT2 DD DSN=DSN!!0.DSN8.DSN8ES3.WORKCOPY,
// DISP=(,CATLG,DELETE),
// UNIT=SYSDA,
// SPACE=(TRK,1),
// DCB=(RECFM=FB,LRECL=80)
/*
/* Step 8: Use TSO edit to modify the work copy into an ALTER PROC-
/* DURE that will make a trivial change to DSN8ES3 VERSION V2
/* -> The generated CREATE PROC statement will now be
/* terminated by a semicolon
*/
//PH066S08 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 EDIT 'DSN!!0.DSN8.DSN8ES3.WORKCOPY' +
 DATA OLD NONUM NORECOVER ASIS
 FIND /CREATE PROCEDURE DSN8.DSN8ES3/
 CHANGE * 1 /CREATE PROCEDURE/ALTER PROCEDURE /
 INSERT ADD VERSION V2
 FIND /PARAMETER CCSID EBCDIC/
 DELETE * 1
 FIND /U100: -- Finish up/
 CHANGE * 1 /Finish up /Add terminating semicolon/
 INSERT SET LINE = ';';
 INSERT SET RETURN_POINT = 'DONE';
 INSERT GOTO INSERTLINE;
 LIST 1 9999
 END SAVE
/*
/* Step 9: Save in DSN!!0.NEW.SDSENSAMP
*/
//PH066S09 EXEC PGM=IEBGENER,COND=(4,LT)
//SYSIN DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DISP=(OLD,DELETE),
// DSN=DSN!!0.DSN8.DSN8ES3.WORKCOPY
//SYSUT2 DD DISP=SHR,
// DSN=DSN!!0.NEW.SDSENSAMP(DSN8ES3)
/*
/* Step 10: Process the ALTER PROCEDURE DSN8ES3 to ADD VERSION V2
/* -> Also generates a package called DSN8.DSN8ES3 (V2)
*/
//PH066S10 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTEP2) PLAN(DSNTEP!!) +
 LIB('DSN!!0.RUNLIB.LOAD') PARMS('/SQLTERM(%)')
END
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM'
 %
// DD DISP=SHR,
// DSN=DSN!!0.NEW.SDSENSAMP(DSN8ES3)
// DD *
 %

```

```

/*
/** Step 11: Activate V2 as the current version of DSN8ES3
*/
//PH066S11 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) +
 LIB('DSN!!0.RUNLIB.LOAD')
 END
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 ALTER PROCEDURE DSN8.DSN8ES3 ACTIVATE VERSION V2;
/*
/** Step 12: Execute DSN8ED9 to request a CREATE PROC statement
** for SYSPROC.DSNUTILU
** -> When using DSN8ES3 V2, it's terminated by a semicolon
*/
//PH066S12 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 REBIND PACKAGE(DSN8ED!!..DSN8ED9) APPLCOMPAT(V!!R1) +
 PLANMGMT(OFF)
 RUN PROGRAM(DSN8ED9) PLAN(DSN8ED9) +
 LIB('DSN!!0.RUNLIB.LOAD') +
 PARMS('/SYSPROC DSNUTILU')
 END
/*
/** Step 13: Setup to DEPLOY DSN8ES3 - Create a global temporary
** table on the remote server
*/
//PH066S13 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSNTEP2) +
 PLAN(DSNTEP!!) +
 LIB('DSN!!0.RUNLIB.LOAD')
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 CONNECT TO SAMPLOC;
* DROP TABLE DSN8.DSN8ES3_RS_TBL;
* COMMIT;
 CREATE GLOBAL TEMPORARY TABLE
 DSN8.DSN8ES3_RS_TBL
 (RS_SEQUENCE INTEGER NOT NULL,
 RS_LINE CHAR(80) NOT NULL)
 CCSID EBCDIC;
/*
/** Step 14: DEPLOY DSN8ES3 on the remote server
*/
//PH066S14 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 BIND PACKAGE(SAMPLOC.DSN8) APPLCOMPAT(V!!R1) +
 DEPLOY(DSN8.DSN8ES3) +
 COPYVER(V2) +
 ACTION(REP)
/*
/** Step 15: Bind the plan for DSN8ED9 on the remote server
*/
//PH066S15 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA,
// DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 GRANT BIND, EXECUTE ON PLAN DSN8ED9
 TO PUBLIC;
//SYSTSIN DD *

```

```

DSN SYSTEM(DSN)
BIND PACKAGE(SAMPLOC.DSN8ED!!) MEMBER(DSN8ED9) +
 APPLCOMPAT(V!!R1) +
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8ED9) +
 PKLIST(DSN8ED!!..DSN8ED9, +
 SAMPLOC.DSN8ED!!..DSN8ED9, +
 SAMPLOC.DSN8ES!!..*) +
 ACTION(REPLACE) RETAIN +
 ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) +
 LIB('DSN!!0.RUNLIB.LOAD')
/*
/* Step 16: Execute DSN8ED9 to request a CREATE PROC statement for
/* SYSPROC.DSNUTILS at the remote site
/*
//PH066S16 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
 RUN PROGRAM(DSN8ED9) PLAN(DSN8ED9) +
 LIB('DSN!!0.RUNLIB.LOAD') +
 PARMS('/SYSPROC DSNUTILS SAMPLOC')
END
/*

```

### Referencia relacionada

["Ejemplos de aplicaciones en TSO"](#) en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ2U

ESTA JCL PREPARA LAS SIGUIENTES FUNCIONES DEFINIDAS POR EL USUARIO (Db2 , UDF) Y UN PROGRAMA CONTROLADOR PARA INVOCARLAS.

```

//*****
//** NAME = DSNTEJ2U
//**
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 2
//** USER DEFINED FUNCTIONS (C/C++)
//**
//** Licensed Materials - Property of IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM Corp. All Rights Reserved.
//**
//** STATUS = Version 12
//**
//** FUNCTION = THIS JCL PREPARES THE FOLLOWING DB2 USER-DEFINED
//** FUNCTIONS (UDF'S) AND A DRIVER PROGRAM TO INVOKE THEM.
//**
//** NOTES = ENSURE THAT LINE NUMBER SEQUENCING IS SET 'ON' IF
//** THIS JOB IS SUBMITTED FROM AN ISPF EDIT SESSION
//**
//** THIS JOB IS RUN AFTER PHASE 1.
//**
//** CHANGE ACTIVITY =
//** 10/16/2013 Don't use prelinker by default PI13612 DM1812
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//**
//*****
//JOBLIB DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
// DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
// DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
// DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
//**
//** STEP 1: DROP ANY EXISTING DB2 SAMPLE UDF'S
//**
//PH02US01 EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
RUN PROGRAM(DSNTIAD) -
 PLAN(DSNTIA!!) -
 LIB('DSN!!0.RUNLIB.LOAD') -

```

```

PARM('RC0')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';

DROP SPECIFIC FUNCTION DSN8.DSN8DUCDDVV RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8DUCDVVV RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8DUADV RESTRICT;

DROP SPECIFIC FUNCTION DSN8.DSN8DUCTTVV RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8DUCTVV RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8DUATV RESTRICT;

DROP SPECIFIC FUNCTION DSN8.DSN8DUCYFV RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8DUCYFVV RESTRICT;

DROP SPECIFIC FUNCTION DSN8.DSN8EUDND RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8EUDNV RESTRICT;

DROP SPECIFIC FUNCTION DSN8.DSN8EUMND RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8EUMNV RESTRICT;

DROP SPECIFIC FUNCTION DSN8.DSN8DUTINV RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8DUTINVV RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8DUTINVVV RESTRICT;

DROP SPECIFIC FUNCTION DSN8.DSN8DUTISV RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8DUTISVV RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8DUTISVVV RESTRICT;

DROP SPECIFIC FUNCTION DSN8.DSN8DUTILV RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8DUTILVV RESTRICT;
DROP SPECIFIC FUNCTION DSN8.DSN8DUTILVVV RESTRICT;

DROP SPECIFIC FUNCTION DSN8.DSN8DUWFV RESTRICT;
/* STEP 2: DEFINE SAMPLE UDF'S TO DB2
*/
//PH02US02 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
RUN PROGRAM(DSNTIAD)
 PLAN(DSNTIA!!) -
 LIB('DSN!!0.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';

CREATE FUNCTION
 DSN8.ALTDAT(
 VARCHAR(13) CCSID EBCDIC)
RETURNS
 VARCHAR(17) CCSID EBCDIC
 SPECIFIC DSN8.DSN8DUADV
 LANGUAGE C
 DETERMINISTIC
 NO SQL
 EXTERNAL NAME DSN8DUAD
 PARAMETER STYLE DB2SQL
 NULL CALL
 NO EXTERNAL ACTION
 NO SCRATCHPAD
 NO FINAL CALL
 ALLOW PARALLEL
 NO COLLID
 ASUTIME LIMIT 10
 STAY RESIDENT NO
 PROGRAM TYPE SUB
 WLM ENVIRONMENT WLMENV
 SECURITY DB2
 NO DBINFO;

CREATE FUNCTION
 DSN8.ALTDAT(
 VARCHAR(17) CCSID EBCDIC,
 VARCHAR(13) CCSID EBCDIC,
 VARCHAR(13) CCSID EBCDIC)
RETURNS
 VARCHAR(17) CCSID EBCDIC

```

```

SPECIFIC DSN8.DSN8DUCDVV
LANGUAGE C
DETERMINISTIC
NO SQL
EXTERNAL NAME DSN8DUCD
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
NO COLLID
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE SUB
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

CREATE FUNCTION
DSN8.ALTDATE(
 DATE,
 VARCHAR(13) CCSID EBCDIC,
 VARCHAR(13) CCSID EBCDIC)
RETURNS
 VARCHAR(17) CCSID EBCDIC
SPECIFIC DSN8.DSN8DUCDDVV
SOURCE SPECIFIC DSN8.DSN8DUCDVV;

CREATE FUNCTION
DSN8.ALTTIME(
 VARCHAR(14) CCSID EBCDIC)
RETURNS
 VARCHAR(11) CCSID EBCDIC
SPECIFIC DSN8.DSN8DUATV
LANGUAGE C
DETERMINISTIC
NO SQL
EXTERNAL NAME DSN8DUAT
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
NO COLLID
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE SUB
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

CREATE FUNCTION
DSN8.ALTTIME(
 VARCHAR(11) CCSID EBCDIC,
 VARCHAR(14) CCSID EBCDIC,
 VARCHAR(14) CCSID EBCDIC)
RETURNS
 VARCHAR(11) CCSID EBCDIC
SPECIFIC DSN8.DSN8DUCTVVV
LANGUAGE C
DETERMINISTIC
NO SQL
EXTERNAL NAME DSN8DUCT
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
NO COLLID
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE SUB
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

CREATE FUNCTION
DSN8.ALTTIME(
 TIME,

```

```

 VARCHAR(14) CCSID EBCDIC,
 VARCHAR(14) CCSID EBCDIC)
RETURNS
 VARCHAR(11) CCSID EBCDIC
SPECIFIC DSN8.DSN8DUCTVV
SOURCE SPECIFIC DSN8.DSN8DUCTVV;

CREATE FUNCTION
DSN8.CURRENCY(
 FLOAT,
 VARCHAR(2) CCSID EBCDIC)
RETURNS
 VARCHAR(19) CCSID EBCDIC
SPECIFIC DSN8.DSN8DUCYFV
LANGUAGE C
DETERMINISTIC
NO SQL
EXTERNAL NAME DSN8DUCY
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
NO COLLID
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE MAIN
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

CREATE FUNCTION
DSN8.CURRENCY(
 FLOAT,
 VARCHAR(2) CCSID EBCDIC,
 VARCHAR(5) CCSID EBCDIC)
RETURNS
 VARCHAR(19) CCSID EBCDIC
SPECIFIC DSN8.DSN8DUCYFVV
LANGUAGE C
DETERMINISTIC
NO SQL
EXTERNAL NAME DSN8DUCY
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
NO COLLID
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE MAIN
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

CREATE FUNCTION
DSN8.DAYNAME(
 VARCHAR(10) CCSID EBCDIC)
RETURNS
 VARCHAR(9) CCSID EBCDIC
SPECIFIC DSN8.DSN8EUDNV
LANGUAGE C
DETERMINISTIC
NO SQL
EXTERNAL NAME DSN8EUDN
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
NO COLLID
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE SUB
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

```

```

CREATE FUNCTION
 DSN8.DAYNAME(
 DATE)
 RETURNS
 VARCHAR(9) CCSID EBCDIC
 SPECIFIC DSN8.DSN8EUDND
 SOURCE SPECIFIC DSN8.DSN8EUDNV;

CREATE FUNCTION
 DSN8.MONTHNAME(
 VARCHAR(10) CCSID EBCDIC)
 RETURNS
 VARCHAR(9) CCSID EBCDIC
 SPECIFIC DSN8.DSN8EUMNV
 LANGUAGE C
 DETERMINISTIC
 NO SQL
 EXTERNAL NAME DSN8EUMN
 PARAMETER STYLE DB2SQL
 NULL CALL
 NO EXTERNAL ACTION
 NO SCRATCHPAD
 NO FINAL CALL
 ALLOW PARALLEL
 NO COLLID
 ASUTIME LIMIT 10
 STAY RESIDENT NO
 PROGRAM TYPE SUB
 WLM ENVIRONMENT WLMENV
 SECURITY DB2
 NO DBINFO;

CREATE FUNCTION
 DSN8.MONTHNAME(
 DATE)
 RETURNS
 VARCHAR(9) CCSID EBCDIC
 SPECIFIC DSN8.DSN8EUMND
 SOURCE SPECIFIC DSN8.DSN8EUMNV;

CREATE FUNCTION
 DSN8.TABLE_NAME(
 VARCHAR(18) CCSID EBCDIC)
 RETURNS
 VARCHAR(18) CCSID EBCDIC
 SPECIFIC DSN8.DSN8DUTINV
 LANGUAGE C
 DETERMINISTIC
 READS SQL DATA
 EXTERNAL NAME DSN8DUTI
 PARAMETER STYLE DB2SQL
 NULL CALL
 NO EXTERNAL ACTION
 NO SCRATCHPAD
 NO FINAL CALL
 ALLOW PARALLEL
 COLLID DSN8DU!!
 ASUTIME LIMIT 10
 STAY RESIDENT NO
 PROGRAM TYPE MAIN
 WLM ENVIRONMENT WLMENV
 SECURITY DB2
 NO DBINFO;

CREATE FUNCTION
 DSN8.TABLE_NAME(
 VARCHAR(18) CCSID EBCDIC,
 VARCHAR(8) CCSID EBCDIC)
 RETURNS
 VARCHAR(18) CCSID EBCDIC
 SPECIFIC DSN8.DSN8DUTINVV
 LANGUAGE C
 DETERMINISTIC
 READS SQL DATA
 EXTERNAL NAME DSN8DUTI
 PARAMETER STYLE DB2SQL
 NULL CALL
 NO EXTERNAL ACTION
 NO SCRATCHPAD
 NO FINAL CALL
 ALLOW PARALLEL
 COLLID DSN8DU!!

```

```

ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE MAIN
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

CREATE FUNCTION
 DSN8.TABLE_NAME(
 VARCHAR(18) CCSID EBCDIC,
 VARCHAR(8) CCSID EBCDIC,
 VARCHAR(16) CCSID EBCDIC)
RETURNS
 VARCHAR(18) CCSID EBCDIC
SPECIFIC DSN8.DSN8DUTINVVV
LANGUAGE C
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME DSN8DUTI
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
COLLID DSN8DU!!
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE MAIN
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

CREATE FUNCTION
 DSN8.TABLE_SCHEMA(
 VARCHAR(18) CCSID EBCDIC)
RETURNS
 VARCHAR(8) CCSID EBCDIC
SPECIFIC DSN8.DSN8DUTISV
LANGUAGE C
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME DSN8DUTI
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
COLLID DSN8DU!!
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE MAIN
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

CREATE FUNCTION
 DSN8.TABLE_SCHEMA(
 VARCHAR(18) CCSID EBCDIC,
 VARCHAR(8) CCSID EBCDIC)
RETURNS
 VARCHAR(8) CCSID EBCDIC
SPECIFIC DSN8.DSN8DUTISVV
LANGUAGE C
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME DSN8DUTI
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
COLLID DSN8DU!!
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE MAIN
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

```

```

CREATE FUNCTION
 DSN8.TABLE_SCHEMA(
 VARCHAR(18) CCSID EBCDIC,
 VARCHAR(8) CCSID EBCDIC,
 VARCHAR(16) CCSID EBCDIC)
RETURNS
 VARCHAR(8) CCSID EBCDIC
SPECIFIC DSN8.DSN8DUTISVVV
LANGUAGE C
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME DSN8DUTI
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
COLLID DSN8DU!!
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE MAIN
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

CREATE FUNCTION
 DSN8.TABLE_LOCATION(
 VARCHAR(18) CCSID EBCDIC)
RETURNS
 VARCHAR(16) CCSID EBCDIC
SPECIFIC DSN8.DSN8DUTILV
LANGUAGE C
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME DSN8DUTI
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
COLLID DSN8DU!!
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE MAIN
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

CREATE FUNCTION
 DSN8.TABLE_LOCATION(
 VARCHAR(18) CCSID EBCDIC,
 VARCHAR(8) CCSID EBCDIC)
RETURNS
 VARCHAR(16) CCSID EBCDIC
SPECIFIC DSN8.DSN8DUTILVV
LANGUAGE C
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME DSN8DUTI
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
COLLID DSN8DU!!
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE MAIN
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

CREATE FUNCTION
 DSN8.TABLE_LOCATION(
 VARCHAR(18) CCSID EBCDIC,
 VARCHAR(8) CCSID EBCDIC,
 VARCHAR(16) CCSID EBCDIC)
RETURNS
 VARCHAR(16) CCSID EBCDIC

```

```

SPECIFIC DSN8.DSN8DUTILVVV
LANGUAGE C
DETERMINISTIC
READS SQL DATA
EXTERNAL NAME DSN8DUTI
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
NO SCRATCHPAD
NO FINAL CALL
ALLOW PARALLEL
COLLID DSN8DU!!!
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE MAIN
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

CREATE FUNCTION
DSN8.WEATHER(
 VARCHAR(44) CCSID EBCDIC)
RETURNS
TABLE(
 CITY VARCHAR(30) CCSID EBCDIC,
 TEMP_IN_F INTEGER,
 HUMIDITY INTEGER,
 WIND VARCHAR(5) CCSID EBCDIC,
 WIND_VELOCITY INTEGER,
 BAROMETER FLOAT,
 FORECAST VARCHAR(25) CCSID EBCDIC)
SPECIFIC DSN8.DSN8DUWFV
LANGUAGE C
DETERMINISTIC
NO SQL
EXTERNAL NAME DSN8DUWF
PARAMETER STYLE DB2SQL
NULL CALL
NO EXTERNAL ACTION
SCRATCHPAD
FINAL CALL
DISALLOW PARALLEL
NO COLLID
ASUTIME LIMIT 10
STAY RESIDENT NO
PROGRAM TYPE SUB
WLM ENVIRONMENT WLMENV
SECURITY DB2
NO DBINFO;

GRANT EXECUTE ON SPECIFIC FUNCTION DSN8.DSN8DUADV,
DSN8.DSN8DUCDVVV,
DSN8.DSN8DUCDDVV,
DSN8.DSN8DUATV,
DSN8.DSN8DUCTVVV,
DSN8.DSN8DUCTTVV,
DSN8.DSN8DUCYFV,
DSN8.DSN8DUCYFVV,
DSN8.DSN8EUDNV,
DSN8.DSN8EUDND,
DSN8.DSN8EUMNV,
DSN8.DSN8EUMND,
DSN8.DSN8DUTINV,
DSN8.DSN8DUTINVV,
DSN8.DSN8DUTINVVV,
DSN8.DSN8DUTISV,
DSN8.DSN8DUTISVV,
DSN8.DSN8DUTISVVV,
DSN8.DSN8DUTILV,
DSN8.DSN8DUTILVV,
DSN8.DSN8DUTILVVV,
DSN8.DSN8DUWFV

TO PUBLIC;

/*
/* STEP 3: PREPARE EXTERNAL FOR CURRENT DATE ALTDATE UDF
/*
//PH02US03 EXEC DSNHC,MEM=DSN8DUAD,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'

```

```

//PC.DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8DUAD),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8DUAD),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUAD),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
NAME DSN8DUAD(R)
/*
/* STEP 4: PREPARE EXTERNAL FOR GIVEN DATE ALTDATE UDF
/*
//PH02US04 EXEC DSNHC,MEM=DSN8DUCD,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8DUCD),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8DUCD),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUCD),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
NAME DSN8DUCD(R)
/*
/* STEP 5: PREPARE EXTERNAL FOR CURRENT TIME ALTTIME UDF
/*
//PH02US05 EXEC DSNHC,MEM=DSN8DUAT,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8DUAT),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8DUAT),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUAT),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
NAME DSN8DUAT(R)
/*
/* STEP 6: PREPARE EXTERNAL FOR GIVEN TIME ALTTIME UDF
/*
//PH02US06 EXEC DSNHC,MEM=DSN8DUCT,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8DUCT),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8DUCT),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUCT),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
NAME DSN8DUCT(R)
/*
/* STEP 7: PREPARE EXTERNAL FOR CURRENCY UDF
/*
//PH02US07 EXEC DSNHC,MEM=DSN8DUCY,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8DUCY),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8DUCY),
// DISP=SHR

```

```

//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUCY),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
NAME DSN8DUCY(R)
///*
//* STEP 8: PREPARE EXTERNAL FOR DAYNAME UDF
//*
//PH02US08 EXEC DSNHCPP,MEM=DSN8EUDN,COND=(4,LT),
// PARM.PC='HOST(CPP),CCSID(1047),MARGINS(1,80),STDSQL(NO)',
// SOURCE,XREF),
// PARM.CP='/'CXX SOURCE XREF OPTFILE(DD:CCOPTS)',
// 'LANGLVL(EXTENDED)'),
// PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8EUDN),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8EUDN),
// DISP=SHR
//CP.CCOPTS DD DSN=SYS1.PROCLIB(DSNHCPPS),DISP=SHR
//CP.USERLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8EUDN),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
NAME DSN8EUDN(R)
///*
//* STEP 9: PREPARE EXTERNAL FOR MONTHNAME UDF
//*
//PH02US09 EXEC DSNHCPP,MEM=DSN8EUMN,COND=(4,LT),
// PARM.PC='HOST(CPP),CCSID(1047),MARGINS(1,80),STDSQL(NO)',
// SOURCE,XREF),
// PARM.CP='/'CXX SOURCE XREF OPTFILE(DD:CCOPTS)',
// 'LANGLVL(EXTENDED)'),
// PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8EUMN),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8EUMN),
// DISP=SHR
//CP.CCOPTS DD DSN=SYS1.PROCLIB(DSNHCPPS),DISP=SHR
//CP.USERLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8EUMN),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
NAME DSN8EUMN(R)
///*
//* STEP 10: PREPARE EXTERNAL FOR TABLE_NAME, TABLE_SCHEMA,
//* AND TABLE_LOCATION UDF'S
//*
//PH02US10 EXEC DSNHC,MEM=DSN8DUTI,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE RENT XREF MARGINS(1,72)_OPTFILE(DD:CCOPTS) ',
// PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8DUTI),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8DUTI),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DUTI),
// DISP=SHR
//LKED.IGNORE DD *
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
NAME DSN8DUTI(R)
///*
//* STEP 11: BIND PACKAGE FOR TABLE_NAME, TABLE_SCHEMA, AND
//* TABLE_LOCATION UDF'S
//*
//PH02US11 EXEC PGM=IKJEFT01,COND=(4,LT)
//DBMLIB DD DSN=DSN!!0.DBMLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*

```

```

//SYSOUT DD SYSOUT=*
//REPORT DD SYSOUT=*
//SYSIN DD *
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE (DSN8DU!!) MEMBER(DSN8DUTI) APPLCOMPAT(V!!R1) +
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
END
///*
///* STEP 12: EXERCISE THE SAMPLE UDF'S
///*
//PH02US12 EXEC PGM=IKJEFT01,COND=(4,LT),DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP!!) -
 LIB('DSN!!0.RUNLIB LOAD') PARMS('/ALIGN(MID)')
END
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
// DD DSN=DSN!!0.SDSNSAMP(DSNTESU),
// DISP=SHR
///*
///* STEP 13: PREPARE EXTERNAL FOR WEATHER UDF TABLE FUNCTION
///*
//PH02US13 EXEC DSNHC,MEM=DSN8DUWF,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DUWF),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8DUWF),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB LOAD(DSN8DUWF),
// DISP=SHR
//LKED.IGNORE DD *
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
NAME DSN8DUWF(R)
///*
///* STEP 14: PREPARE CLIENT FOR WEATHER UDF TABLE FUNCTION
///*
//PH02US14 EXEC DSNHC,MEM=DSN8DUWC,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA(DSN8DUWC),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8DUWC),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB LOAD(DSN8DUWC),
// DISP=SHR
//LKED.IGNORE DD *
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNELI)
INCLUDE SYSLIB(DSNTIAR)
NAME DSN8DUWC(R)
///*
///* STEP 15: BIND PACKAGE & PLAN FOR WEATHER TBL FUNC CLIENT
///*
//PH02US15 EXEC PGM=IKJEFT01,COND=(4,LT)
//DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//REPORT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE (DSN8DU!!) MEMBER(DSN8DUWC) -
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN (DSN8UW!!) PKLIST(DSN8DU!!.*)
ACTION(REPLACE) RETAIN +

```

```

ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC) SQLRULES(DB2)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB LOAD')
END
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT EXECUTE,BIND ON PLAN DSN8UW!!
TO PUBLIC;
/***
/* STEP 16: INVOKE THE SAMPLE UDF TABLE CLIENT
/***
//PH02US16 EXEC PGM=IKJEFT01,COND=(4,LT),DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
RUN PROGRAM(DSN8DUWC) PLAN(DSN8UW!!) -
LIB('DSN!!0.RUNLIB LOAD') -
PARMS('DSN!!0.SDSNIVPD(DSN8LWC')
END
/***

```

## Referencia relacionada

“Ejemplos de aplicaciones en TSO” en la página 1095

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

DSNTEJ71

```

//***** *****
//** NAME = DSNTEJ71
//*
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 7
//** SAMPLE APPLICATIONS: POPULATE, CHECK LOB TABLE
//** C LANGUAGE
//*
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
//*
//** STATUS = VERSION 12
//*
//** FUNCTION = PREPARES AND RUNS THE FOLLOWING PROGRAMS IN SUPPORT
//** OF THE DB2 LOB SAMPLE C APPLICATION:
//** - DSN8DLPL: POPULATES THE PSEG AND BMP IMAGE COLUMNS
//** IN THE DSN8!!0.EMP_PHOTO_RESUME SAMPLE LOB
//** TABLE. THE INPUT DATA IS READ FROM A TSO
//** DATA SET. THIS PROGRAM DEMONSTRATES HOW
//** TO POPULATE LOB COLUMNS WITH MORE THAN 32
//** KB OF DATA.
//*
//** - DSN8DLTC: VALIDATES THE CONTENTS OF THE LOB COLUMNS
//** IN THE DSN8!!0.EMP_PHOTO_RESUME TABLE.
//** THIS IS DONE BY COMPARING THE DATA IN THE
//** TABLE TO THE SOURCE DATA SETS.
//*
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//*
//***** *****
//JOBLIB DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
// DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
// DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
// DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
//*
//** STEP 1: PREPARE LOADER FOR EMPLOYEE PHOTO IMAGES
//*
//PH071S01 EXEC DSNHC, MEM=DSN8DLPL,COND=(4,LT),
// PARM.PC=('HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',

// SOURCE,XREF),
// PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM.IKFD='MAP RENT REUS AMODE=31 RMODE=ANY'
//***** *****

```

```

//PC.DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8DLPL),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8DLPL),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DLPL),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNELI)
NAME DSN8DLPL(R)
/*
/*
 STEP 2: PREPARE SAMPLE LOB TABLE VALIDATOR
/*
//PH071S02 EXEC DSNHC,MEM=DSN8DLTC,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBMLIB DD DSN=DSN!!0.DBMLIB.DATA(DSN8DLTC),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8DLTC),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DLTC),
// DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNELI)
NAME DSN8DLTC(R)
/*
/*
 STEP 3: BIND PACKAGES AND PLANS FOR DSN8DLPL AND DSN8DLTC
/*
//PH071S03 EXEC PGM=IKJEFT01,COND=(4,LT)
//DBMLIB DD DSN=DSN!!0.DBMLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//REPORT DD SYSOUT=*
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT BIND, EXECUTE ON PLAN DSN8LC!!, DSN8LL!!
TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE (DSN8LL!!) MEMBER(DSN8DLPL) APPLCOMPAT(V!!R1) +
QUALIFIER(DSN8!!0) -
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8LL!!) PKLIST(DSN8LL!!.*)
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC) SQLRULES(DB2)

BIND PACKAGE (DSN8LC!!) MEMBER(DSN8DLTC) APPLCOMPAT(V!!R1) +
QUALIFIER(DSN8!!0) -
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8LC!!) PKLIST(DSN8LC!!.*)
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC) SQLRULES(DB2)

RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END
/*
/*
 STEP 4: LOAD SAMPLE EMPLOYEE PHOTO IMAGES
/*
//PH071S04 EXEC PGM=IKJEFT01,COND=(4,LT),DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
RUN PROGRAM(DSN8DLPL) PLAN(DSN8LL!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END
//PSEG1N00 DD DSN=DSN!!0.SDSNIVPD(DSN8P130),DISP=SHR
//BMPING0 DD DSN=DSN!!0.SDSNIVPD(DSN8B130),DISP=SHR
//PSEG1N01 DD DSN=DSN!!0.SDSNIVPD(DSN8P140),DISP=SHR

```

```

//BMPIN01 DD DSN=DSN!!0.SDSNIVPD(DSN8B140),DISP=SHR
//PSEGIN02 DD DSN=DSN!!0.SDSNIVPD(DSN8P150),DISP=SHR
//BMPIN02 DD DSN=DSN!!0.SDSNIVPD(DSN8B150),DISP=SHR
//PSEGIN03 DD DSN=DSN!!0.SDSNIVPD(DSN8P190),DISP=SHR
//BMPING03 DD DSN=DSN!!0.SDSNIVPD(DSN8B190),DISP=SHR
//*
//** STEP 5: VERIFY THE CONTENTS OF THE SAMPLE LOB TABLE
//*
//PH071S05 EXEC PGM=IKJEFT01,COND=(4,LT),DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DSN)
 RUN PROGRAM(DSN8DLTC) PLAN(DSN8LC!!)
 END
//PSEGIN00 DD DSN=DSN!!0.SDSNIVPD(DSN8P130),DISP=SHR
//BMPIN00 DD DSN=DSN!!0.SDSNIVPD(DSN8B130),DISP=SHR
//RESUME00 DD DSN=DSN!!0.SDSNIVPD(DSN8R130),DISP=SHR
//PSEGIN01 DD DSN=DSN!!0.SDSNIVPD(DSN8P140),DISP=SHR
//BMPING01 DD DSN=DSN!!0.SDSNIVPD(DSN8B140),DISP=SHR
//RESUME01 DD DSN=DSN!!0.SDSNIVPD(DSN8R140),DISP=SHR
//PSEGIN02 DD DSN=DSN!!0.SDSNIVPD(DSN8P150),DISP=SHR
//BMPING02 DD DSN=DSN!!0.SDSNIVPD(DSN8B150),DISP=SHR
//RESUME02 DD DSN=DSN!!0.SDSNIVPD(DSN8R150),DISP=SHR
//PSEGIN03 DD DSN=DSN!!0.SDSNIVPD(DSN8P190),DISP=SHR
//BMPING03 DD DSN=DSN!!0.SDSNIVPD(DSN8B190),DISP=SHR
//RESUME03 DD DSN=DSN!!0.SDSNIVPD(DSN8R190),DISP=SHR

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## DSNTEJ73

Db2 PREPARA Y EJECUTA LOS SIGUIENTES PROGRAMAS EN APOYO DE LA SOLICITUD DE MUESTRA DE LOB DE L

```

//*****
//** NAME = DSNTEJ73
//**
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 7
//** SAMPLE APPLICATIONS: VIEW, MANIPULATE CLOB DATA
//** C LANGUAGE
//**
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
//**
//** STATUS = VERSION 12
//**
//** FUNCTION = PREPARES AND RUNS THE FOLLOWING PROGRAMS IN SUPPORT
//** OF THE DB2 LOB SAMPLE C APPLICATION:
//** - DSN8SDM: CAF CONNECTION MANAGER (C LANGUAGE), USED
//** TO INVOKE THE DB2 SAMPLE APPLICATIONS MENU
//** UNDER ISPF AND TO MANAGE INVOKATION OF THE
//** DB2 SAMPLE APPLICATION PROGRAMS, INCLUDING
//** THE LOB SAMPLE RESUME AND PHOTO IMAGE
//** VIEWERS.
//**
//** - DSN8DLRV: EXTRACTS A SPECIFIED EMPLOYEE'S RESUME IN
//** CLOB FORMAT FROM DSN8!!0.EMP_PHOTO_RESUME.
//** DB2 LOB LOCATOR FUNCTIONS ARE USED TO PARSE
//** DATA FROM CLOB FORMAT INTO ISPF FIELDS AND
//** THE RESULT IS DISPLAYED TO THE USER.
//**
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//**
//*****JOBLIB DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
// DD DSN=DSN!!0.SDSNLOAD,DISP=SHR

```

```

// DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
// DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
///*
//** STEP 1: PREPARE SAMPLE CALL ATTACH CONTROLLER
//*/
//PH073S01 EXEC DSNHC,MEM=DSN8SDM,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM,LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8SDM),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSENSAMP(DSN8SDM),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8SDM),
// DISP=SHR
//LKED.SYSIN DD *
//INCLUDE SYSLIB(DSNALI)
NAME DSN8SDM(R)
//*
//** STEP 2: PREPARE EMPLOYEE RESUME VIEWER (ISPF)
//*/
//PH073S02 EXEC DSNHC,MEM=DSN8DLRV,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF),
// PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM,LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8DLRV),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSENSAMP(DSN8DLRV),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DLRV),
// DISP=SHR
//LKED.SYSIN DD *
//INCLUDE SYSLIB(DSNALI)
NAME DSN8DLRV(R)
//*
//** STEP 3: BIND PACKAGE AND PLAN FOR THE RESUME VIEWER
//*/
//PH073S03 EXEC PGM=IKJEFT01,COND=(4,LT)
//DBRLIB DD DSN=DSN!!0.DBRLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//REPORT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
 BIND PACKAGE (DSN8LR!!) APPLCOMPAT(V!!R1) +
 MEMBER(DSN8DLRV) -
 QUALIFIER(DSN8!!0) -
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8LR!!) -
 PKLIST(DSN8LR!!..*) -
 ACTION(REPLACE) RETAIN +
 ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC) SQLRULES(DB2)
RUN PROGRAM(DSNTIAD) -
 PLAN(DSNTIA!!) -
 LIB('DSN!!0.RUNLIB.LOAD')
END
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 GRANT EXECUTE,BIND ON PLAN DSN8LR!!
 TO PUBLIC;

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

### DSNTEJ75

PREPARA Y EJECUTA EL SIGUIENTE PROGRAMA EN APOYO DE LA SOLICITUD DE MUESTRA DE LOB DE LA OFICINA DE VENTAS ( Db2 ).

```

//***** *****
///* NAME = DSNTEJ75
///*
///* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
///* PHASE 7
///* SAMPLE APPLICATIONS: VIEW, MANIPULATE BLOB DATA
///* C LANGUAGE
///*
///* LICENSED MATERIALS - PROPERTY OF IBM
///* 5650-DB2
///* (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
///*
///* STATUS = VERSION 12
///*
///* FUNCTION = PREPARES AND RUNS THE FOLLOWING PROGRAM IN SUPPORT
///* OF THE DB2 LOB SAMPLE C APPLICATION:
///* - DSN8DLPV: EXTRACTS A SPECIFIED EMPLOYEE'S PSEG PHOTO
///* IMAGE IN BLOB FORMAT FROM THE SMAPLE TABLE
///* DSN8!!0.EMP_PHOTO_RESUME. THE DATA IS
///* HANDED OFF TO GDDM FOR CONVERSION FOR CON-
///* VERSION AND DISPLAY.
///*
///* CHANGE ACTIVITY =
///* 08/18/2014 Single-phase migration s21938_inst1 s21938
///*
//***** *****
//JOBLIB DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
// DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
// DD DSN=CEE.V!R!M!.SCEERUN,DISP=SHR
// DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
///*
///* STEP 1: PREPARE EMPLOYEE PHOTO VIEWER (GDDM)
///*
//PH075S01 EXEC DSNHC,MEM=DSN8DLPV,COND=(4,LT),
// PARM.PC='HOST(C),CCSID(1047),MARGINS(1,72),STDSQL(NO)',
// SOURCE,XREF',
// PARM.C='SOURCE RENT XREF MARGINS(1,72) OPTFILE(DD:CCOPTS)',
// PARM.LKED='MAP,RENT,REUS,AMODE=31,RMODE=ANY'
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8DLPV),
// DISP=SHR
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8DLPV),
// DISP=SHR
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8DLPV),
// DISP=SHR
//LKED.SYSIN DD *
// INCLUDE SYSLIB(ADMASRT)
// INCLUDE SYSLIB(DSNTIAR)
// INCLUDE SYSLIB(DSNALI)
// NAME DSN8DLPV(R)
///*
///* STEP 2: BIND PACKAGE AND PLAN FOR THE PHOTO VIEWER
///*
//PH075S02 EXEC PGM=IKJEFT01,COND=(4,LT)
//DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//REPORT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE (DSN8LP!!) APPLCOMPAT(V!!R1) +
 MEMBER(DSN8DLPV) -
 QUALIFIER(DSN8!!0) -
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8LP!!) -
 PKLIST(DSN8LP!!.*.) -
 ACTION(REPLACE) RETAIN +
 ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC) SQLRULES(DB2)
RUN PROGRAM(DSNTIAD) -
 PLAN(DSNTIA!!) -
 LIB('DSN!!0.RUNLIB.LOAD')
END
//SYSIN DD *
SET CURRENT SQLID = 'SYSADM';
GRANT EXECUTE ON PLAN DSN8LP!!
 TO PUBLIC;

```

### Referencia relacionada

[“Ejemplos de aplicaciones en TSO” en la página 1095](#)

Un conjunto de aplicaciones de muestra de Db2 , ejecutadas en el entorno TSO.

## Ejemplos de aplicaciones en IMS

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

Tabla 183. Ejemplos de solicitudes de Db2 para IMS

Aplicación	Nombre de programa	Nombre de miembro JCL	Descripción
Organización	DSN8ICO0	DSNTEJ4C	IMS Aplicación de organización COBOL
	DSN8IC1		
	DSN8IC2		
Organización	DSN8IP0	DSNTEJ4P	IMS Solicitud de organización PL/I
	DSN8IP1		
	DSN8IP2		
Proyecto	DSN8IP6	DSNTEJ4P	IMS Solicitud de proyecto PL/I
	DSN8IP7		
	DSN8IP8		
Teléfono	DSN8IP3	DSNTEJ4P	IMS Aplicación para teléfonos PL/I. Este programa enumera los números de teléfono de los empleados y los actualiza si se solicita.

### Referencia relacionada

[Conjuntos de datos que utiliza el precompilador](#)

Cuando invoca el precompilador, necesita proporcionar conjuntos de datos que contienen entrada para el precompilador, como las sentencias de programación de host o sentencias SQL. También necesita proporcionar conjuntos de datos donde el precompilador pueda almacenar la salida, como el código de origen modificado o los mensajes de diagnóstico.

## DSN8ICO0

ESTE MÓDULO RECIBE UN MENSAJE DE ENTRADA Y LO DEFORMA, LLAMA A DSN8IC1, FORMATEA EL MENSAJE DE SALIDA Y LO ENVÍA.

```
IDENTIFICATION DIVISION. 00010000
*----- 00012000
PROGRAM-ID. DSN8ICO0. 00014000
 00016000
***** DSN8ICO0 - IMS SUBSYSTEM INTERFACE MODULE - COBOL *****
* 00018000
* * 00020000
* MODULE NAME = DSN8ICO0 * 00030000
* * 00040000
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION * 00050000
* SUBSYSTEM INTERFACE MODULE * 00060000
* IMS * 00070000
* COBOL * 00080000
* ORGANIZATION APPLICATION * 00090000
*
*LICENSED MATERIALS - PROPERTY OF IBM * 00100000
*5615-DB2 * 00110000
*(C) COPYRIGHT 1996, 2013 IBM CORP. ALL RIGHTS RESERVED. * 00125000
* * 00126000
*STATUS = VERSION 11 * 00128001
* * 00130000
```

```
* FUNCTION = THIS MODULE RECEIVES AN INPUT MESSAGE AND
* DEFORMATS IT, CALLS DSN8IC1,
* FORMATS OUTPUT MESSAGE AND SENDS IT.
*
* NOTES = NONE
*
* MODULE TYPE =
* PROCESSOR = DB2 PREPROCESSOR, COBOL COMPILER
* MODULE SIZE = SEE LINKEDIT
* ATTRIBUTES = REUSABLE
*
* ENTRY POINT = DSN8ICO
* PURPOSE = SEE FUNCTION
* LINKAGE = FROM IMS
*
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
*
* SYMBOLIC LABEL/NAME = DSN8ICGI
* DESCRIPTION = IMS/VS MFS GENERAL MENU
*
* SYMBOLIC LABEL/NAME = DSN8ICDI
* DESCRIPTION = IMS/VS MFS DETAIL MENU
*
* OUTPUT = PARAMETERS EXPLICITLY RETURNED:
*
* SYMBOLIC LABEL/NAME = DSN8ICGO
* DESCRIPTION = IMS/VS MFS GENERAL MENU
*
* SYMBOLIC LABEL/NAME = DSN8ICDO
* DESCRIPTION = IMS/VS MFS DETAIL MENU
*
* EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION
*
* EXIT-ERROR =
*
* RETURN CODE = NONE
*
* ABEND CODES = NONE
*
* ERROR-MESSAGES =
* DSN8064E - INVALID DL/I STC-CODE ON GU MSG
* DSN8065E - INVALID DL/I STC-CODE ON ISRT MSG
*
* EXTERNAL REFERENCES =
* ROUTINES/SERVICES = MODULE DSN8IC1
* MODULE CBLTDLI
* MODULE DSN8MCG
*
* DATA-AREAS =
* DSN8MCCA - PARAMETER TO BE PASSED TO DSN8IC1
* CONTAINS TERMINAL INPUT AND
* OUTPUT AREAS.
*
* CONTROL-BLOCKS =
* IN-MESSAGE - MFS INPUT
* OUT-MESSAGE - MFS OUTPUT
*
* TABLES = NONE
*
* CHANGE-ACTIVITY =
* 05/18/2012: SWITCH ARITHMETICS FROM COMP TO COMP-5 PM66408
*
* *PSEUDOCODE*
*
* PROCEDURE
* DECLARATIONS.
* ALLOCATE COBOL WORK AREA FOR COMMAREA.
* INITIALIZATION.
* PUT MODNAME 'DSN8ICGO' IN MODNAME FIELD.
* PUT MODULE NAME 'DSN8ICO' IN AREA USED BY
* ERROR-HANDLER.
*
* STEP1.
* CALL DLI GU INPUT MESSAGE.
* IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE AND
* STOP PROGRAM.
*
* IF SCREEN CLEARED/UNFORMATTED , MOVE '00' TO PFKIN.
* MOVE INPUT MESSAGE FIELDS TO CORRESPONDING
* INAREA FIELDS IN COMPARM.
* CALL DSN8IC1 (COMMAREA)
* MOVE OUTAREA FIELDS IN PCONVSTA TO CORRESPONDING
```

```

* OUTPUT MESSAGE FIELDS. * 00970000
* IF LASTSCR 'DSN8001' MOVE 'DSN8ICGO' TO MODNAME FIELD * 00980000
* ELSE MOVE 'DSN8ICDO' TO MODNAME FIELD. * 00990000
* * 01000000
* CALL DLI ISRT OUTPUT MESSAGE. * 01010000
* IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE AND * 01020000
* STOP PROGRAM. * 01030000
* * 01040000
* END. * 01050000
* * 01060000
*****ENVIRONMENT DIVISION. 01070000
*----- 01080000
*----- 01130000
*----- 01140000
*----- 01150000
*----- 01160000
*----- 01170000
*-----WORKING-STORAGE SECTION. 01180000
*****DECLARATION FOR PASSING INPUT/OUTPUT DATA BETWEEN THE * 01200000
*-----SUBSYSTEM DEPENDENT MODULE IMS/DLI AND SQL1 AND SQL2 * 01210000
*****01 COMMAREA. 01220000
*-----EXEC SQL INCLUDE DSN8MCCA END-EXEC. 01230000
*****01 IN-MESSAGE. 01240000
*-----02 LL PIC S9(3) COMP-5. 01250000
*-----02 Z1 PIC X. 01260000
*-----02 Z2 PIC X. 01270000
*-----02 TC-CODE PIC X(7). 01280000
*-----02 IN-PUT. 01290000
*-----03 MAJSYS PIC X. 01300000
*-----03 ACTION PIC X. 01310000
*-----03 OBJFLD PIC X(2). 01320000
*-----03 SRCH PIC X(2). 01330000
*-----03 PFKIN PIC X(2). 01340000
*-----03 DATAIN PIC X(60). 01350000
*-----03 TRANDATA PIC X(40) OCCURS 15. 01360000
*-----02 IN-PUT0 REDEFINES IN-PUT PIC X(668). 01370000
*****01 OUT-MESSAGE. 01380000
*-----02 LL PIC S9(3) COMP-5. 01390000
*-----02 ZZ PIC S9(3) COMP-5 VALUE +0. 01400000
*-----02 OUTPUTAREA. 01410000
*-----03 MAJSYS PIC X. 01420000
*-----03 ACTION PIC X. 01430000
*-----03 OBJFLD PIC X(2). 01440000
*-----03 SRCH PIC X(2). 01450000
*-----03 DATAOUT PIC X(60). 01460000
*-----03 HTITLE PIC X(50). 01470000
*-----03 DESC2 PIC X(50). 01480000
*-----03 DESC3 PIC X(50). 01490000
*-----03 DESC4 PIC X(50). 01500000
*-----03 MSG. 01510000
*-----05 STC PIC X(4). 01520000
*-----05 MSGTEXT PIC X(75). 01530000
*-----03 PFKTEXT PIC X(79). 01540000
*-----03 OUTPUT0 OCCURS 15. 01550000
*-----05 LINE0 PIC X(79). 01560000
*-----02 OUTPUTARE0 REDEFINES OUTPUTAREA PIC X(1609). 01570000
*****01 MSGCODE. 01580000
*-----PIC X(04). 01590000
*-----01 OUTMSG PIC X(69). 01600000
*-----01 OUTMSG PIC X(69). 01610000
*-----01 OUTMSG PIC X(69). 01620000
*-----05 STC PIC X(4). 01630000
*-----05 MSGTEXT PIC X(75). 01640000
*-----03 PFKTEXT PIC X(79). 01650000
*-----03 OUTPUT0 OCCURS 15. 01660000
*-----05 LINE0 PIC X(79). 01670000
*-----02 OUTPUTARE0 REDEFINES OUTPUTAREA PIC X(1609). 01680000
*****01 FIELDS SENT TO MESSAGE ROUTINE. 01690000
*-----01 MSGCODE PIC X(04). 01700000
*****01 MSGCODE. 01710000
*****01 MSGCODE. 01720000
*-----01 MSGCODE PIC X(04). 01730000
*-----01 MSGCODE PIC X(04). 01740000
*-----01 OUTMSG PIC X(69). 01750000
*****01 DECLARATION FOR PGM-LOGIC. 01760000
*****01 DECLARATION FOR PGM-LOGIC. 01770000
*****01 DECLARATION FOR PGM-LOGIC. 01780000
*****01 DECLARATION FOR PGM-LOGIC. 01790000
*-----77 GU-FKT PIC X(4) VALUE 'GU '. 01800000
*-----77 ISRT-FKT PIC X(4) VALUE 'ISRT'. 01810000
*-----77 CHNG-FKT PIC X(4) VALUE 'CHNG'. 01820000

```

```

77 ROLL-FKT PIC X(4) VALUE 'ROLL'. 01830000
* 01840000
77 MODNAME PIC X(8). 01850000
*****LINKAGE SECTION***** 01860000
* * 01870000
*****LINKAGE SECTION. 01880000
*****DECLARATION FOR IO / ALTPCB 01890000
* * 01900000
*****DEclaration FOR IO / ALTPCB 01910000
* * 01920000
* 01930000
01 IOPCB. 01940000
02 IOLTERM PIC X(8). 01950000
02 FILLER PIC X(2). 01960000
02 STC-CODE PIC X(2). 01970000
02 CDATE PIC X(4). 01980000
02 CTIME PIC X(4). 01990000
02 SEQNUM PIC X(4). 02000000
02 MOD-NAME PIC X(8). 02010000
02 USERID PIC X(8). 02020000
 02030000
* 02040000
01 ALTPCB. 02050000
02 ALTLTERM PIC X(8). 02060000
02 FILLER PIC X(2). 02070000
02 STC-CODE PIC X(2). 02080000
 02090000
PROCEDURE DIVISION. 02100000
*----- 02110000
* 02120000
ENTRY 'DLITCBL' USING IOPCB ALTPCB. 02130000
*****ALLOCATE COBOL WORK AREA /INITIALIZATIONS * 02140000
***** 02150000
* 02160000
CSTART. 02170000
MOVE SPACES TO COMMAREA. 02180000
MOVE SPACES TO IN-MESSAGE. 02190000
MOVE 'DSN8ICGO' TO MODNAME. 02200000
MOVE 'DSN8IC0' TO MAJOR IN DSN8-MODULE-NAME. 02210000
MOVE '0' TO MAJSYS IN OUTAREA. 02220000
MOVE '0' TO EXITCODE. 02230000
MOVE +1613 TO LL IN OUT-MESSAGE. 02240000
 02250000
***** 02260000
* CALL DL1 GU INPUT MESSAGE * 02270000
* PRINT ERROR MESSAGE IF STATUS CODE NOT OK * 02280000
***** 02290000
 02300000
* **CALL DL1 GU 02310000
CALL 'CBLTDLI' 02320000
USING GU-FKT IOPCB IN-MESSAGE. 02330000
 02340000
* **ERROR? 02350000
IF STC-CODE IN IOPCB NOT = '' 02360000
 THEN MOVE '064E' TO MSGCODE 02370000
 CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG 02380000
 MOVE OUTMSG TO MSGTEXT IN OUTPUTAREA 02390000
 MOVE STC-CODE IN IOPCB TO STC IN OUTPUTAREA 02400000
 GO TO CSEND. 02410000
 02420000
***** 02430000
* CLEARED AND UNFORMATTED SCREEN? * 02440000
***** 02450000
 02460000
IF Z2 = LOW-VALUE 02470000
 THEN MOVE '00' TO PFKIN IN INAREA. 02480000
 MOVE IOLTERM IN IOPCB TO TRMID IN CONVID. 02490000
 MOVE USERID IN IOPCB TO USERID IN CONVID. 02500000
 02510000
* **MOVE INPUT MESSAGE 02520000
* **FIELDS TO INAREA FIELDS 02530000
 MOVE IN-PUT0 TO INAREA0. 02540000
 MOVE '0' TO MAJSYS IN INAREA. 02550000
* 02560000
* CALL 'DSN8IC1' USING COMMAREA. 02570000
 02580000
* **MOVE OUTAREA FIELDS TO 02590000
* **OUTPUT MESSAGE FIELDS 02600000
* MOVE OUTAREA0 TO OUTPUTAREA0. 02610000
* 02620000
* IF LASTSCR = 'DSN8002' 02630000
* THEN MOVE 'DSN8ICD0' TO MODNAME 02640000

```

## Referencia relacionada

“Ejemplos de aplicaciones en IMS” en la página 1415

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

DSN8IC1

ESTE MÓDULO RECUPERA LA FILA QUE CONTIENE INFORMACIÓN SOBRE LA CONVERSACIÓN ACTUAL, VALIDA LOS CRITERIOS DE SELECCIÓN Y ENVÍA MENSAJES PARA COMPLETAR LOS CRITERIOS DE ACCIÓN, OBJETO Y BÚSQUEDA.

```
IDENTIFICATION DIVISION.
*-----
PROGRAM-ID. DSN8IC1.

***** DSN8IC1 - SQL 1 MAINLINE FOR IMS - COBOL *****
*
* MODULE NAME = DSN8IC1
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
* SQL 1 MAINLINE
* IMS
* COBOL
*
*COPYRIGHT = 5615-DB2 (C) COPYRIGHT IBM CORP 1982, 2013
*REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
*
*STATUS = VERSION 11
*
* FUNCTION = THIS MODULE RETRIEVES THE ROW CONTAINING
* INFORMATION ON THE CURRENT CONVERSATION,
* VALIDATES SELECTION CRITERIA, AND ISSUES
* MESSAGES TO COMPLETE THE ACTION, OBJECT,
*
```

```

* AND SEARCH CRITERIA.
*
* NOTES = NONE
*
* MODULE TYPE =
* PROCESSOR = DB2 PRECOMPILER, COBOL COMPILER
* MODULE SIZE = SEE LINKEDIT
* ATTRIBUTES = REUSABLE
*
* ENTRY POINT = DSN8IC1
* PURPOSE = SEE FUNCTION
* LINKAGE = CALLED BY DSN8IC0
*
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
*
* SYMBOLIC LABEL/NAME = COMMPTR
* DESCRIPTION = POINTER TO COMMAREA
*
* SYMBOLIC LABEL/NAME = INAREA
* DESCRIPTION = USER INPUT
*
* SYMBOLIC LABEL/NAME = PFKIN
* DESCRIPTION = 00/01/02/03/07/08/10/11
*
* OUTPUT = PARAMETERS EXPLICITLY RETURNED:
*
* SYMBOLIC LABEL/NAME = OUTAREA
* DESCRIPTION = GENERAL MENU OR SECONDARY
* SELECTION MENU
*
* SYMBOLIC LABEL/NAME = LASTSCR
* DESCRIPTION = DSN8001/DSN8002
*
* EXIT-NORMAL = DSN8IC0
*
* EXIT-ERROR = DSN8IC0
*
* RETURN CODE = NONE
*
* ABEND CODES = NONE
*
* ERROR-MESSAGES = NONE
*
* EXTERNAL REFERENCES =
* ROUTINES/SERVICES =
* DSN8IC2
* DSN8MCG
* DSNTIAR
*
* DATA-AREAS =
* DSN8MCCA - COBOL STRUCTURE FOR COMMAREA
* DSN8MCC2 - COMMAREA PART 2
* DSN8MCCS - VCONA TABLE DCL & PCONA DCLGEN
* DSN8MC0V - VOPTVAL TABLE DCL & POPTVAL DCLGEN
* DSN8MCVO - VALIDATION CURSORS
* DSN8MCXX - SQL ERROR HANDLING MODULE
* DSN8MC1 - SQL1 COMMON MODULE FOR IMS & CICS
* DSN8MC3 - DSN8MC5 - VALIDATION MODULES CALLED BY DSN8MC1
*
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA
*
* TABLES = NONE
*
* CHANGE-ACTIVITY = NONE
*
*
* *PSEUDOCODE*
* PROCEDURE
* INCLUDE DECLARATIONS.
* INCLUDE DSN8MC1.
*
* CC1EXIT: (REFERENCED BY DSN8MC1)
* RETURN.
*
* CC1CALL: (REFERENCED BY DSN8MC1)
* CALL 'DSN8IC2' USING COMMAREA.
* GO TO MC1SAVE. (LABEL IN DSN8MC1)
*
* INCLUDE VALIDATION MODULES.
*
* END.

```

```

* ENVIRONMENT DIVISION.

 DATA DIVISION.

 WORKING-STORAGE SECTION.
***** * DECLARE FIELD SENT TO MESSAGE ROUTINE *
***** * DECLARE CONVERSATION STATUS *
***** * DECLARE MESSAGE TEXT *
***** * DECLARE OPTION VALIDATION *
***** * DECLARE COMMON AREA AND COMMON AREA PART 2 *
***** *
01 MSGCODE PIC X(04).

01 OUTMSG PIC X(69).

 EXEC SQL INCLUDE DSN8MCCTS END-EXEC.
 EXEC SQL INCLUDE DSN8MCOV END-EXEC.
 EXEC SQL INCLUDE SQLCA END-EXEC.
 EXEC SQL INCLUDE DSN8MCC2 END-EXEC.
*
LINKAGE SECTION.
01 COMMAREA.
 EXEC SQL INCLUDE DSN8MCCA END-EXEC.
*
PROCEDURE DIVISION USING COMMAREA.

***** **SQL ERROR HANDLING
***** EXEC SQL WHENEVER SQLERROR GO TO DB-ERROR END-EXEC
***** EXEC SQL WHENEVER SQLWARNING GO TO DB-ERROR END-EXEC.

*
MOVE 'DSN8IC1' TO MAJOR IN DSN8-MODULE-NAME.

***** * FIND VALID OPTIONS FOR ACTION, OBJECT, SEARCH CRITERION*
***** * RETRIEVE CONVERSATION, VALIDATE, CALL SQL2 *
***** *
 EXEC SQL INCLUDE DSN8MCVO END-EXEC. **INCLUDE SQL1 MAIN
 EXEC SQL INCLUDE DSN8MC1 END-EXEC.
*
* **RETURN
 CC1-EXIT.
 GOBACK.
***** * VALIDATE ACTION, OBJECT, SEARCH CRITERIA *
***** * HANDLE ERRORS *
***** *

CC1-CALL.
 CALL 'DSN8IC2' USING COMMAREA.
 GO TO MC1-SAVE.

 EXEC SQL INCLUDE DSN8MC3 END-EXEC.
 EXEC SQL INCLUDE DSN8MC4 END-EXEC.
 EXEC SQL INCLUDE DSN8MC5 END-EXEC.

 EXEC SQL INCLUDE DSN8MCXX END-EXEC.
 GOBACK.

```

## Referencia relacionada

[“Ejemplos de aplicaciones en IMS” en la página 1415](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

## DSN8IC2

ROUTER PARA SELECCIÓN SECUNDARIA Y/O LLAMADAS DE PROCESAMIENTO DE DETALLES SELECCIÓN SECUNDARIA MÓDULOS DSN8MCA DSN8MCM LLAMADAS MÓDULOS DETALLES DSN8MCD DSN8MCE DSN8MCF DSN8MCT DSN8MCV DSN8MCW DSN8MCX DSN8MCZ LLAMADAS POR DSN8IC1 ( SQL1 ).

```
IDENTIFICATION DIVISION. 00010000
*----- 00020000
PROGRAM-ID. DSN8IC2. 00030000
 00040000
***** DSN8IC2 - SQL 2 COMMON MODULE FOR IMS - COBOL ***** 00050000
*
* MODULE NAME = DSN8IC2 00060000
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION 00070000
* SQL 2 COMMON MODULE 00080000
* IMS 00090000
* COBOL 00100000
*
* LICENSED MATERIALS - PROPERTY OF IBM 00110000
* 5615-DB2 00120000
* (C) COPYRIGHT 1995, 2013 IBM CORP. ALL RIGHTS RESERVED 00130000
*
* STATUS = VERSION 11 00132000
*
*
*
* FUNCTION = ROUTER FOR SECONDARY SELECTION AND/OR 00134000
* DETAIL PROCESSING 00137000
* CALLS SECONDARY SELECTION MODULES 00140000
* DSN8MCA DSN8MCM 00150000
* CALLS DETAIL MODULES 00160000
* DSN8MCD DSN8MCE DSN8MCF 00170000
* DSN8MCT DSN8MCV DSN8MCW DSN8MCX DSN8MCZ 00180000
* CALLED BY DSN8IC1 (SQL1) 00190000
*
* NOTES = NONE 00200000
*
*
* MODULE TYPE = 00203000
* PROCESSOR = DB2 PRECOMPILER, COBOL COMPILER 00206000
* MODULE SIZE = SEE LINKEDIT 00210000
* ATTRIBUTES = REUSABLE 00220000
*
* ENTRY POINT = DSN8IC2 00230000
* PURPOSE = SEE FUNCTION 00240000
* LINKAGE = NONE 00250000
* INPUT = POINTER TO COMMAREA (COMMUNICATION AREA) 00260000
*
* SYMBOLIC LABEL/NAME = COMMAREA 00270000
* DESCRIPTION = COMMUNICATION AREA PASSED BETWEEN 00280000
* MODULES 00290000
*
* OUTPUT = POINTER TO COMMAREA (COMMUNICATION AREA) 00300000
*
* SYMBOLIC LABEL/NAME = COMMAREA 00310000
* DESCRIPTION = COMMUNICATION AREA PASSED BETWEEN 00320000
* MODULES 00330000
*
* EXIT-NORMAL = 00340000
*
* EXIT-ERROR = IF SQLERROR OR SQLWARNING, SQL WHENEVER 00350000
* CONDITION SPECIFIED IN DSN8IC2 WILL BE RAISED 00360000
* AND PROGRAM WILL GO TO THE LABEL DB-ERROR. 00370000
*
* RETURN CODE = NONE 00380000
*
* ABEND CODES = NONE 00390000
*
* ERROR-MESSAGES = 00400000
* DSN8062E-AN OBJECT WAS NOT SELECTED 00410000
* DSN8066E-UNSUPPORTED PFK OR LOGIC ERROR 00420000
* DSN8072E-INVALID SELECTION ON SECONDARY SCREEN 00430000
*
* EXTERNAL REFERENCES = 00440000
* ROUTINES/SERVICES = 10 MODULES LISTED ABOVE 00450000
* DSN8MCG - ERROR MESSAGE ROUTINE 00460000
*
* 00470000
* 00480000
*
* 00490000
* 00500000
* 00510000
* 00520000
*
* 00530000
* 00540000
*
* 00550000
* 00560000
* 00570000
* 00580000
*
* 00590000
*
* 00600000
* 00610000
* 00620000
* 00630000
*
* 00640000
* 00650000
* 00660000
* 00670000
* 00680000
*
* 00690000
*
* 00700000
* 00710000
* 00720000
```

```

* DATA-AREAS = 00730000
* DSN8MCA - SECONDARY SELECTION FOR 00740000
* DSN8MCD - DEPARTMENT STRUCTURE DETAIL 00750000
* DSN8MCE - DEPARTMENT DETAIL 00760000
* DSN8MCF - EMPLOYEE DETAIL 00770000
* DSN8MCF - ORGANIZATION 00780000
* DSN8MCAD - DECLARE ADMINISTRATION DETAIL 00790000
* DSN8MCAE - CURSOR EMPLOYEE LIST 00800000
* DSN8MCAL - CURSOR ADMINISTRATION LIST 00810000
* DSN8MCA2 - DECLARE ADMINISTRATION DETAIL 00820000
* DSN8MCC2 - SQL COMMON AREA PART 2 00830000
* DSN8MCDA - CURSOR ADMINISTRATION DETAIL 00840000
* DSN8MCDH - CURSOR FOR DISPLAY TEXT FROM 00850000
* TDSPTXT TABLE 00860000
* DSN8MCDP - DECLARE DEPARTMENT 00870000
* DSN8MCEM - DECLARE EMPLOYEE 00880000
* DSN8MCED - DECLARE EMPLOYEE-DEPARTMENT 00890000
* DSN8MCDM - DECLARE DEPARTMENT MANAGER 00900000
* DSN8MCAD - DECLARE ADMINISTRATION DETAIL 00910000
* DSN8MCA2 - DECLARE ADMINISTRATION DETAIL 00920000
* DSN8MCOV - DECLARE OPTION VALIDATION 00930000
* DSN8MCDT - DECLARE DISPLAY TEXT 00940000
* DSN8MCCA - SQL COMMON AREA 00950000
* DSN8MCXX - ERROR HANDLER 00960000
* 00970000
* 00980000
* CONTROL-BLOCKS = 00990000
* SQLCA - SQL COMMUNICATION AREA 01000000
* 01010000
* TABLES = NONE 01020000
* 01030000
* CHANGE-ACTIVITY = 01040000
* - ADD NEW VARIABLES FOR REFERENTIAL INTEGRITY V2R1 01050000
* 01060000
* *PSEUDOCODE*
* THIS MODULE DETERMINES WHICH SECONDARY SELECTION AND/OR 01070000
* DETAIL MODULE(S) ARE TO BE CALLED FOR THE IMS/COBOL ENVIRONMENT 01080000
* 01090000
* 01100000
* WHAT HAS HAPPENED SO FAR?..... THE SUBSYSTEM 01110000
* DEPENDENT MODULE (IMS,CICS) (SQL 0) HAS READ THE 01120000
* INPUT SCREEN, FORMATTED THE INPUT, AND PASSED CONTROL 01130000
* TO SQL 1. SQL 1 PERFORMS VALIDATION ON THE SYSTEM DEPENDENT 01140000
* FIELDS (MAJOR SYSTEM, ACTION, OBJECT, SEARCH CRITERIA). IF 01150000
* ALL SYSTEM FIELDS ARE VALID, SQL 1 PASSED CONTROL TO THIS 01160000
* MODULE. PASSED PARAMETERS CONSIST ONLY OF A POINTER WHICH 01170000
* POINTS TO A COMMUNICATION CONTROL AREA USED TO COMMUNICATE 01180000
* BETWEEN SQL 0 , SQL 1, SQL 2, AND THE SECONDARY SELECTION 01190000
* AND DETAIL MODULES. 01200000
* 01210000
* 01220000
* WHAT IS INCLUDED IN THIS MODULE?..... 01230000
* ALL SECONDARY SELECTION AND DETAIL MODULES ARE 'INCLUDED'. 01240000
* ALL VARIABLES KNOWN IN THIS PROCEDURE ARE KNOWN IN THE 01250000
* SUB PROCEDURES. ALL SQL CURSOR DEFINITIONS AND 01260000
* SQL 'INCLUDES' ARE DONE IN THIS PROCEDURE. ALL CURSOR HOST 01270000
* VARIABLES ARE DECLARED IN THIS PROCEDURE BECAUSE OF THE 01280000
* RESTRICTION THAT CURSOR HOST VARIABLES MUST BE DECLARED BEFORE 01290000
* THE CURSOR DEFINITION. 01300000
* 01310000
* PROCEDURE 01320000
* IF ANSWER TO DETAIL SCREEN & DETAIL PROCESSOR 01330000
* IS NOT WILLING TO ACCEPT AN ANSWER THEN 01340000
* NEW REQUEST* 01350000
* 01360000
* ELSE 01370000
* IF ANSWER TO A SECONDARY SELECTION THEN 01380000
* DETERMINE IF NEW REQUEST. 01390000
* 01400000
* CASE (NEW REQUEST) 01410000
* 01420000
* SUBCASE ('ADD') 01430000
* DETAIL PROCESSOR 01440000
* RETURN TO SQL 1 01450000
* ENDSUB 01460000
* 01470000
* SUBCASE ('DISPLAY', 'ERASE', 'UPDATE') 01480000
* CALL SECONDARY SELECTION 01490000
* IF # OF POSSIBLE CHOICES IS ^= 1 THEN 01500000
* RETURN TO SQL 1 01510000
* ELSE 01520000
* CALL THE DETAIL PROCESSOR 01530000
* RETURN TO SQL 1. 01540000

```

```

* ENDSUB 01550000
* 01560000
* ENDCASE 01570000
* 01580000
* IF ANSWER TO SECONDARY SELECTION AND A SELECTION HAS 01590000
* ACTUALLY BEEN MADE THEN 01600000
* VALID SELECTION #? 01610000
* IF IT IS VALID THEN 01620000
* CALL DETAIL PROCESSOR 01630000
* RETURN TO SQL 1 01640000
* ELSE 01650000
* PRINT ERROR MSG 01660000
* RETURN TO SQL 1. 01670000
* 01680000
* IF ANSWER TO SECONDARY SELECTION THEN 01690000
* CALL SECONDARY SELECTION 01700000
* RETURN TO SQL 1. 01710000
* 01720000
* IF ANSWER TO DETAIL THEN 01730000
* CALL DETAIL PROCESSOR 01740000
* RETURN TO SQL 1. 01750000
* 01760000
* END. 01770000
* 01780000
* *EXAMPLE- A ROW IS SUCCESSFULLY ADDED, THE OPERATOR RECEIVES 01790000
* THE SUCCESSFULLY ADDED MESSAGE AND JUST HITS ENTER. 01800000
*----- 01810000
*----- 01820000
*----- 01830000
ENVIRONMENT DIVISION. 01840000
*----- 01850000
*----- 01860000
DATA DIVISION. 01870000
*----- 01880000
WORKING-STORAGE SECTION. 01890000
*----- 01900000
***** 01910000
* FIELD SENT TO MESSAGE ROUTINE 01920000
***** 01930000
 01 MSGCODE PIC X(04). 01940000
 01 OUTMSG PIC X(69). 01950000
 01960000
***** 01970000
* NULL INDICATOR * 01980000
***** 01990000
 01 NULLIND1 PIC S9(4) COMP-4. 02000000
 01 NULLIND2 PIC S9(4) COMP-4. 02010000
 01 NULLIND3 PIC S9(4) COMP-4. 02020000
 01 NULLIND4 PIC S9(4) COMP-4. 02030000
 01 NULLIND5 PIC S9(4) COMP-4. 02040000
 01 NULLARRY. 02050000
 03 NULLARRY1 PIC S9(4) USAGE COMP OCCURS 13 TIMES. 02060000
 02070000
 EXEC SQL INCLUDE SQLCA END-EXEC. 02080000
 02090000
 EXEC SQL INCLUDE DSN8MCC2 END-EXEC. 02100000
 EXEC SQL INCLUDE DSN8MCDP END-EXEC. 02110000
 EXEC SQL INCLUDE DSN8MCEM END-EXEC. 02120000
 EXEC SQL INCLUDE DSN8MCDM END-EXEC. 02130000
 EXEC SQL INCLUDE DSN8MCAD END-EXEC. 02140000
 EXEC SQL INCLUDE DSN8MCA2 END-EXEC. 02150000
 EXEC SQL INCLUDE DSN8MCOV END-EXEC. 02160000
 EXEC SQL INCLUDE DSN8MCDT END-EXEC. 02170000
 EXEC SQL INCLUDE DSN8MCED END-EXEC. 02180000
 02190000
 01 CONSTRAINTS. 02200000
 03 PARM-LENGTH PIC S9(4) COMP-4. 02210000
 03 REF-CONSTRAINT PIC X(08). 02220000
 03 FILLER PIC X(62). 02230000
 01 MGRNO-CONSTRAINT PIC X(08) VALUE 'RDE' 02240000
 02250000
LINKAGE SECTION. 02260000
 01 COMMAREA. 02270000
 EXEC SQL INCLUDE DSN8MCCA END-EXEC. 02280000
 02290000
 PROCEDURE DIVISION USING COMMAREA. 02300000
*----- 02310000
***** 02320000
* SQL ERROR CODE HANDLING 02330000
***** 02340000
 EXEC SQL WHENEVER SQLERROR GO TO DB-ERROR END-EXEC 02350000
 EXEC SQL WHENEVER SQLWARNING GO TO DB-ERROR END-EXEC. 02360000

```

```

EXEC SQL INCLUDE DSN8MCAE END-EXEC. 02370000
EXEC SQL INCLUDE DSN8MCAL END-EXEC. 02380000
EXEC SQL INCLUDE DSN8MCDH END-EXEC. 02390000
EXEC SQL INCLUDE DSN8MCDA END-EXEC. 02400000
EXEC SQL INCLUDE DSN8MCDA END-EXEC. 02410000
EXEC SQL INCLUDE DSN8MCDA END-EXEC. 02420000

* INITIALIZATIONS 02430000

MOVE 'DSN8IC2' TO MAJOR. 02440000
MOVE SPACES TO MINOR. 02450000
 02460000
IF NEWREQ OF COMPARM = 'Y' THEN GO TO IC2008. 02470000
 02480000
 02490000
 02500000
 02510000
 02520000

* DETERMINES WHETHER NEW REQUEST OR NOT 02530000

IC2005. 02540000
IF PREV OF PCONVSTA = ' ' THEN 02550000
MOVE 'Y' TO NEWREQ OF COMPARM. 02560000
 02570000
 02580000
 02590000
IF NEWREQ_OF COMPARM = 'N' AND PREV OF PCONVSTA = 'S' 02600000
AND DATA01 NOT = ' ' 02610000
AND DATAIN NOT = 'NEXT' 02620000
THEN MOVE 'Y' TO NEWREQ_OF COMPARM. 02630000
 02640000
IF NEWREQ_OF COMPARM NOT = 'Y' THEN GO TO IC2010. 02650000

* IF NEW REQUEST AND ACTION IS 'ADD' THEN 02660000
* CALL DETAIL PROCESSOR 02670000
* ELSE CALL SECONDARY SELECTION 02680000

IC2008. 02690000
IF ACTION OF INAREA = 'A' THEN 02700000
 02710000
* **DETAIL PROCESSOR 02720000
* GO TO DETAIL0. 02730000
* **SECONDARY SELECTION 02740000
* PERFORM SECSEL THRU END-SECSEL. 02750000
* **IF NO. OF CHOICES = 1 02760000
* **GO TO DETAIL PROCESSOR 02770000
* IF MAXSEL = 1 THEN GO TO DETAIL0. 02780000
GO TO EXIT0. 02790000

* DETERMINE IF VALID SELECTION NUMBER GIVEN 02800000

IC2010. 02810000
* **VALID SELECTION NO. GIVEN 02820000
* IF PREV OF PCONVSTA NOT = 'S' 02830000
 OR MAXSEL < 1 02840000
 OR DATAIN = 'NEXT' 02850000
 OR DATA2 = DAT02 THEN GO TO IC201. 02860000
* IF DAT1 NUMERIC AND DAT2 = ' ' THEN 02870000
 MOVE DAT1 TO DAT2 02880000
 MOVE '0' TO DAT1. 02890000
* **DETAIL SELECTION GIVEN 02900000
* IF DATA2 NUMERIC
 AND DATA2 > '00' AND DATA2 NOT > MAXSEL THEN 02910000
 MOVE 'Y' TO NEWREQ_OF COMPARM. 02920000
 GO TO DETAIL0. 02930000
 02940000
* **INVALID SELECTION NO. 02950000
* **PRINT ERROR MESSAGE 02960000
MOVE '072E' TO MSGCODE. 02970000
CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG. 02980000
MOVE OUTMSG TO MSG OF OUTAREA. 02990000
GO TO EXIT0. 03000000
 03010000
 03020000
MOVE '072E' TO MSGCODE. 03030000
CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG. 03040000
MOVE OUTMSG TO MSG OF OUTAREA. 03050000
GO TO EXIT0. 03060000
 03070000

* DETERMINES WHETHER SECONDARY SELECTION OR DETAIL 03080000

IC201. 03090000
* **SECONDARY SELECTION 03100000
* IF PREV OF PCONVSTA = 'S' THEN 03110000
 PERFORM SECSEL THRU END-SECSEL. 03120000
 GO TO EXIT0. 03130000
 03140000
 03150000
 03160000
* **DETAIL PROCESSOR 03170000
* IF PREV OF PCONVSTA = 'D' THEN GO TO DETAIL0. 03180000

```

```

*
* **LOGIC ERROR 03190000
* **PRINT ERROR MESSAGE 03200000
*
MOVE '066E' TO MSGCODE. 03210000
CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG. 03220000
MOVE OUTMSG TO MSG OF OUTAREA. 03230000
GO TO EXIT0. 03240000
03250000
03260000

* CALLS SECONDARY SELECTION PROCESSOR AND RETURNS TO SQL 1 03270000

SECSEL. 03280000
MOVE 'DSN8001 ' TO LASTSCR IN PCONVSTA. 03290000
IF OBJFLD OF INAREA = 'DS' THEN
*
* **ADMINISTRATIVE 03300000
* **DEPARTMENT STRUCTURE 03310000
* **DEPARTMENT STRUCTURE 03320001
PERFORM DSN8MCA THRU END-DSN8MCA
ELSE
IF OBJFLD OF INAREA = 'DE' THEN
*
* **INDIVIDUAL DEPARTMENT 03330000
* **PROCESSING 03340000
* **PROCESSING 03350000
PERFORM DSN8MCA THRU END-DSN8MCA
ELSE
IF OBJFLD OF INAREA = 'EM' THEN
*
* **INDIVIDUAL EMPLOYEE 03360000
* **PROCESSING 03370001
* **PROCESSING 03380000
* **PROCESSING 03390000
PERFORM DSN8MCA THRU END-DSN8MCA
ELSE
*
* **ERROR MESSAGE 03400000
* **UNSUPPORTED SEARCH 03410000
* **CRITERIA FOR OBJECT 03420001
MOVE '062E' TO MSGCODE
CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
MOVE OUTMSG TO MSG OF OUTAREA
GO TO EXIT0.
END-SECSEL.
03430000
03440000
03450000
03460000
03470000
03480000
03490000
03500000
03510000
03520000
03530000
03540000
03550000
03560000
03570000
03580000
03590000
DETAIL0.
MOVE 'DSN8002 ' TO LASTSCR IN PCONVSTA.
IF OBJFLD OF INAREA = 'DS' THEN
*
* **ADMINISTRATIVE 03600000
* **DEPARTMENT STRUCTURE 03610000
* **DEPARTMENT STRUCTURE 03620001
PERFORM DSN8MCD THRU END-DSN8MCD
ELSE
IF OBJFLD OF INAREA = 'DE' THEN
*
* **INDIVIDUAL DEPARTMENT 03630000
* **PROCESSING 03640000
* **PROCESSING 03650000
PERFORM DSN8MCE THRU END-DSN8MCE
ELSE
IF OBJFLD OF INAREA = 'EM' THEN
*
* **INDIVIDUAL EMPLOYEE 03660000
* **PROCESSING 03670001
* **PROCESSING 03680000
* **PROCESSING 03690000
PERFORM DSN8MCF THRU END-DSN8MCF
ELSE
*
* **ERROR MESSAGE 03700000
* **UNSUPPORTED SEARCH 03710000
* **CRITERIA FOR OBJECT 03720001
MOVE '062E' TO MSGCODE
CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
MOVE OUTMSG TO MSG OF OUTAREA.
GO TO EXIT0.
*
* **HANDLES ERRORS 03730000
* **RETURN TO SQL 1 03740000
EXEC SQL INCLUDE DSN8MCXX END-EXEC.
EXIT0. GOBACK.
03750000
03760000
03770000
03780000
03790000
03800000
03810000
03820000
03830000
03840000
03850000
03860000
03870000
03880000
03890000
03900000
03910000
03920000
03930000

```

## Referencia relacionada

["Ejemplos de aplicaciones en IMS" en la página 1415](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

## DSN8IPO

ESTE MÓDULO RECIBE EL MENSAJE DE ENTRADA Y LO DEFORMA, LLAMA A DSN8IP1, FORMATEA EL MENSAJE DE SALIDA Y LO ENVÍA.

```
DSN8IPO: PROC(IOPCB_ADDR,ALTPCB_ADDR) OPTIONS (MAIN); 00010000
/****** ******/ 00020000
* * 00030000
* MODULE NAME = DSN8IPO * 00040000
* * 00050000
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION * 00060000
* SUBSYSTEM INTERFACE MODULE
* IMS
* PL/I
* ORGANIZATION APPLICATION
*
* COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1985 * 00120000
* REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083 * 00130000
*
* STATUS = RELEASE 2, LEVEL 0 * 00140000
* * 00150000
* * 00160000
* FUNCTION = THIS MODULE RECEIVES INPUT MESSAGE AND DEFORMATS IT, * 00170000
* CALLS DSN8IP1, FORMATS OUTPUT MESSAGE AND SENDS IT * 00180000
*
* NOTES = NONE * 00190000
* * 00200000
* * 00210000
* MODULE TYPE = PL/I PROC OPTIONS(MAIN) * 00220000
* PROCESSOR = PL/I OPTIMIZER * 00230000
* MODULE SIZE = SEE LINKEDIT * 00240000
* ATTRIBUTES = REUSABLE * 00250000
*
* ENTRY POINT = DSN8IPO * 00260000
* PURPOSE = SEE FUNCTION * 00270000
* LINKAGE = FROM IMS * 00280000
* * 00290000
* * 00300000
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION: * 00310000
* * 00320000
* SYMBOLIC LABEL/NAME = DSN8IPGI * 00330000
* DESCRIPTION = IMS/VS MFS GENERAL MENU * 00340000
* * 00350000
* SYMBOLIC LABEL/NAME = DSN8IPDI * 00360000
* DESCRIPTION = IMS/VS MFS SECONDARY SELECTION MENU * 00370000
* * 00380000
* OUTPUT = PARAMETERS EXPLICITLY RETURNED: * 00390000
* * 00400000
* SYMBOLIC LABEL/NAME = DSN8IPGO * 00410000
* DESCRIPTION = IMS/VS MFS GENERAL MENU * 00420000
* * 00430000
* SYMBOLIC LABEL/NAME = DSN8IPDO * 00440000
* DESCRIPTION = IMS/VS MFS SECONDARY SELECTION MENU * 00450000
* * 00460000
*
* EXIT-NORMAL =
* * 00470000
* * 00480000
* EXIT-ERROR =
* * 00490000
* * 00500000
* RETURN CODE = NONE * 00510000
* * 00520000
* ABEND CODES = NONE * 00530000
* * 00540000
* ERROR-MESSAGES =
* DSN8064E - INVALID DL/I STC-CODE ON GU MSG * 00550000
* DSN8065E - INVALID DL/I STC-CODE ON ISRT MSG * 00560000
* * 00570000
* * 00580000
*
* EXTERNAL REFERENCES =
* ROUTINES/SERVICES = MODULE DSN8IP1 * 00590000
* MODULE PLITDLI * 00600000
* MODULE DSN8MPG * 00610000
* * 00620000
* * 00630000
*
* DATA-AREAS =
* DSN8MPCA - PARAMETER TO BE PASSED TO DSN8CP1 * 00640000
* CONTAINS TERMINAL INPUT AND
* OUTPUT AREAS. * 00650000
* * 00660000
* * 00670000
* IN_MESSAGE - MFS INPUT * 00680000
* OUT_MESSAGE - MFS OUTPUT * 00690000
* * 00700000
* * 00710000
* CONTROL-BLOCKS = NONE * 00720000
* * 00730000
* TABLES = NONE * 00740000
* * 00750000
```

```

/*
* *PSEUDOCODE*
*
* PROCEDURE
* DECLARATIONS.
* ALLOCATE PL/I WORK AREA FOR COMMAREA.
* INITIALIZATION.
* PUT MODNAME 'DSN8IPGO' IN MODNAME FIELD.
* PUT MODULE NAME 'DSN8IP0' IN AREA USED BY
* ERROR_HANDLER.
*
* STEP1.
* CALL DLI GU INPUT MESSAGE.
* IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE AND
* STOP PROGRAM.
*
* IF SCREEN CLEARED/UNFORMATTED , MOVE '00' TO PFKIN.
* MOVE INPUT MESSAGE FIELDS TO CORRESPONDING
* INAREA FIELDS IN COMPARM.
* CALL DSN8IP1 (COMMAREA)
* MOVE OUTAREA FIELDS IN PCONVSTA TO CORRESPONDING
* OUTPUT MESSAGE FIELDS.
* IF LASTSCR 'DSN8001' MOVE 'DSN8IPGO' TO MODNAME FIELD
* ELSE MOVE 'DSN8IPDO' TO MODNAME FIELD.
*
* CALL DLI ISRT OUTPUT MESSAGE.
* IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE AND
* STOP PROGRAM.
* END.
*
-----/01070000
1/**/01080000
/* DECLARATION FOR INPUT: MIDNAME DSN8IPGI/DSN8IPDI */01090000
/**/01100000
ODCL 1 IN_MESSAGE STATIC,
 2 LL BIN FIXED (31), 01110000
 2 Z1 CHAR (1), 01120000
 2 Z2 CHAR (1), 01130000
 2 TC_CODE CHAR (7), 01140000
 2 MESSAGE,
 3 INPUT,
 5 MAJSYS CHAR (1), 01150000
 5 ACTION CHAR (1), 01160000
 5 OBJFLD CHAR (2), 01170000
 5 SEARCH CHAR (2), 01180000
 5 PFKIN CHAR (2), 01190000
 5 DATA CHAR (60), 01200000
 5 TRANDATA(15) CHAR (40); 01210000
-/**/01250000
/* DECLARATION FOR OUTPUT: MODNAME DSN8IPGO/DSN8IPDO */01260000
/**/01270000
ODCL 1 OUT_MESSAGE STATIC,
 2 LL BIN FIXED (31) INIT (1613), 01280000
 2 ZZ BIN FIXED (15) INIT (0), 01290000
 2 OUTPUT,
 3 OUTPUTAREA,
 5 MAJSYS CHAR (1), 01300000
 5 ACTION CHAR (1), 01310000
 5 OBJFLD CHAR (2), 01320000
 5 SEARCH CHAR (2), 01330000
 5 DATA CHAR (60), 01340000
 5 TITLE CHAR (50), 01350000
 5 DESC2 CHAR (50), 01360000
 5 DESC3 CHAR (50), 01370000
 5 DESC4 CHAR (50), 01380000
 5 MSG CHAR (79), 01390000
 5 PFKTEXT CHAR (79), 01400000
 5 OUTPUT,
 7 LINE (15) CHAR (79); 01410000
-/**/01460000
/* DECLARATION FOR PASSING INPUT/OUTPUT DATA BETWEEN THE */01470000
/* SUBSYSTEM DEPENDENT MODULE IMS/DL1 AND SQL1 AND SQL2 */01480000
/**/01490000
EXEC SQL INCLUDE DSN8MPCA;
01500000
01510000
DCL DSN8MPG EXTERNAL ENTRY;
01520000
01530000
1/**/01540000
/* FIELDS SENT TO MESSAGE ROUTINE */01550000
/**/01560000
DCL MODULE CHAR(07) INIT('DSN8IP0'); 01570000

```

```

DCL OUTMSG CHAR(69); 01580000
1******/ 01590000
/* DECLARATION FOR PGM-LOGIC */01600000
/******/ 01610000
0DCL ONE BIN FIXED(31) INIT(1) STATIC; 01620000
DCL THREE BIN FIXED(31) INIT(3) STATIC; 01630000
DCL FOUR BIN FIXED(31) INIT(4) STATIC; 01640000
0DCL GU_FKT CHAR(4) INIT('GU') STATIC; 01650000
DCL ISRT_FKT CHAR(4) INIT('ISRT') STATIC; 01660000
DCL CHNG_FKT CHAR(4) INIT('CHNG') STATIC; 01670000
DCL ROLL_FKT CHAR(4) INIT('ROLL') STATIC; 01680000
0DCL MODNAME CHAR(8) STATIC; 01690000
0DCL (ADDR,LOW) BUILTIN; 01700000
0DCL PLITDLI EXTERNAL ENTRY; 01710000
DCL DSN8IP1 EXTERNAL ENTRY; 01720000
0DCL (IOPCB_ADDR,ALTPCB_ADDR) POINTER; 01730000
1******/ 01740000
/* DECLARATION FOR IO / ALTPCB MASK */01750000
/******/ 01760000
0DCL 1 IOPCB BASED(IOPCB_ADDR), 01770000
 2 IOLTERM CHAR(8), 01780000
 2 FILLER CHAR(2), 01790000
 2 STC_CODE CHAR(2), 01800000
 2 CDATE CHAR(4), 01810000
 2 CTIME CHAR(4), 01820000
 2 SEQNUM CHAR(4), 01830000
 2 MOD_NAME CHAR(8), 01840000
 2 USERID CHAR(8); 01850000
0DCL 1 ALTPCB BASED(ALTPCB_ADDR), 01860000
 2 ALTLTERM CHAR(8), 01870000
 2 FILLER CHAR(2), 01880000
 2 STC_CODE CHAR(2); 01890000
******/ 01900000
/* ALLOCATE COBOL WORK AREA /INITIALIZATIONS */01910000
/******/ 01920000
01930000
ALLOCATE COMMAREA SET(COMMPTR); 01940000
COMMAREA = ''; /* CLEAR COMMON AREA*/ 01950000
IN_MESSAGE = ''; /* CLEAR INPUT FIELD*/ 01960000
MODNAME = 'DSN8IPGO'; /* GET MODULE NAME */ 01970000
DSN8_MODULE_NAME.MAJOR = 'DSN8IP0'; /* GET MODULE NAME */ 01980000
OUTAREA.MAJSYS = '0'; /* MAJOR SYSTEM - 0 */ 01990000
EXITCODE = '0'; /* CLEAR EXIT CODE */ 02000000
02010000
******/ 02020000
/* CALL DL1 GU INPUT MESSAGE */02030000
/* PRINT ERROR MESSAGE IF STATUS CODE NOT OK */02040000
/******/ 02050000
02060000
0 CALL PLITDLI (THREE, GU_FKT, IOPCB, IN_MESSAGE); /*CALL DL1 GU */ 02070000
02080000
0 IF IOPCB.STC_CODE ^= ' ' THEN /* ERROR ? */ 02090000
 DO;
 CALL DSN8MPG (MODULE, '064E', OUTMSG);
 OUTPUTAREA.MSG = OUTMSG;
 02100000
 GO TO CSEND; /*CALL DL1 ISRT OUTPUT MESSAGE */ 02150000
 END; 02160000
 02170000
******/ 02180000
/* CLEARED AND UNFORMATTED SCREEN? */02190000
/******/ 02200000
02210000
0 IF Z2 = LOW(1) THEN COMPARM.PFKIN = '00';
 PCONVSTA.CONVID = IOPCB.IOLTERM||USERID; 02220000
 02230000
 02240000
 INAREA = INPUT, BY NAME; /*MOVE INPUT MESSAGE */ 02250000
 INAREA.MAJSYS = '0'; /*FIELDS TO INAREA FIELDS*/ 02260000
 02270000
0 CALL DSN8IP1 (COMMPTR); 02280000
 02290000
 /*MOVE OUTAREA FIELDS */ 02300000
0 OUTPUTAREA = OUTAREA, BY NAME; /*TO OUTPUT MESSAGE FIELDS*/ 02310000
0 IF LASTSCR = 'DSN8002' THEN MODNAME = 'DSN8IP0';
 ELSE MODNAME = 'DSN8IPGO'; 02320000
 02330000
 02340000
******/ 02350000
/* CALL DL ISRT OUTPUT MESSAGE */02360000
/* PRINT ERROR MESSAGE IF STATUS CODE NOT OK */02370000
/******/ 02380000
02390000

```

```

CSEND:
 /*CALL DL1 ISRT*/ 02400000
 CALL PLITDLI (FOUR,ISRT_FKT,IOPCB,OUT_MESSAGE,MODNAME); 02410000
0 IF IOPCB.STC_CODE = ' ' THEN GO TO CEND; /*STATUS CODE OK*/ 02420000
 /*STATUS CODE NOT OK*/ 02430000
0 CALL DSN8MPG (MODULE, '065E', OUTMSG); 02440000
0 OUTPUTAREA.MSG = OUTMSG; /*PRINT ERROR MESSAGE*/ 02450000
0 CALL PLITDLI (THREE,CHNG_FKT,ALTPCB,IOLTERM); /* CALL DL1 CHNG */ 02460000
0 CALL PLITDLI (FOUR,ISRT_FKT,ALTPCB,OUT_MESSAGE,MODNAME); 02470000
0 OUTPUTAREA.MSG = OUTMSG; /*PRINT ERROR MESSAGE*/ 02480000
0 CALL PLITDLI (ONE,ROLL_FKT); /* PERFORM ROLLBACK*/ 02490000
0CSEND1: /* RETURN */ 02500000
0 CALL PLITDLI (ONE,ROLL_FKT); /* PERFORM ROLLBACK*/ 02510000
0 IF ALTPCB.STC_CODE ^= ' ' THEN GO TO CSEND1; /* ERROR? */ 02520000
 /* CALL DL1 ISRT */ 02530000
0 CALL PLITDLI (FOUR,ISRT_FKT,ALTPCB,OUT_MESSAGE,MODNAME); 02540000
0 CALL PLITDLI (FOUR,ISRT_FKT,ALTPCB,OUT_MESSAGE,MODNAME); 02550000
0 OUTPUTAREA.MSG = OUTMSG; /*PRINT ERROR MESSAGE*/ 02560000
0CSEND1: /* PERFORM ROLLBACK*/ 02570000
0 CALL PLITDLI (ONE,ROLL_FKT); /* PERFORM ROLLBACK*/ 02580000
0 OCEND: /* RETURN */ 02590000
0 END DSN8IP0; /* RETURN */ 02600000
 /* RETURN */ 02610000

```

### Referencia relacionada

["Ejemplos de aplicaciones en IMS" en la página 1415](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

## DSN8IP1

LA EJECUCIÓN INCLUYE INTRODUCIR LAS ESTRUCTURAS DCLS Y DCLGEN DE LA TABLA SQL, ASÍ COMO EL ÁREA DE PARÁMETROS.

```

DSN8IP1:PROC (COMMTPTR) ;
/***
*
* MODULE NAME = DSN8IP1
*
* DESCRIPTIVE NAME = SAMPLE APPLICATION
* SQL 1 MAINLINE
* IMS
* PL/I
*
* COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1985
* REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
*
* STATUS = RELEASE 2, LEVEL 0
*
* FUNCTION = PERFORM INCLUDES TO BRING IN SQL TABLE DCLS AND
* DCLGEN STRUCTURES AS WELL AS PARAMETER AREA.
* INCLUDE DSN8MP1.
* CALL DSN8IP2.
* RETURN TO DSN8IP0.
*
* NOTES = NONE
*
* MODULE TYPE = PL/I PROC(COMMTPTR).
* PROCESSOR = DB2 PRECOMPILER, PL/I OPTIMIZER
* MODULE SIZE = SEE LINKEDIT
* ATTRIBUTES = REUSABLE
*
* ENTRY POINT = DSN8IP1
* PURPOSE = SEE FUNCTION
* LINKAGE = CALLED BY DSN8IP0
*
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
*
* SYMBOLIC LABEL/NAME = COMMTPTR
* DESCRIPTION = POINTER TO COMMUNICATION AREA
*
* COMMON AREA.
*
* SYMBOLIC LABEL/NAME = PFKIN
* DESCRIPTION = 00/01/02/03/08/10
*
* SYMBOLIC LABEL/NAME = INAREA
* DESCRIPTION = USER INPUT
*
* OUTPUT = PARAMETERS EXPLICITLY RETURNED:
*
```

```

* COMMON AREA.
*
* SYMBOLIC LABEL/NAME = OUTAREA
* DESCRIPTION = GENERAL MENU OR SECONDARY
* SELECTION MENU
*
* SYMBOLIC LABEL/NAME = LASTSCR
* DESCRIPTION = DSN8001/DSN8002
*
* EXIT-NORMAL = DSN8IP0
*
* EXIT-ERROR = DSN8IP0
*
* RETURN CODE = NONE
*
* ABEND CODES = NONE
*
* ERROR-MESSAGES = NONE
*
* EXTERNAL REFERENCES =
* ROUTINES/SERVICES = NONE
*
* DATA-AREAS =
* DSN8MPCA - PLI STRUCTURE FOR COMMAREA
* DSN8MPCS - VCONA TABLE DCL AND PCONA DCLGEN
* DSN8MPOV - VOPTVAL TABLE DCL & POPTVAL DCLGEN
* DSN8MPVO - VALIDATION CURSORS
* DSN8MP1 - SQL1 COMMON MODULE FOR IMS AND CICS
* DSN8MP3 -- DSN8MP5 - VALIDATION MODULES CALLED BY DSN8MP1
* DSN8MPXX - SQL ERROR HANDLER
*
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA
*
* TABLES = NONE
*
* CHANGE-ACTIVITY = NONE
*
* *PSEUDOCODE*
*
* PROCEDURE
* INCLUDE DECLARATIONS.
* INCLUDE DSN8MP1.
* INCLUDE ERROR HANDLER.
*
* CP1EXIT: (REFERENCED BY DSN8MP1)
* RETURN.
*
* CP1CALL: (REFERENCED BY DSN8MP1)
* CALL 'DSN8IP2'(COMM PTR).
* GO TO MP1SAVE. (LABEL IN DSN8MP1)
*
* INCLUDE VALIDATION MODULES.
*
* END.

1/*****
/* SQL1 MAINLINE */
/*****
DCL STRING BUILTIN;
DCL J FIXED BIN;
DCL SAVE_CONVID CHAR(16);
DCL (SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1);
 /* SQL RETURN CODE HANDLING */
EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR;
EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;

/*****
/* FIELDS PASSED TO MESSAGE ROUTINE */
/*****
DCL MODULE CHAR(07);
DCL OUTMSG CHAR(69);

DCL DSN8IP2 EXTERNAL ENTRY;
DCL DSN8MPG EXTERNAL ENTRY;
EXEC SQL INCLUDE DSN8MPCA; /* INCLUDE COMMAREA */
DSN8_MODULE_NAME.MAJOR = 'DSN8IP1'; /* INITIALIZE MODULE NAME*/
EXEC SQL INCLUDE DSN8MPCS; /* INCLUDE PCONA */
EXEC SQL INCLUDE DSN8MPOV; /* INCLUDE VOPTVAL */
EXEC SQL INCLUDE DSN8MPVO; /* INCLUDE CURSOR */
EXEC SQL INCLUDE SQLCA; /* SQL COMMON AREA */

```

```

EXEC SQL INCLUDE DSN8MP1; /* INCLUDE SQL1 MAIN*/
EXEC SQL INCLUDE DSN8MPXX; /* HANDLES ERRORS */

CP1EXIT : /* STANDARD EXIT */
 RETURN;

CP1CALL : /* GO TO DSN8IP2 (SQL2) */
 CALL DSN8IP2 (COMMPTR);
 GO TO MP1SAVE;

EXEC SQL INCLUDE DSN8MP3; /* INCLUDE ACTION VALIDATION*/
EXEC SQL INCLUDE DSN8MP4; /* INCLUDE OBJECT VALIDATION*/
EXEC SQL INCLUDE DSN8MP5; /* INCLUDE SEARCH CRITERIA*/
END; /* VALIDATION */

```

### Referencia relacionada

[“Ejemplos de aplicaciones en IMS” en la página 1415](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

## DSN8IP2

ROUTER PARA SELECCIÓN SECUNDARIA Y/O PROCESAMIENTO DE DETALLES LLAMA A MÓDULOS DE SELECCIÓN SECUNDARIA DSN8MPA LLAMA A MÓDULOS DE DETALLES DSN8MPD DSN8MPE DSN8MPF LLAMADA POR DSN8IP1 (SQL1).

```

DSN8IP2: PROC(COMMPTR) ; /* SQL 2 FOR IMS AND PLI */ 00010000
 %PAGE; 00020000
/****** 00030000
* * 00040000
* MODULE NAME = DSN8IP2 * 00050000
* * 00060000
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION * 00070000
* SQL 2 COMMON MODULE * 00080000
* IMS * 00090000
* PL/I * 00100000
* ORGANIZATION APPLICATION * 00110000
* * 00120000
* LICENSED MATERIALS - PROPERTY OF IBM * 00130000
* 5695-DB2 * 00136000
* (C) COPYRIGHT 1982, 1995 IBM CORP. ALL RIGHTS RESERVED. * 00143000
* * 00150000
* STATUS = VERSION 4 * 00160000
* * 00170000
* FUNCTION = ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSIN* 00180000
* CALLS SECONDARY SELECTION MODULES * 00190000
* DSN8MPA * 00200000
* CALLS DETAIL MODULES * 00210000
* DSN8MPD DSN8MPE DSN8MPF * 00220000
* CALLED BY DSN8IP1 (SQL1) * 00230000
* * 00240000
* NOTES = NONE * 00250000
* * 00260000
* * 00270000
* MODULE TYPE = BLOCK OF PL/I CODE * 00280000
* PROCESSOR = DB2 PRECOMPILER, PL/I OPTIMIZER * 00290000
* MODULE SIZE = SEE LINKEDIT * 00300000
* ATTRIBUTES = REUSABLE * 00310000
* * 00320000
* ENTRY POINT = DSN8IP2 * 00330000
* PURPOSE = SEE FUNCTION * 00340000
* LINKAGE = CALL DSN8IP2(COMMPTR) * 00350000
* INPUT = * 00360000
* * 00370000
* SYMBOLIC LABEL/NAME = COMMPTR * 00380000
* DESCRIPTION = POINTER TO COMMUNICATION AREA * 00390000
* * 00400000
* OUTPUT = * 00410000
* * 00420000
* SYMBOLIC LABEL/NAME = COMMPTR * 00430000
* DESCRIPTION = POINTER TO COMMUNICATION AREA * 00440000
* * 00450000
* EXIT-NORMAL = * 00460000
* * 00470000
* EXIT-ERROR = IF SQLERROR OR SQLWARNING, SQL WHENEVER CONDITION * 00480000
* SPECIFIED IN DSN8IP2 WILL BE RAISED AND PROGRAM * 00490000
* WILL GO TO THE LABEL DB_ERROR. * 00500000

```

```

*
*
* RETURN CODE = NONE * 00510000
*
* ABEND CODES = NONE * 00520000
*
* ERROR-MESSAGES =
* DSN8062E-AN OBJECT WAS NOT SELECTED * 00530000
* DSN8066E-UNSUPPORTED PFK OR LOGIC ERROR * 00540000
* DSN8072E-INVALID SELECTION ON SECONDARY SCREEN * 00550000
*
* EXTERNAL REFERENCES =
*
* ROUTINES/SERVICES = MODULES LISTED ABOVE
* DSN8MPG - ERROR MESSAGE ROUTINE * 00560000
*
* DATA-AREAS =
* DSN8MPA - SECONDARY SELECTION FOR ORGANIZATION * 00570000
* DSN8MPAD - DECLARE ADMINISTRATIVE DETAIL * 00580000
* DSN8MPAE - CURSOR EMPLOYEE LIST * 00590000
* DSN8MPAL - CURSOR ADMINISTRATION LIST * 00600000
* DSN8MPA2 - DECLARE ADMINISTRATIVE DETAIL * 00610000
* DSN8MPCA - DECLARE SQL COMMON AREA * 00620000
* DSN8MPD - DEPARTMENT STRUCTURE DETAIL * 00630000
* DSN8MPDA - CURSOR ADMINISTRATION DETAIL * 00640000
* DSN8MPDH - CURSOR FOR DISPLAY TEXT FROM * 00650000
* TDSPTXT TABLE * 00660000
* DSN8MPDM - DECLARE DEPARTMENT MANAGER * 00670000
* DSN8MPDP - DELCLARE DEPARTMENT * 00680000
* DSN8MPDT - DECLARE DISPLAY TEXT * 00690000
* DSN8MPE - DEPARTMENT DETAIL * 00700000
* DSN8MPEM - DECLARE EMPLOYEE * 00710000
* DSN8MPED - DECLARE EMPLOYEE-DEPARTMENT * 00720000
* DSN8MPF - EMPLOYEE DETAIL * 00730000
* DSN8MPOV - DECLARE OPTION VALIDATION * 00740000
*
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA * 00750000
*
* TABLES = NONE * 00760000
*
* CHANGE-ACTIVITY = NONE * 00770000
*
* *PSEUDOCODE*
*
* THIS MODULE DETERMINES WHICH SECONDARY SELECTION AND/OR
* DETAIL MODULE(S) ARE TO BE CALLED FOR THE IMS/PL1 ENVIRONMENT.* 00780000
*
* WHAT HAS HAPPENED SO FAR?..... THE SUBSYSTEM * 00790000
* DEPENDENT MODULE (IMS,CICS) (SQL 0) HAS READ THE * 00800000
* INPUT SCREEN, FORMATTED THE INPUT, AND PASSED CONTROL * 00810000
* TO SQL 1. SQL 1 PERFORMS VALIDATION ON THE SYSTEM DEPENDENT * 00820000
* FIELDS (MAJOR SYSTEM, ACTION, OBJECT, SEARCH CRITERIA). IF * 00830000
* ALL SYSTEM FIELDS ARE VALID, SQL 1 PASSED CONTROL TO THIS * 00840000
* MODULE. PASSED PARAMETERS CONSIST ONLY OF A POINTER WHICH * 00850000
* POINTS TO A COMMUNICATION CONTROL AREA USED TO COMMUNICATE * 00860000
* BETWEEN SQL 0 , SQL 1, SQL 2, AND THE SECONDARY SELECTION * 00870000
* AND DETAIL MODULES. * 00880000
*
* WHAT IS INCLUDED IN THIS MODULE?..... * 00890000
* ALL SECONDARY SELECTION AND DETAIL MODULES ARE 'INCLUDED'. * 00900000
* ALL VARIABLES KNOWN IN THIS PROCEDURE ARE KNOWN IN THE * 00910000
* SUB PROCEDURES. ALL SQL CURSOR DEFINITIONS AND * 00920000
* SQL 'INCLUDES' ARE DONE IN THIS PROCEDURE. ALL CURSOR HOST * 00930000
* VARIABLES ARE DECLARED IN THIS PROCEDURE BECAUSE OF THE * 00940000
* RESTRICTION THAT CURSOR HOST VARIABLES MUST BE DECLARED BEFORE* 00950000
* THE CURSOR DEFINITION. * 00960000
*
* PROCEDURE
* IF ANSWER TO DETAIL SCREEN & DETAIL PROCESSOR * 00970000
* IS NOT WILLING TO ACCEPT AN ANSWER THEN * 00980000
* NEW REQUEST* * 00990000
*
* ELSE
* IF ANSWER TO A SECONDARY SELECTION THEN * 01000000
* DETERMINE IF NEW REQUEST. * 01010000
*
* CASE (NEW REQUEST)
* SUBCASE ('ACTION') * 01020000
* DETAIL PROCESSOR * 01030000
* RETURN TO SQL 1 * 01040000
*
* END CASE
*
* END IF
*
* END PROCEDURE

```

```

* ENDSUB * 01330000
* *
* SUBCASE ('DISPLAY','ERASE','UPDATE') * 01340000
* CALL SECONDARY SELECTION * 01350000
* IF # OF POSSIBLE CHOICES IS ^= 1 THEN * 01360000
* RETURN TO SQL 1 * 01370000
* ELSE * 01380000
* CALL THE DETAIL PROCESSOR * 01390000
* RETURN TO SQL 1. * 01400000
* ENDSUB * 01410000
* *
* ENDCASE * 01420000
* *
* IF ANSWER TO SECONDARY SELECTION AND A SELECTION HAS
* ACTUALLY BEEN MADE THEN * 01430000
* VALID SELECTION #?
* IF IT IS VALID THEN * 01440000
* CALL DETAIL PROCESSOR * 01450000
* RETURN TO SQL 1 * 01460000
* ELSE * 01470000
* PRINT ERROR MSG * 01480000
* RETURN TO SQL 1. * 01490000
* *
* IF ANSWER TO SECONDARY SELECTION THEN * 01500000
* CALL SECONDARY SELECTION * 01510000
* RETURN TO SQL 1. * 01520000
* *
* IF ANSWER TO DETAIL THEN * 01530000
* CALL DETAIL PROCESSOR * 01540000
* RETURN TO SQL 1. * 01550000
* *
* IF ANSWER TO DETAIL THEN * 01560000
* CALL DETAIL PROCESSOR * 01570000
* RETURN TO SQL 1. * 01580000
* *
* IF ANSWER TO DETAIL THEN * 01590000
* CALL DETAIL PROCESSOR * 01600000
* RETURN TO SQL 1. * 01610000
* *
* IF ANSWER TO DETAIL THEN * 01620000
* CALL DETAIL PROCESSOR * 01630000
* RETURN TO SQL 1. * 01640000
* *
* END. * 01650000
* *
* *EXAMPLE- A ROW IS SUCCESSFULLY ADDED, THE OPERATOR RECEIVES
* THE SUCCESSFULLY ADDED MESSAGE AND JUST HITS ENTER. * 01660000
-----/ * 01670000
-----/ * 01680000
-----/ 01690000
* DCL DSN8MPG EXTERNAL ENTRY; 01700000
* DCL LENGTH BUILTIN; 01710000
* *
* 01720000
* /* INCLUDE DECLARES */
* EXEC SQL INCLUDE DSN8MPCA; /*COMMUNICATION AREA BETWEEN MODULES */ 01730000
* EXEC SQL INCLUDE SQLCA; /*SQL COMMUNICATION AREA */ 01740000
* /* ORGANIZATION
* EXEC SQL INCLUDE DSN8MPDP; /* DCLGEN FOR DEPARTMENT */ 01750000
* EXEC SQL INCLUDE DSN8MPEM; /* DCLGEN FOR EMPLOYEE */ 01760000
* EXEC SQL INCLUDE DSN8MPED; /* DCLGEN FOR EMPLOYEE-DEPARTMENT */ 01770000
* EXEC SQL INCLUDE DSN8MPDM; /* DCLGEN FOR DEPARTMENT/MANAGER */ 01780000
* EXEC SQL INCLUDE DSN8MPAD; /* DCLGEN FOR ADMINISTRATION DETAIL */ 01790000
* EXEC SQL INCLUDE DSN8MPA2; /* DCLGEN FOR ADMINISTRATION DETAIL */ 01800000
* /* PROGRAMMING TABLES
* EXEC SQL INCLUDE DSN8MPOV; /* DCLGEN FOR OPTION VALIDATION */ 01810000
* EXEC SQL INCLUDE DSN8MPDT; /* DCLGEN FOR DISPLAY TEXT TABLE */ 01820000
* 01830000
* /* CURSORS */
* EXEC SQL INCLUDE DSN8MPAL; /* MAJSYS O - SEC SEL FOR DS AND DE */ 01840000
* EXEC SQL INCLUDE DSN8MPAE; /* MAJSYS O - SEC SEL FOR EM */ 01850000
* EXEC SQL INCLUDE DSN8MPDA; /* MAJSYS O - DETAIL FOR DS */ 01860000
* EXEC SQL INCLUDE DSN8MPDH; /* PROG TABLES - DISPLAY HEADINGS */ 01870000
* 01880000
* 01890000
* 01900000
* 01910000
* ****
* ** FIELDS SENT TO MESSAGE ROUTINE
* ****
* 01920000
* 01930000
* 01940000
* 01950000
* DCL MODULE CHAR (07) INIT ('DSN8IP2'); 01960000
* DCL OUTMSG CHAR (69); 01970000
* 01980000
* ****
* /* SQL RETURN CODE HANDLING
* ****
* 01990000
* 02000000
* 02010000
* 02020000
* EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR; 02030000
* EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR; 02040000
* 02050000
* 02060000
* 02070000
* 02080000
* ****
* /* INITIALIZATIONS
* ****
* 02090000
* 02100000
* 02110000
* 02120000
* DSN8_MODULE_NAME.MAJOR='DSN8IP2'; 02130000

```

```

DSN8_MODULE_NAME.MINOR= ' ' ; 02140000
 02150000
/******DETERMINES WHETHER NEW REQUEST OR NOT***** 02160000
/* DETERMINES WHETHER NEW REQUEST OR NOT 02170000
/******DETERMINES WHETHER NEW REQUEST OR NOT***** 02180000
 02190000
/* IF 'NO ANSWER POSSIBLE' SET BY DETAIL PROCESSOR THEN FORCE A */ 02200000
/* NEW REQUEST. */ 02210000
 02220000
IF PCONVSTA.PREV = ' ' THEN 02230000
 COMPARM.NEWREQ = 'Y'; 02240000
 02250000
/* IF ANSWER TO SECONDARY SELECTION THEN DETERMINE IF REALLY A */ 02260000
/* NEW REQUEST. IT WILL BE CONSIDERED A NEW REQUEST IF POSITIONS*/ 02270000
/* 3 TO 60 ARE NOT ALL BLANK AND THE ENTERED DATA IF NOT 'NEXT' */ 02280000
 02290000
IF COMPARM.NEWREQ = 'N' & PCONVSTA.PREV = 'S' & 02300000
 SUBSTR(COMPARM.DATA,3,58) ^= ' ' & 02310000
 COMPARM.DATA ^= 'NEXT' 02320000
 THEN COMPARM.NEWREQ = 'Y'; 02330000
 02340000
/******DETERMINES IF NEW REQUEST AND ACTION IS 'ADD' THEN***** 02350000
/* IF NEW REQUEST AND ACTION IS 'ADD' THEN 02360000
/* CALL DETAIL PROCESSOR 02370000
/* ELSE CALL SECONDARY SELECTION 02380000
/******DETERMINES IF NEW REQUEST AND ACTION IS 'ADD' THEN***** 02390000
 02400000
IF COMPARM.NEWREQ='Y' THEN 02410000
DO;
 IF COMPARM.ACTION = 'A' THEN 02420000
 DO;
 CALL DETAIL; /* CALL DETAIL PROCESSOR */ 02430000
 GO TO EXIT; /* RETURN */ 02440000
 END;
 CALL SECSEL; /* CALL SECONDARY SELECTION */ 02450000
 GO TO EXIT; /* RETURN */ 02460000
 END;
 CALL SECSEL; /* CALL SECONDARY SELECTION */ 02470000
 GO TO EXIT; /* RETURN */ 02480000
END;

IF MAXSEL = 1 THEN /* IF NO. OF CHOICES = 1 */ 02490000
 CALL DETAIL; /* CALL DETAIL PROCESSOR */ 02500000
 GO TO EXIT; /* RETURN */ 02510000
END;
CALL SECSEL; /* CALL SECONDARY SELECTION */ 02520000
GO TO EXIT; /* RETURN */ 02530000
END;
CALL SECSEL; /* CALL SECONDARY SELECTION */ 02540000
GO TO EXIT; /* RETURN */ 02550000
/* IF ANSWER TO SECONDARY SELECTION AND NOT A SCROLLING REQUEST */ 02560000
/* (INPUT NOT EQUAL TO 'NEXT') AND THE POSITIONS */ 02570000
/* 1 TO 2 IN INPUT DATA FIELD NOT EQUAL TO POSITIONS 1 TO 2 */ 02580000
/* IN OUTPUT DATA FIELD THEN SEE IF VALID SELECTION. */ 02590000
 02600000
/******DETERMINES IF VALID SELECTION NUMBER***** 02610000
/* DETERMINES IF VALID SELECTION NUMBER */ 02620000
/******DETERMINES IF VALID SELECTION NUMBER***** 02630000
 02640000
IF PCONVSTA.PREV ^= 'S' THEN GO TO IP201; /* TO SECONDARY SEL */ 02650000
IF PCONVSTA.MAXSEL < 1 THEN GO TO IP201; /* NO VALID CHOICES */ 02660000
IF COMPARM.DATA = 'NEXT' THEN GO TO IP201; /* SCROL REQUEST*/ 02670000
02680000
IF SUBSTR(COMPARM.DATA,1,2) = SUBSTR(PCONVSTA.DATA,1,2) 02690000
 THEN GO TO IP201; /* NO CHANGE ON INPUT SCREEN */ 02700000
02710000
IF SUBSTR(COMPARM.DATA,2,1) = ' ' THEN /* SECOND CHAR BLANK */ 02720000
 IF VERIFY(SUBSTR(COMPARM.DATA,1,1),'123456789') = 0 THEN DO;
 SUBSTR(COMPARM.DATA,2,1) = SUBSTR(COMPARM.DATA,1,1); 02730000
 SUBSTR(COMPARM.DATA,1,1) = '0';
 END;
02740000
IF VERIFY(SUBSTR(COMPARM.DATA,1,2),'0123456789') = 0 & 02750000
 SUBSTR(COMPARM.DATA,1,2) > '00' THEN DO;
 IF DATAP <= PCONVSTA.MAXSEL THEN DO;
 COMPARM.NEWREQ = 'Y'; /* TELL DETAIL PROCESSOR NEW REQ */ 02760000
 CALL DETAIL; /* CALL DETAIL PROCESSOR */ 02770000
 GO TO EXIT; /* RETURN */ 02780000
 END;
02790000
 /* INVALID SELECTION NO. */ 02800000
 /* PRINT ERROR MESSAGE */ 02810000
02820000
CALL DSN8MPG (MODULE, '072E', OUTMSG);
PCONVSTA.MSG = OUTMSG; 02830000
02840000
GO TO EXIT; /* RETURN */ 02850000
02860000
02870000
02880000
02890000
02900000
02910000
02920000
02930000
02940000
02950000
/******DETERMINES IF VALID SELECTION NUMBER***** 02960000

```

```

/* DETERMINES WHETHER SECONDARY SELECTION OR DETAIL */ 02960000
/***/ 02970000
02980000
/* MUST BE ANY ANSWER TO EITHER SEC SEL OR DETAIL */ 02990000
IP201: 03000000
 IF PCONVSTA.PREV = 'S' THEN 03010000
 DO; 03020000
 CALL SECSEL; /* CALL SECONDARY SELECTION */ 03030000
 GO TO EXIT; /* RETURN */ 03040000
 END; 03050000
 IF PCONVSTA.PREV = 'D' THEN 03060000
 DO; 03070000
 CALL DETAIL; /* CALL DETAIL PROCESSOR */ 03080000
 GO TO EXIT; /* RETURN */ 03090000
 END; 03100000
 /* LOGIC ERROR */ 03110000
CALL DSN8MPG (MODULE, '066E', OUTMSG); /* PRINT ERROR MESSAGE*/ 03120000
PCONVSTA.MSG = OUTMSG; 03130000
GO TO EXIT; 03140000
03150000
03160000
EXEC SQL INCLUDE DSN8MPXX; /* HANDLES SQL ERRORS */ 03170000
GO TO EXIT; /* RETURN */ 03180000
03190000
/***/ 03200000
/* CALLS SECONDARY SELECTION AND RETURNS TO SQL 1 */ 03210000
/* NOTE - SAME SECONDARY SELECTION MODULE FOR DS, DE AND EM */ 03220000
/***/ 03230000
03240000
SECSEL: PROC; /* CALL APPROPRIATE SECONDARY SELECTION MODULE */ 03250000
 PCONVSTA.LASTSCR = 'DSN8001'; /* NOTE GENERAL SCREEN */ 03260000
 IF COMPARM.OBJFLD='DS' | /* DEPARTMENT STRUCTURE*/ 03270002
 03280000
 COMPARM.OBJFLD='DE' | /* INDIVIDUAL DEPARTMENT*/ 03290002
 03300000
 COMPARM.OBJFLD='EM' THEN /* INDIVIDUAL EMPLOYEE */ 03310002
 DO;
 CALL DSN8MPA;
 RETURN;
 END; 03320000
 /* MISSING SECONDARY SEL*/ 03330000
CALL DSN8MPG (MODULE, '062E', OUTMSG); /* PRINT ERROR MESSAGE*/ 03340000
PCONVSTA.MSG = OUTMSG; 03350000
GO TO EXIT; 03360000
03370000
03380000
03390000
03400000
03410000
03420000
03430000
03440000
03450000
DETAIL: PROC; /* CALL APPROPRIATE DETAIL MODULE */ 03460000
 PCONVSTA.LASTSCR = 'DSN8002'; /* NOTE DETAIL SCREEN */ 03470000
 03480000
 IF COMPARM.OBJFLD='DS' THEN /* ADMINISTRATIVE */ 03490002
 DO; 03500000
 CALL DSN8MPD;
 RETURN;
 END; 03510000
 /* DEPARTMENT STRUCTURE */ 03520000
 03530000
 03540000
 IF COMPARM.OBJFLD='DE' THEN /* INDIVIDUAL DEPARTMENT */ 03550002
 DO; 03560000
 CALL DSN8MPE;
 RETURN;
 END; 03570000
 /* PROCESSING */ 03580000
 03590000
 03600000
 IF COMPARM.OBJFLD='EM' THEN /* INDIVIDUAL EMPLOYEE */ 03610002
 DO; 03620000
 CALL DSN8MPF;
 RETURN;
 END; 03630000
 /* PROCESSING */ 03640000
 03650000
 03660000
 03670000
 03680000
 03690000
 03700000
 03710000
 03720000
03730000
 03740000
 03750000
 03760000
 03770000

```

```

EXEC SQL INCLUDE DSN8MPF; /* DETAIL - EMPLOYEES */
END; /* DSN8IP2 */ 03780000
 */ 03790000

```

## Referencia relacionada

[“Ejemplos de aplicaciones en IMS” en la página 1415](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

## DSN8IP6

ESTE MÓDULO RECIBE EL MENSAJE DE ENTRADA Y LO DEFORMA, LLAMA A DSN8IP7, FORMATEA EL MENSAJE DE SALIDA Y LO ENVÍA.

```

DSN8IP6: PROC(IOPCB_ADDR,ALTPCB_ADDR) OPTIONS (MAIN);
/***/
/*
* MODULE NAME = DSN8IP6
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
* SUBSYSTEM INTERFACE MODULE
* IMS
* PL/I
* PROJECT
*
* COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1985
* REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
*
* STATUS = RELEASE 2, LEVEL 0
*
* FUNCTION = THIS MODULE RECEIVES INPUT MESSAGE AND DEFORMATS IT,
* CALLS DSN8IP7, FORMATS OUTPUT MESSAGE AND SENDS IT.
*
* NOTES = NONE
*
* MODULE TYPE = PL/I PROC OPTIONS(MAIN)
* PROCESSOR = PL/I OPTIMIZER
* MODULE SIZE = SEE LINKEDIT
* ATTRIBUTES = REUSABLE
*
* ENTRY POINT = DSN8IP6
* PURPOSE = SEE FUNCTION
* LINKAGE = FROM IMS
*
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
* COMMON AREA:
*
* SYMBOLIC LABEL/NAME = COMPARM.PFKIN
* DESCRIPTION = 00/01/02/03/08/10
*
* SYMBOLIC LABEL/NAME = COMPARM.INAREA
* DESCRIPTION = USER INPUT
*
* INPUT-MESSAGE:
*
* SYMBOLIC LABEL/NAME = DSN8IPFI
* DESCRIPTION = GENERAL MENU
*
* SYMBOLIC LABEL/NAME = DSN8IPEI
* DESCRIPTION = SECONDARY SELECTION MENU
*
* OUTPUT = PARAMETERS EXPLICITLY RETURNED:
* COMMON AREA:
*
* SYMBOLIC LABEL/NAME = COMPARM.OUTAREA
* DESCRIPTION = USER OUTPUT
*
* SYMBOLIC LABEL/NAME = COMPARM.LASTSCR
* DESCRIPTION = DSN8001/DSN8002
*
* OUTPUT-MESSAGE:
*
* SYMBOLIC LABEL/NAME = DSN8IPFO
* DESCRIPTION = GENERAL MENU
*
* SYMBOLIC LABEL/NAME = DSN8IPEO
* DESCRIPTION = SECONDARY SELECTION MENU
*
* EXIT-NORMAL =
*
* 00010000
* 00020000
* 00030000
* 00040000
* 00050000
* 00060000
* 00070000
* 00080000
* 00090000
* 00100000
* 00110000
* 00120000
* 00130000
* 00140000
* 00150000
* 00160000
* 00170000
* 00180000
* 00190000
* 00200000
* 00210000
* 00220000
* 00230000
* 00240000
* 00250000
* 00260000
* 00270000
* 00280000
* 00290000
* 00300000
* 00310000
* 00320000
* 00330000
* 00340000
* 00350000
* 00360000
* 00370000
* 00380000
* 00390000
* 00400000
* 00410000
* 00420000
* 00430000
* 00440000
* 00450000
* 00460000
* 00470000
* 00480000
* 00490000
* 00500000
* 00510000
* 00520000
* 00530000
* 00540000
* 00550000
* 00560000
* 00570000
* 00580000
* 00590000
* 00600000
* 00610000
* 00620000
* 00630000
* 00640000
* 00650000
* 00660000

```

```

* EXIT-ERROR =
* RETURN CODE = NONE
* ABEND CODES = NONE
* ERROR-MESSAGES =
* DSN8064E - INVALID DL/I STC-CODE ON GU MSG
* DSN8065E - INVALID DL/I STC-CODE ON ISRT MSG
* EXTERNAL REFERENCES =
* ROUTINES/SERVICES = MODULE DSN8IP7
* MODULE PLITDLI
* MODULE DSN8MPG
* DATA-AREAS =
* DSN8MPCA - PARAMETER TO BE PASSED TO DSN8CP7
* CONTAINS TERMINAL INPUT AND
* OUTPUT AREAS.
* IN_MESSAGE - MFS INPUT
* OUT_MESSAGE - MFS OUTPUT
* CONTROL-BLOCKS = NONE
* TABLES = NONE
* CHANGE-ACTIVITY = NONE
* *PSEUDOCODE*
* PROCEDURE
* DECLARATIONS.
* ALLOCATE PL/I WORK AREA FOR COMMAREA.
* INITIALIZATION.
* PUT MODULE NAME 'DSN8IP6' IN AREA USED BY ERROR-HANDLER
* PUT MODNAME 'DSN8IPFO' IN MODNAME FIELD.
* STEP1.
* CALL DLI GU INPUT MESSAGE.
* IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE AND
* STOP PROGRAM.
* IF SCREEN CLEARED/UNFORMATTED , MOVE '00' TO PFKIN.
* MOVE INPUT MESSAGE FIELDS TO CORRESPONDING
* INAREA FIELDS IN COMPARM.
* CALL DSN8IP7 (COMMAREA)
* MOVE OUTAREA FIELDS IN PCONVSTA TO CORRESPONDING
* OUTPUT MESSAGE FIELDS.
* IF LASTSCR 'DSN8001' MOVE 'DSN8IPFO' TO MODNAME FIELD
* ELSE MOVE 'DSN8IPEO' TO MODNAME FIELD.
* CALL DLI ISRT OUTPUT MESSAGE.
* IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE AND
* STOP PROGRAM.
* END.
* -----
/* **** FIELDS SENT TO MESSAGE ROUTINE */
DCL MODULE CHAR (07) INIT ('DSN8IP6');
DCL OUTMSG CHAR (69);
1/* DECLARATION FOR INPUT: MIDNAME DSN8IPFI/DSN8IPEI */01310000
0DCL 1 IN_MESSAGE STATIC,
 2 LL BIN FIXED (31),
 2 Z1 CHAR (1),
 2 Z2 CHAR (1),
 2 TC_CODE CHAR (7),
 2 MESSAGE,
 3 INPUT,
 5 MAJSYS CHAR (1),
 5 ACTION CHAR (1),
 5 OBJFLD CHAR (2),
 5 SEARCH CHAR (2),
 5 PFKIN CHAR (2),
 5 DATA CHAR (60),
 5 TRANDATA(15) CHAR (40);
-/*01480000

```

```

/* DECLARATION FOR OUTPUT: MODNAME DSN8IPFO/DSN8IPEO */01490000
/*******/01500000
ODCL 1 OUT_MESSAGE STATIC,
 2 LL BIN FIXED (31) INIT (1613),
 2 ZZ BIN FIXED (15) INIT (0),
 2 OUTPUT,
 3 OUTPUTAREA,
 5 MAJSYS CHAR (1),
 5 ACTION CHAR (1),
 5 OBJFLD CHAR (2),
 5 SEARCH CHAR (2),
 5 DATA CHAR (60),
 5 TITLE CHAR (50),
 5 DESC2 CHAR (50),
 5 DESC3 CHAR (50),
 5 DESC4 CHAR (50),
 5 MSG CHAR (79),
 5 PFKTEXT CHAR (79),
 5 OUTPUT,
 7 LINE (15) CHAR (79);
/*******/01690000
/*
 * DECLARATION FOR PASSING INPUT/OUTPUT DATA BETWEEN THE */01700000
/*
 * SUBSYSTEM DEPENDENT MODULE IMS/DL1 AND SQL1 AND SQL2 */01710000
/*******/01720000
 EXEC SQL INCLUDE DSN8MPCA; 01730000
1/*******/01740000
/*
 * DECLARATION FOR PGM-LOGIC */01750000
/*******/01760000
ODCL ONE BIN FIXED (31) INIT (1) STATIC; 01770000
DCL THREE BIN FIXED (31) INIT (3) STATIC; 01780000
DCL FOUR BIN FIXED (31) INIT (4) STATIC; 01790000
ODCL GU_FKT CHAR (4) INIT ('GU ') STATIC; 01800000
DCL ISRT_FKT CHAR (4) INIT ('ISRT') STATIC; 01810000
DCL CHNG_FKT CHAR (4) INIT ('CHNG') STATIC; 01820000
DCL ROLL_FKT CHAR (4) INIT ('ROLL') STATIC; 01830000
ODCL MODNAME CHAR (8) STATIC; 01840000
ODCL (ADDR,LOW) BUILTIN; 01850000
ODCL PLITDLI EXTERNAL ENTRY; 01860000
DCL DSN8IP7 EXTERNAL ENTRY; 01870000
ODCL (IOPCB_ADDR,ALTPCB_ADDR) POINTER; 01880000
ODCL DSN8MPG EXTERNAL ENTRY; 01890000
1/*******/01900000
/*
 * DECLARATION FOR IO / ALTPCB MASK */01910000
/*******/01920000
ODCL 1 IOPCB BASED (IOPCB_ADDR), 01930000
 2 IOLTERM CHAR (8), 01940000
 2 FILLER CHAR (2), 01950000
 2 STC_CODE CHAR (2), 01960000
 2 CDATE CHAR (4), 01970000
 2 CTIME CHAR (4), 01980000
 2 SEQNUM CHAR (4), 01990000
 2 MOD_NAME CHAR (8), 02000000
 2 USERID CHAR (8); 02010000
ODCL 1 ALTPCB BASED (ALTPCB_ADDR), 02020000
 2 ALTLTERM CHAR (8), 02030000
 2 FILLER CHAR (2), 02040000
 2 STC_CODE CHAR (2); 02050000
 02060000
/*******/02070000
/*
 * ALLOCATE COBOL WORK AREA /INITIALIZATIONS */02080000
/*******/02090000
 02100000
0 ALLOCATE COMMAREA SET(COMMPTR); 02110000
 COMMAREA = ''; /* CLEAR COMMON AREA*/ 02120000
 IN_MESSAGE = ''; /* CLEAR INPUT FIELD*/ 02130000
 MODNAME = 'DSN8IPFO'; /* GET MODULE NAME */ 02140000
 DSN8_MODULE_NAME.MAJOR = 'DSN8IP6'; /* GET MODULE NAME */ 02150000
 OUTAREA.MAJSYS = 'P'; /* MAJOR SYSTEM - P */ 02160000
 02170000
/*******/02180000
/*
 * CALL DL1 GU INPUT MESSAGE */02190000
/*
 * PRINT ERROR MESSAGE IF STATUS CODE NOT OK */02200000
/*******/02210000
 02220000
0 CALL PLITDLI (THREE,GU_FKT,IOPCB,IN_MESSAGE); /* CALL DL1 GU */ 02230000
 02240000
0 IF IOPCB.STC_CODE ^= ' ' THEN /* ERROR? */ 02250000
 DO;
 CALL DSN8MPG (MODULE, '064E', OUTMSG); /* PRINT MESSAGE */ 02270000
 OUTPUTAREA.MSG = OUTMSG|| IOPCB.STC_CODE; 02280000
 02290000
 02300000

```

```

 GO TO CSEND; /*CALL DL1 ISRT OUTPUT MESSAGE */ 02310000
 END; 02320000
 02330000
 02340000
/****** 02350000
/* CLEARED AND UNFORMATTED SCREEN? */ 02360000
/****** 02370000
 02380000
0 IF Z2 = LOW(1) THEN COMPARM.PFKIN = '00'; 02390000
0 PCONVSTA.CONVID = IOPCB.IOLTERM||USERID; 02400000
0 INAREA = INPUT, BY NAME; /*MOVE INPUT MESSAGE */ 02410000
0 INAREA.MAJSYS = 'P'; /*FIELDS TO INAREA FIELDS*/ 02420000
0 CALL DSN8IP7 (COMMPTR); 02430000
 02440000
 02450000
0 OUTPUTAREA = OUTAREA , BY NAME; /*TO OUTPUT MESSAGE FIELDS*/ 02460000
0 02470000
0 02480000
0 IF LASTSCR = 'DSN8002' THEN MODNAME = 'DSN8IPE0'; 02490000
0 ELSE MODNAME = 'DSN8IPF0'; 02500000
0 02510000
/****** 02520000
/* CALL DL ISRT OUTPUT MESSAGE */ 02530000
/* PRINT ERROR MESSAGE IF STATUS CODE NOT OK */ 02540000
/****** 02550000
 02560000
CSEND: 02570000
 02580000
 02590000
 CALL PLITDLI (FOUR,ISRT_FKT,IOPCB,OUT_MESSAGE,MODNAME); 02600000
 02610000
 02620000
0 IF IOPCB.STC_CODE = ' ' THEN GO TO CEND; /* STATUS CODE OK*/ 02630000
0 02640000
 /*PRINT ERROR MESSAGE*/ 02650000
 CALL DSN8MPG (MODULE, '065E', OUTMSG); 02660000
 OUTPUTAREA.MSG = OUTMSG||IOPCB.STC_CODE; 02670000
 02680000
0 CALL PLITDLI (THREE,CHNG_FKT,ALTPCB,IOLTERM); /* CALL DL1 CHNG */ 02690000
0 02700000
0 IF ALTPCB.STC_CODE ^= ' ' THEN GO TO CSEND1; /* ERROR? */ 02710000
0 02720000
 /* CALL DL1 ISRT */ 02730000
0 CALL PLITDLI (FOUR,ISRT_FKT,ALTPCB,OUT_MESSAGE,MODNAME); 02740000
 02750000
0CSEND1: /* PERFORM ROLLBACK */ 02760000
 CALL PLITDLI (ONE,ROLL_FKT); 02770000
 02780000
0CEND: /* RETURN */ 02790000
 END DSN8IP6; 02800000

```

## Referencia relacionada

[“Ejemplos de aplicaciones en IMS” en la página 1415](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

## DSN8IP7

ESTE MÓDULO REALIZA LOS INCLUIDOS PARA INTRODUCIR LAS ESTRUCTURAS DE TABLA SQL DCLS Y DCLGEN, ASÍ COMO EL ÁREA DE PARÁMETROS.

```

DSN8IP7:PROC (COMMPTR) ;
/****** *
* MODULE NAME = DSN8IP7 *
* DESCRIPTIVE NAME = SAMPLE APPLICATION *
* SQL 1 MAINLINE *
* IMS *
* PL/I *
* COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1985 *
* REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083 *
* STATUS = RELEASE 2, LEVEL 0 *
* FUNCTION = THIS MODULE PERFORMS THE INCLUDES TO BRING IN THE *
* SQL TABLE DCLS AND DCLGEN STRUCTURES AS WELL AS *
* THE PARAMETER AREA. *

```

```

* INCLUDE DSN8MP1. *
* CALL DSN8IP8. *
* RETURN TO DSN8IP6. *
*
* NOTES = NONE *
*
* MODULE TYPE = PL/I PROC(COMMPPTR). *
* PROCESSOR = DB2 PRECOMPILER, PL/I OPTIMIZER *
* MODULE SIZE = SEE LINKEDIT *
* ATTRIBUTES = REUSABLE *
*
* ENTRY POINT = DSN8IP7. *
* PURPOSE = SEE FUNCTION *
* LINKAGE = CALLED BY DSN8IP6 *
*
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION: *
*
* SYMBOLIC LABEL/NAME = COMMPPTR *
* DESCRIPTION = POINTER TO COMMAREA *
*
* COMMON AREA. *
*
* SYMBOLIC LABEL/NAME = PFKIN *
* DESCRIPTION = 00/01/02/03/07/08/10 *
*
* SYMBOLIC LABEL/NAME = INAREA *
* DESCRIPTION = USER INPUT *
*
* OUTPUT = PARAMETERS EXPLICITLY RETURNED: *
* COMMON AREA. *
*
* SYMBOLIC LABEL/NAME = OUTAREA *
* DESCRIPTION =GENERAL MENU OR *
* SECONDARY SELECTION MENU *
*
* SYMBOLIC LABEL/NAME = LASTSCR *
* DESCRIPTION = DSN8001/DSN8002 *
*
*
* EXIT-NORMAL = DSN8IP6. *
* EXIT-ERROR = DSN8IP6. *
*
* RETURN CODE = NONE. *
*
* ABEND CODES = NONE. *
*
* ERROR-MESSAGES = NONE. *
*
* EXTERNAL REFERENCES = *
* ROUTINES/SERVICES = NONE. *
*
* DATA-AREAS = *
* DSN8MPCA - PLI STRUCTURE FOR COMMAREA *
* DSN8MPCS - VCONA TABLE DCL AND PCONA DCLGEN *
* DSN8MPOV - VOPTVAL TABLE DCL & POPTVAL DCLGEN *
* DSN8MPVO - VALIDATION CURSORS *
* DSN8MP1 - SQL1 COMMON MODULE FOR IMS AND CICS *
* DSN8MP3 -- DSN8MP5 - VALIDATION MODULES CALLED BY DSN8MP1 *
* DSN8MPXX - SQL ERROR HANDLER *
*
* CONTROL-BLOCKS = *
* SQLCA - SQL COMMUNICATION AREA *
*
* TABLES = NONE. *
*
* CHANGE-ACTIVITY = NONE. *
*
* *PSEUDOCODE*
* PROCEDURE
* INCLUDE DECLARATIONS.
* INCLUDE DSN8MP1.
* INCLUDE ERROR HANDLER.
*
* CP1EXIT: (REFERENCED BY DSN8MP1)
* RETURN.
*
* CP1CALL: (REFERENCED BY DSN8MP1)
* CALL 'DSN8IP8'(COMMPPTR).
* GO TO MP1SAVE. (LABEL IN DSN8MP1)

```

```

* INCLUDE VALIDATION MODULES. *
*
* END. *
*****SQL1 MAINLINE***** */
/* SQL RETURN CODE HANDLING*/
EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR;
EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;

/*FIELDS SENT TO MESSAGE ROUTINE */
/*****
DCL MODULE CHAR (07) INIT ('DSN8IP7');
DCL OUTMSG CHAR (69);

DCL STRING BUILTIN;
DCL J FIXED BIN;
DCL SAVE_CONVID CHAR(16);
DCL (SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1);
DCL DSN8IP8 EXTERNAL ENTRY;
DCL DSN8MPG EXTERNAL ENTRY;
EXEC SQL INCLUDE DSN8MPCA; /* INCLUDE COMMAREA */
DSN8_MODULE_NAME.MAJOR = 'DSN8IP7 ' ; /* INITIALIZE MODULE NAME*/
EXEC SQL INCLUDE DSN8MPCS; /* INCLUDE PCONA */
EXEC SQL INCLUDE DSN8MPOV; /* INCLUDE POPTVAL */
EXEC SQL INCLUDE DSN8MPVO; /* INCLUDE CURSOR */
EXEC SQL INCLUDE SQLCA; /* INCLUDE SQL COMMAREA*/
EXEC SQL INCLUDE DSN8MP1; /* INCLUDE SQL1 MAIN*/
EXEC SQL INCLUDE DSN8MPXX; /* INCLUDE ERRORHANDLER */

CP1EXIT :
 RETURN; /* EXIT */

CP1CALL :
 CALL DSN8IP8 (COMM PTR); /* GO TO DSN8IP8 (SQL2) */
 GO TO MP1SAVE;

EXEC SQL INCLUDE DSN8MP3; /* INCLUDE ACTION VALIDATION*/
EXEC SQL INCLUDE DSN8MP4; /* INCLUDE OBJECT VALIDATION*/
EXEC SQL INCLUDE DSN8MP5; /* INCLUDE SEARCH CRITERIA*/
END; /* VALIDATION */

```

### Referencia relacionada

[“Ejemplos de aplicaciones en IMS” en la página 1415](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

## DSN8IP8

ROUTER PARA SELECCIÓN SECUNDARIA Y/O PROCESAMIENTO DETALLADO LLAMA A MÓDULOS DE SELECCIÓN SECUNDARIA DSN8MPM LLAMA A MÓDULOS DETALLADOS DSN8MPT DSN8MPV DSN8MPW DSN8MPX DSN8MPZ LLAMADA POR DSN8IP7 (SQL1).

```

DSN8IP8: PROC(COMMPTR) ;
%PAGE; 00010000
 00020000
/***** 00030000
* * 00040000
* MODULE NAME = DSN8IP8 * 00050000
* * 00060000
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION * 00070000
* SQL 2 COMMON MODULE * 00080000
* IMS * 00090000
* PL/I * 00100000
* PROJECT * 00110000
* * 00120000
* LICENSED MATERIALS - PROPERTY OF IBM * 00130000
* 5695-DB2 * 00136000
* (C) COPYRIGHT 1982, 1995 IBM CORP. ALL RIGHTS RESERVED. * 00143000
* * 00150000
* STATUS = VERSION 4 * 00160000
* * 00170000
* FUNCTION = ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSING* 00180000
* CALLS SECONDARY SELECTION MODULES * 00190000

```

```

* DSN8MPM * 00200000
* CALLS DETAIL MODULES * 00210000
* DSN8MPT DSN8MPV DSN8MPW DSN8MPX DSN8MPZ * 00220000
* CALLED BY DSN8IP7 (SQL1) * 00230000
* * 00240000
* NOTES = NONE * 00250000
* * 00260000
* MODULE TYPE = BLOCK OF PL/I CODE * 00270000
* PROCESSOR = DB2 PRECOMPILER, PL/I OPTIMIZER * 00280000
* MODULE SIZE = SEE LINKEDIT * 00290000
* ATTRIBUTES = REUSABLE * 00300000
* * 00310000
* ENTRY POINT = DSN8IP8 * 00320000
* PURPOSE = SEE FUNCTION * 00330000
* LINKAGE = NONE * 00340000
* INPUT = * 00350000
* SYMBOLIC LABEL/NAME = COMMPTR * 00360000
* DESCRIPTION = POINTER TO COMMAREA * 00370000
* * 00380000
* OUTPUT = * 00390000
* SYMBOLIC LABEL/NAME = COMMPTR * 00400000
* DESCRIPTION = POINTER TO COMMAREA * 00410000
* * 00420000
* EXIT-NORMAL = * 00430000
* * 00440000
* EXIT-ERROR = IF SQLERROR OR SQLWARNING, SQL WHENEVER CONDITION * 00450000
* SPECIFIED IN DSN8IP8 WILL BE RAISED AND PROGRAM * 00460000
* WILL GO TO THE LABEL DB_ERROR. * 00470000
* * 00480000
* * 00490000
* RETURN CODE = NONE * 00500000
* * 00510000
* ABEND CODES = NONE * 00520000
* * 00530000
* ERROR-MESSAGES = * 00540000
* DSN8062E-AN OBJECT WAS NOT SELECTED * 00550000
* DSN8066E-UNSUPPORTED PFK OR LOGIC ERROR * 00570000
* DSN8072E-INVALID SELECTION ON SECONDARY SCREEN * 00580000
* * 00590000
* EXTERNAL REFERENCES = NONE * 00600000
* ROUTINES/SERVICES = 8 MODULES LISTED ABOVE * 00610000
* DSN8MPG - ERROR MESSAGE ROUTINE * 00620000
* * 00630000
* DATA-AREAS = * 00640000
* DSN8MPAC - DCLGEN FOR ACTIVITY TYPES * 00650000
* DSN8MPAS - CURSOR SECONDARY SELECTION FOR * 00660000
* STAFF
* DSN8MPCA - COMMUNICATION AREA BETWEEN MODULES * 00680000
* DSN8MPDH - CURSOR FOR DISPLAY TEXT FROM * 00690000
* TDSPTXT TABLE
* DSN8MPDP - DCLGEN FOR DEPARTMENT * 00710000
* DSN8MPDT - DCLGEN FOR DISPLAY TEXT TABLE * 00720000
* DSN8MPEM - DCLGEN FOR EMPLOYEE * 00730000
* DSN8MPEP - DCLGEN FOR PROJECT/STAFFING * 00740000
* DSN8MPES - CURSOR SECONDARY SELECTION FOR * 00750000
* ESTIMATES
* DSN8MPOV - DCLGEN FOR OPTION VALIDATION * 00770000
* DSN8MPPA - DCLGEN FOR PROJECT/ACTIVITIES * 00780000
* DSN8MPPD - DCLGEN FOR PROJ STRUCTURE DETAIL * 00790000
* DSN8MPP2 - DCLGEN FOR PROJ STRUCTURE DETAIL * 00795000
* DSN8MPPE - CURSOR PROJECT DETAIL * 00800000
* DSN8MPPJ - DCLGEN FOR PROJECTS * 00810000
* DSN8MPPL - CURSOR PROJECT LIST * 00820000
* DSN8MPPR - DCLGEN FOR PROJ/RESP EMPLOYEE * 00830000
* DSN8MPSA - DCLGEN FOR PROJ ACTIVITY LISTING * 00850000
* DSN8MPFP - DCLGEN FOR PROJECT-EMPLOYEE * 00855000
* DSN8MPED - DCLGEN FOR EMPLOYEE-DEPT * 00857000
* DSN8MPSL - CURSOR STAFFING LIST * 00860000
* DSN8MPS2 - DCLGEN FOR PROJ ACTIVITY LISTING * 00870000
* DSN8MPM - SECONDARY SELECTION FOR PROJECTS * 00880000
* DSN8MPT - PROJECT ACTIVITY LIST * 00890000
* DSN8MPV - PROJECT STRUCTURE DETAIL * 00900000
* DSN8MPW - ACTIVITY STAFFING DETAIL * 00910000
* DSN8MPX - ACTIVITY ESTIMATE DETAIL * 00920000
* DSN8MPZ - PROJECT DETAIL * 00930000
* * 00940000
* CONTROL-BLOCKS = * 00950000
* SQLCA - SQL COMMUNICATION AREA * 00960000
* * 00970000
* TABLES = NONE * 00980000
* * 00990000
* CHANGE-ACTIVITY = NONE * 01000000

```

```

*
*
* *PSEUDOCODE*
*
* THIS MODULE DETERMINES WHICH SECONDARY SELECTION AND/OR
* DETAIL MODULE(S) ARE TO BE CALLED IN THE IMS/PL/I
* ENVIRONMENT.
*
* WHAT HAS HAPPENED SO FAR?.....THE SUBSYSTEM
* DEPENDENT MODULE (IMS,CICS,TSO) OR (SQL 0) HAS
* READ THE INPUT SCREEN, FORMATTED THE INPUT AND PASSED CONTROL
* TO SQL 1. SQL 1 PERFORMS VALIDATION ON THE SYSTEM DEPENDENT
* FIELDS (MAJOR SYSTEM, ACTION, OBJECT, SEARCH CRITERIA). IF
* ALL SYSTEM FIELDS ARE VALID SQL 1 PASSED CONTROL TO THIS
* MODULE. PASSED PARAMETERS CONSIST ONLY OF A POINTER WHICH
* POINTS TO A COMMUNICATION CONTROL AREA USED TO COMMUNICATE
* BETWEEN SQL 0 , SQL 1, SQL 2 AND THE SECONDARY SELECTION
* AND DETAIL MODULES.
*
* WHAT IS 'INCLUDED' IN THIS MODULE?.....
* ALL SECONDARY SELECTION AND DETAIL MODULES ARE 'INCLUDED'.
* ALL VARIABLES KNOWN IN THIS PROCEDURE ARE KNOWN IN THE
* SUB PROCEDURES. ALL SQL CURSOR DEFINITIONS AND
* SQL 'INCLUDES' ARE DONE IN THIS PROCEDURE. BECAUSE OF THE
* RESTRICTION THAT CURSOR HOST VARIABLES MUST BE DECLARED BEFORE
* THE CURSOR DEFINITION ALL CURSOR HOST VARIABLES ARE DECLARED
* IN THIS PROCEDURE.
*
* PROCEDURE
* IF ANSWER TO DETAIL SCREEN & DETAIL PROCESSOR
* IS NOT WILLING TO ACCEPT AN ANSWER THEN
* NEW REQUEST*
*
* ELSE
* IF ANSWER TO A SECONDARY SELECTION THEN
* DETERMINE IF NEW REQUEST.
*
* CASE (NEW REQUEST)
*
* SUBCASE ('ADD')
* DETAIL PROCESSOR
* RETURN TO SQL 1
* ENDSUB
*
* SUBCASE ('DISPLAY', 'ERASE', 'UPDATE')
* CALL SECONDARY SELECTION
* IF # OF POSSIBLE CHOICES IS ^= 1 THEN
* RETURN TO SQL 1
* ELSE
* CALL THE DETAIL PROCESSOR
* RETURN TO SQL 1
* ENDSUB
*
* ENDCASE
*
* IF ANSWER TO SECONDARY SELECTION AND A SELECTION HAS
* ACTUALLY BEEN MADE THEN
* VALID SELECTION #?
* IF IT IS VALID THEN
* CALL DETAIL PROCESSOR
* RETURN TO SQL 1
* ELSE
* PRINT ERROR MSG
* RETURN TO SQL 1.
*
* IF ANSWER TO SECONDARY SELECTION THEN
* CALL SECONDARY SELECTION
* RETURN TO SQL 1.
*
* IF ANSWER TO DETAIL THEN
* CALL DETAIL PROCESSOR
* RETURN TO SQL 1.
*
* END.
*
* *EXAMPLE- A ROW IS SUCCESSFULLY ADDED, THE OPERATOR RECEIVES*
* THE SUCCESSFULLY ADDED MESSAGE AND JUST HITS ENTER.
*
* END.
*
-----/
/* INCLUDE DECLares */
EXEC SQL INCLUDE DSN8MPCA; /*COMMUNICATION AREA BETWEEN MODULES */

```

```

EXEC SQL INCLUDE SQLCA; /*SQL COMMUNICATION AREA */ 01830000
 /* PROJECTS */ 01840000
EXEC SQL INCLUDE DSN8MPDP; /* DCLGEN FOR DEPARTMENT */ 01850000
EXEC SQL INCLUDE DSN8MPEM; /* DCLGEN FOR EMPLOYEE */ 01860000
EXEC SQL INCLUDE DSN8MPPJ; /* DCLGEN FOR PROJECTS */ 01870000
EXEC SQL INCLUDE DSN8MPAC; /* DCLGEN FOR ACTIVITY TYPES */ 01880000
EXEC SQL INCLUDE DSN8MPPA; /* DCLGEN FOR PROJECT/ACTIVITIES */ 01890000
EXEC SQL INCLUDE DSN8MPEP; /* DCLGEN FOR PROJECT/STAFFING */ 01900000
EXEC SQL INCLUDE DSN8MPPR; /* DCLGEN FOR PROJ/RESP EMPLOYEE */ 01910000
EXEC SQL INCLUDE DSN8MPPD; /* DCLGEN FOR PROJ STRUCTURE DETAIL*/ 01920000
EXEC SQL INCLUDE DSN8MPP2; /* DCLGEN FOR PROJ STRUCTURE DETAIL*/ 01930000
EXEC SQL INCLUDE DSN8MPSA; /* DCLGEN FOR PROJ ACTIVITY LISTING */ 01940000
EXEC SQL INCLUDE DSN8MPS2; /* DCLGEN FOR PROJ ACTIVITY LISTING */ 01950000
EXEC SQL INCLUDE DSN8MPFP; /* DCLGEN FOR PROJECT-EMPLOYEE */ 01955000
EXEC SQL INCLUDE DSN8MPED; /* DCLGEN FOR EMPLOYEE-DEPT */ 01957000
 /* PROGRAMMING TABLES */ 01960000
EXEC SQL INCLUDE DSN8MPOV; /* DCLGEN FOR OPTION VALIDATION */ 01970000
EXEC SQL INCLUDE DSN8MPDT; /* DCLGEN FOR DISPLAY TEXT TABLE */ 01980000
 01990000
 02000000
/* CURSORS */
EXEC SQL INCLUDE DSN8MPPL; /* MAJSYS P - SEC SEL FOR PS, AL, PR */ 02010000
EXEC SQL INCLUDE DSN8MPES; /* MAJSYS P - SEC SEL FOR AE */ 02020000
EXEC SQL INCLUDE DSN8MPAS; /* MAJSYS P - SEC SEL FOR AS */ 02030000
EXEC SQL INCLUDE DSN8MPPE; /* MAJSYS P - DETAIL FOR PS */ 02040000
EXEC SQL INCLUDE DSN8MPSL; /* MAJSYS P - DETAIL FOR AL */ 02050000
EXEC SQL INCLUDE DSN8MPDH; /* PROG TABLES - DISPLAY HEADINGS */ 02060000
 02070000
DCL LENGTH BUILTIN; 02080000

 /* FIELDS SENT TO MESSAGE ROUTINE */ 02090000

 02100000
 02110000
 02120000
DCL MODULE CHAR (07) INIT ('DSN8IP8'); 02130000
DCL OUTMSG CHAR (69); 02140000
 02150000
DCL DSN8MPG EXTERNAL ENTRY; 02160000
 02170000

 /* SQL RETURN CODE HANDLING */ 02180000

 02190000
 02200000
 02210000

 EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR;
 EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;
 02220000
 02230000
 02240000
0 DCL UNSPEC BUILTIN; 02250000
DCL VERIFY BUILTIN; 02260000
 02270000

 /* INITIALIZATIONS */ 02280000

 02290000
 02300000
 02310000

 DSN8_MODULE_NAME.MAJOR='DSN8IP8';
 DSN8_MODULE_NAME.MINOR=' ';
 02320000
 02330000
 02340000

 /* DETERMINES WHETHER NEW REQUEST OR NOT */ 02350000

 02360000
 02370000
 02380000

 /* IF 'NO ANSWER POSSIBLE' SET BY DETAIL PROCESSOR THEN FORCE A */
 /* NEW REQUEST. */ 02390000

 02400000
 02410000

 IF PCONVSTA.PREV = ' ' THEN COMPARM.NEWREQ = 'Y';
 02420000
 02430000

 /* IF ANSWER TO SECONDARY SELECTION THEN DETERMINE IF REALLY A */
 /* NEW REQUEST. IT WILL BE CONSIDERED A NEW REQUEST IF POSITIONS*/
 /* 3 TO 60 ARE NOT ALL BLANK AND THE ENTERED DATA IF NOT 'NEXT' */ 02440000
 02450000
 02460000
 02470000

 IF COMPARM.NEWREQ = 'N' & PCONVSTA.PREV = 'S' &
 SUBSTR(COMPARM.DATA,3,58) ^= ' ' &
 COMPARM.DATA ^= 'NEXT'
 THEN COMPARM.NEWREQ = 'Y';
 02480000
 02490000
 02500000
 02510000
 02520000

 /* IF NEW REQUEST AND ACTION IS 'ADD' THEN */
 /* CALL DETAIL PROCESSOR */ 02530000
 /* ELSE CALL SECONDARY SELECTION */ 02540000
 02550000
 02560000

 02570000
 02580000

 IF COMPARM.NEWREQ='Y' THEN
 DO;
 IF COMPARM.ACTION = 'A' THEN
 DO;
 02590000
 02600000
 02610000
 02620000

```

```

CALL DETAIL; /* CALL DETAIL PROCESSOR */ 02630000
GO TO EXIT; /* RETURN */ 02640000
END; 02650000
02660000

CALL SECSEL; /* CALL SECONDARY SELECTION */ 02670000
02680000

IF MAXSEL = 1 THEN /* IF NO. OF CHOICES = 1 */ 02690000
CALL DETAIL; /* CALL DETAIL PROCESSOR */ 02700000
GO TO EXIT; /* RETURN */ 02710000
END; 02720000
02730000

/* IF ANSWER TO SECONDARY SELECTION AND NOT A SCROLLING REQUEST */ 02740000
/* (INPUT NOT EQUAL TO 'NEXT') AND THE POSITIONS */ 02750000
/* 1 TO 2 IN INPUT DATA FIELD NOT EQUAL TO POSITIONS 1 TO 2 */ 02760000
/* IN OUTPUT DATA FIELD THEN SEE IF VALID SELECTION. */ 02770000
02780000

/***************************** 02790000
/* DETERMINES IF VALID SELECTION NUMBER */ 02800000
/***************************** 02810000
02820000

IF PCONVSTA.PREV ^= 'S' THEN GO TO IP201; /* TO SECONDARY SEL */ 02830000
02840000

IF PCONVSTA.MAXSEL < 1 THEN GO TO IP201; /* NO VALID CHOICES */ 02850000
02860000

IF COMPARM.DATA = 'NEXT' THEN GO TO IP201; /* SCROLL REQUEST */ 02870000
02880000

IF SUBSTR(COMPARM.DATA,1,2) = SUBSTR(PCONVSTA.DATA,1,2)
 THEN GO TO IP201; /* NO CHANGE ON INPUT SCREEN */ 02890000
02900000
02910000

IF SUBSTR(COMPARM.DATA,2,1) = ' ' THEN /* SECOND CHAR BLANK */ 02920000
02930000

IF VERIFY(SUBSTR(COMPARM.DATA,1,1), '123456789') = 0 THEN 02940000
DO;
 SUBSTR(COMPARM.DATA,2,1) = SUBSTR(COMPARM.DATA,1,1); 02950000
 SUBSTR(COMPARM.DATA,1,1) = '0'; 02960000
END; 02970000
02980000
02990000

IF VERIFY(SUBSTR(COMPARM.DATA,1,2), '0123456789') = 0 &
 SUBSTR(COMPARM.DATA,1,2) > '00' THEN 03000000
03010000
03020000

IF DATAP <= PCONVSTA.MAXSEL THEN 03030000
DO;
 COMPARM.NEWREQ = 'Y'; /* TELL DETAIL PROCESSOR NEW REQ */ 03040000
 CALL DETAIL; /* CALL DETAIL PROCESSOR */ 03050000
 GO TO EXIT; /* RETURN */ 03060000
END; 03070000
03080000
03090000

/* INVALID SECONDARY */ 03100000
/* SELECTION */ 03110000
/* PRINT ERROR MESSAGE */ 03120000
CALL DSN8MPG (MODULE, '072E', OUTMSG); 03130000
PCONVSTA.MSG = OUTMSG;
GO TO EXIT; /* RETURN */ 03140000
03150000
03160000

/***************************** 03170000
/* DETERMINES WHETHER SECONDARY SELECTION OR DETAIL */ 03180000
/***************************** 03190000
03200000

/* MUST BE ANY ANSWER TO EITHER SEC SEL OR DETAIL */ 03210000
IP201:
IF PCONVSTA.PREV = 'S' THEN 03220000
DO;
 CALL SECSEL; /* CALL SECONDARY SELECTION*/ 03230000
 GO TO EXIT; /* RETURN */ 03240000
END; 03250000
03260000
03270000
03280000

IF PCONVSTA.PREV = 'D' THEN 03290000
DO;
 CALL DETAIL; /* CALL DETAIL PROCESSOR */ 03300000
 GO TO EXIT; /* RETURN */ 03310000
END; 03320000
03330000
03340000

/* LOGIC ERROR */ 03350000
/* PRINT ERROR MESSAGE */ 03360000
CALL DSN8MPG (MODULE, '066E', OUTMSG); 03370000
PCONVSTA.MSG= OUTMSG;
GO TO EXIT; /* RETURN */ 03380000
03390000
03400000

EXEC SQL INCLUDE DSN8MPXX; /* HANDLES SQL ERRORS */ 03410000
GO TO EXIT; /* RETURN */ 03420000
03430000
03440000

```

```

/* CALLS SECONDARY SELECTION AND RETURNS TO SQL 1 */ 03450000
/* NOTE - SAME SECONDARY SELECTION MODULE FOR DS, DE AND EM */ 03460000
/***/ 03470000
 03480000
SECSEL: PROC; /* CALL APPROPRIATE SECONDARY SELECTION MODULE */ 03490000
 PCONVSTA.LASTSCR = 'DSN8001'; /* NOTE GENERAL SCREEN */ 03500000
 IF COMPARM.OBJFLD='AE' | /*ACTIVITY ESTIMATE */ 03510002
 COMPARM.OBJFLD='AL' | /*PROJECT ACTIVITY LISTING */ 03520002
 COMPARM.OBJFLD='AS' | /*INDIVIDUAL PROJECT STAFFING */ 03530002
 COMPARM.OBJFLD='PR' | /*INDIVIDUAL PROJECT PROCESSING*/ 03540002
 COMPARM.OBJFLD='PS' THEN /*PROJECT STRUCTURE */ 03550002
 DO;
 CALL DSN8MPM; /*SECONDARY SELECTION FOR PROJECTS*/ 03570000
 RETURN; /*RETURN */ 03580000
 END;
 /* SECONDARY SELECTION MISSING */ 03600000
 /* PRINT ERROR MESSAGE */ 03610000
 CALL DSN8MPG (MODULE, '062E', OUTMSG); 03620000
 PCONVSTA.MSG= OUTMSG;
 GO TO EXIT;
END SECSEL;
/***/ 03650000
/* CALLS DETAIL PROCESSOR AND RETURNS TO SQL 1 */ 03660000
/***/ 03670000
 03680000
 03690000
DETAIL: PROC; /* CALL APPROPRIATE DETAIL MODULE */ 03700000
 PCONVSTA.LASTSCR = 'DSN8002'; /* SET FOR DETAIL MAP */ 03710000
 03720000
 IF COMPARM.OBJFLD='PS' THEN 03730002
 DO;
 CALL DSN8MPV; /* PROJECT STRUCTURE DETAIL */ 03740000
 RETURN; 03750000
 END;
 03760000
 IF COMPARM.OBJFLD='AL' THEN 03770000
 DO;
 CALL DSN8MPT; /* PROJECT ACTIVITY LIST */ 03780000
 RETURN; 03790000
 END;
 03800000
 IF COMPARM.OBJFLD='PR' THEN 03810000
 DO;
 CALL DSN8MPZ; /* PROJECT DETAIL */ 03820000
 RETURN; 03830000
 END;
 03840000
 IF COMPARM.OBJFLD='AE' THEN 03850002
 DO;
 CALL DSN8MPX; /* ACTIVITY ESTIMATE DETAIL */ 03860000
 RETURN; 03870000
 END;
 03880000
 IF COMPARM.OBJFLD='AS' THEN 03890000
 DO;
 CALL DSN8MPW; /* ACTIVITY STAFFING DETAIL */ 03900000
 RETURN; 03910000
 END;
 03920000
 CALL DSN8MPG (MODULE, '062E', OUTMSG); 03930000
 PCONVSTA.MSG= OUTMSG;
 GO TO EXIT;
END DETAIL;
 03940000
 03950000
 03960000
 03970002
 03980000
 03990000
 04000000
 04010000
 04020000
 04030000
 04040000
 04050000
 04060000
 04070000
 04080000
EXIT: RETURN; /* RETURNS TO SQL 1 */ 04090000
 04100000
 04110000
 /* PROJECTS */
 EXEC SQL INCLUDE DSN8MPM; /* SEC SEL - PROJECTS */ 04120000
 EXEC SQL INCLUDE DSN8MPV; /* DETAIL - PROJ STRUCTURE */ 04130000
 EXEC SQL INCLUDE DSN8MPT; /* DETAIL - PROJ ACT LISTING*/ 04140000
 EXEC SQL INCLUDE DSN8MPZ; /* DETAIL - INDIVIDUAL PROJ */ 04150000
 EXEC SQL INCLUDE DSN8MPX; /* DETAIL - INDIVID ACTIVITY*/ 04160000
 EXEC SQL INCLUDE DSN8MPW; /* DETAIL - INDIVID STAFFING*/ 04170000
END DSN8IP8; 04180000

```

## Referencia relacionada

[“Ejemplos de aplicaciones en IMS” en la página 1415](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

## DSN8IP3

ESTE MÓDULO LISTA LOS NÚMEROS DE TELÉFONO DE EMPLEADOS Y LOS ACTUALIZA SI LO DESEA.

```
DSN8IP3: PROC(IOPCB_ADDR,ALTPCB_ADDR) OPTIONS (MAIN);
/***/
* MODULE NAME = DSN8IP3 *
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION *
* PHONE APPLICATION *
* IMS *
* PL/I *
* COPYRIGHT = 5665-DB2 (C) COPYRIGHT IBM CORP 1982, 1991 *
* SEE COPYRIGHT INSTRUCTIONS *
* LICENSED MATERIALS - PROPERTY OF IBM *
* STATUS = VERSION 2 RELEASE 3, LEVEL 0 *
* FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND *
* UPDATES THEM IF DESIRED. *
* NOTES = *
* DEPENDENCIES = TWO MFS MAPS ARE REQUIRED: *
* DSN8IPL AND DSN8IPN *
* RESTRICTIONS = NONE *
* MODULE TYPE = PL/I PROC OPTIONS(MAIN) *
* PROCESSOR = DB2 PRECOMPLIER, PL/I OPTIMIZER *
* MODULE SIZE = SEE LINKEDIT *
* ATTRIBUTES = REENTRANT *
* ENTRY POINT = DSN8IP3 *
* PURPOSE = SEE FUNCTION *
* LINKAGE = INVOKED FROM IMS *
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION: *
* INPUT-MESSAGE: *
* SYMBOLIC LABEL/NAME = DSN8IPNO *
* DESCRIPTION = PHONE MENU 1 (SELECT) *
* SYMBOLIC LABEL/NAME = DSN8IPL0 *
* DESCRIPTION = PHONE MENU 2 (UPDATE) *
* SYMBOLIC LABEL/NAME = VPHONE *
* DESCRIPTION = VIEW OF TELEPHONE INFORMATION *
* SYMBOLIC LABEL/NAME = VEMPLP *
* DESCRIPTION = VIEW OF EMPLOYEE INFORMATION *
* OUTPUT = PARAMETERS EXPLICITLY RETURNED: *
* OUTPUT-MESSAGE: *
* SYMBOLIC LABEL/NAME = DSN8IPNO *
* DESCRIPTION = PHONE MENU 1 (SELECT) *
* SYMBOLIC LABEL/NAME = DSN8IPL0 *
* DESCRIPTION = PHONE MENU 2 (UPDATE) *
* EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION *
* EXIT-ERROR = *
* RETURN CODE = NONE *
* ABEND CODES = NONE *
* ERROR-MESSAGES = *
* DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED *
* DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE *
* DSN8008I - NO EMPLOYEE FOUND IN TABLE *
* DSN8058I - PRESS PA1 FOR NEXT PAGE / ENTER FOR *
* SELECTION MENU *
* DSN8060E - SQL ERROR, RETURN CODE IS: *
* DSN8064E - INVALID DL/I STC-CODE ON GU MSG *
* DSN8065E - INVALID DL/I STC-CODE ON ISRT MSG *
* EXTERNAL REFERENCES =
```

```

* ROUTINES/SERVICES =
* DSN8MPG - ERROR MESSAGE ROUTINE
*
* DATA-AREAS =
* IN_MESSAGE - MFS INPUT
* OUT_MESSAGE - MFS OUTPUT
*
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA
*
* TABLES = NONE
*
*
* CHANGE-ACTIVITY = NONE
*
*
* *PSEUDOCODE*
*
* PROCEDURE
* CALL DLI GU INPUT MESSAGE.
* IF STATUS CODE NOT OK THEN SEND ERROR MESSAGE ,
* PGM-STOP.
*
* CASE (ACTION)
*
* SUBCASE ('L')
* IF LASTNAME IS '*' THEN
* LIST ALL EMPLOYEES
* PREPARE OUTPUT_MESSAGE
* CALL DLI ISRT OUTPUT_MESSAGE
* ELSE
* IF LASTNAME CONTAINS '%' THEN
* LIST EMPLOYEES GENERIC
* PREPARE OUTPUT_MESSAGE
* CALL DLI ISRT OUTPUT_MESSAGE
* ELSE
* LIST EMPLOYEES SPECIFIC
* PREPARE OUTPUT_MESSAGE
* CALL DLI ISRT OUTPUT_MESSAGE
* ENDIF
* ENDIF
* ENDIF
* ENDSUB
*
* SUBCASE ('U')
* DO WHILE INPUT PHONE_NO ^= BLANK
* UPDATE PHONE_NO FOR EMPLOYEE
* END
* PREPARE OUTPUT_MESSAGE
* CALL DLI ISRT OUTPUT_MESSAGE
* OTHERWISE
* UNFORMATTED SCREEN
* PREPARE OUTPUT_MESSAGE
* CALL DLI ISRT OUTPUT_MESSAGE
* ENDSUB
*
* ENDCASE
*
* IF SQL OR DL/I ERROR HAS OCCURRED THEN
* ROLLBACK
* PGM-STOP.
* END.
*
-----/
1/****** DECLARATION FOR INPUT: MODNAME DSN8IPNI/DSN8IPLI */
/****** DECLARATION FOR OUTPUT: MODNAME DSN8IPNO/DSN8IPLO */
ODCL 1 IN_MESSAGE STATIC,
 2 LL BIN FIXED (31),
 2 ZZ CHAR (2),
 2 TC_CODE CHAR (7),
 2 ACTION CHAR (1),
 2 MESSAGE CHAR (500);
ODCL 1 INPUT_1 BASED(ADDR(MESSAGE)),
 2 LNAME CHAR (15),
 2 FNAME CHAR (12);
ODCL 1 INPUT_2 (15) BASED(ADDR(MESSAGE)),
 2 NEWNO CHAR (4),
 2 ENO CHAR (6);
/****** DECLARATION FOR OUTPUT: MODNAME DSN8IPNO/DSN8IPLO */
ODCL 1 OUT_MESSAGE STATIC,
 2 LL BIN FIXED (31),
 2 ZZ BIN FIXED (15) INIT (0),

```

```

2 ERROR CHAR (79),
2 OUTPUT CHAR (1095);
ODCL 1 OUTPUT_1 BASED(ADDR(OUTPUT)),
2 LASTNAME CHAR (15),
2 FIRSTNAME CHAR (12);
ODCL 1 OUTPUT_2 (15) BASED(ADDR(OUTPUT)),
2 FIRSTNAME CHAR (12),
2 MIDDLEINITIAL CHAR (1),
2 LASTNAME CHAR (15),
2 PHONENUMBER CHAR (4),
2 EMPLOYEENUMBER CHAR (6),
2 DEPTNUMBER CHAR (3),
2 DEPTNAME CHAR (32); /* 32 TO FIT ON ONE LINE */
DCL CHAR_SQLCODE CHAR (14);
DCL 1 CHAR_SQLSTR BASED(ADDR(CHAR_SQLCODE)),
2 CHAR_BLNK CHAR(4),
2 CHAR_SQLCOD CHAR(10);
/******
 * DECLARATION FOR IO / ALTPCB MASK
 *****/
ODCL (IOPCB_ADDR,ALTPCB_ADDR) POINTER;
ODCL 1 IOPCB BASED (IOPCB_ADDR),
2 IOLTERM CHAR (8),
2 FILLER CHAR (2),
2 STC_CODE CHAR (2);
ODCL 1 ALTPCB BASED (ALTPCB_ADDR),
2 ALTLTERM CHAR (8),
2 FILLER CHAR (2),
2 STC_CODE CHAR (2);
/******
 * DECLARATION FOR PGM-LOGIC
 *****/
ODCL ONE BIN FIXED (31) INIT (1) STATIC;
DCL THREE BIN FIXED (31) INIT (3) STATIC;
DCL FOUR BIN FIXED (31) INIT (4) STATIC;
ODCL GU_FKT CHAR (4) INIT ('GU ') STATIC;
DCL ISRT_FKT CHAR (4) INIT ('ISRT') STATIC;
DCL CHNG_FKT CHAR (4) INIT ('CHNG') STATIC;
DCL ROLL_FKT CHAR (4) INIT ('ROLL') STATIC;
ODCL MODNAME CHAR (8) STATIC;
DCL FIRST BIT (1) STATIC;
DCL EMPLOYEE_NO CHAR (6) STATIC;
DCL PHONE_NO CHAR (4) STATIC;
ODCL (I,M,ITAB) BIN FIXED(15);
ODCL (ADDR,INDEX,SUBSTR) BUILTIN;
ODCL TRANSLATE BUILTIN;
ODCL SYSPRINT EXTERNAL PRINT FILE;
ODCL PLITDLI EXTERNAL ENTRY;
ODCL DSN8MPG EXTERNAL ENTRY;

/******
 ** FIELDS SENT TO MESSAGE ROUTINE
 *****/
DCL MODULE CHAR (07) INIT ('DSN8IP3');
DCL OUTMSG CHAR (69);

/******
 * DECLARATION FOR SQL
 *****/
OEXEC SQL INCLUDE SQLCA; /* SQL COMMUNICATION AREA */
 /* SQL DECLARATION FOR VIEW PHONE */
EXEC SQL DECLARE VPHONE TABLE
 (LASTNAME VARCHAR(15) ,
 FIRSTNAME VARCHAR(12) ,
 MIDDLEINITIAL CHAR(1) ,
 PHONENUMBER CHAR(4) ,
 EMPLOYEENUMBER CHAR(6) ,
 DEPTNUMBER CHAR(3) NOT NULL,
 DEPTNAME VARCHAR(36) NOT NULL);
 /* STUCTURE FOR PHONE RECORD */

DCL 1 PPHONE,
2 LASTNAME CHAR (15) VAR,
2 FIRSTNAME CHAR (12) VAR,
2 MIDDLEINITIAL CHAR (1),
2 PHONENUMBER CHAR (4),
2 EMPLOYEENUMBER CHAR (6),
2 DEPTNUMBER CHAR (3),
2 DEPTNAME CHAR (36) VAR;
 /* SQL DECLARATION FOR VIEW VEMPLP*/
EXEC SQL DECLARE VEMPLP TABLE
 (EMPLOYEENUMBER CHAR(6) ,

```

```

 PHONENUMBER CHAR(4));
 /* STRUCTURE FOR PEMPLP RECORD */

DCL 1 PEMPLP,
 2 EMPLOYEENUMBER CHAR (6),
 2 PHONENUMBER CHAR (4);
1/***
/* SQL CURSORS
***** */
1/**

EXEC SQL DECLARE TELE1 CURSOR FOR
 SELECT *
 FROM VPHONE;

EXEC SQL DECLARE TELE2 CURSOR FOR
 SELECT *
 FROM VPHONE
 WHERE LASTNAME LIKE :INPUT_1.LNAME
 AND FIRSTNAME LIKE :INPUT_1.FNAME;

EXEC SQL DECLARE TELE3 CURSOR FOR
 SELECT *
 FROM VPHONE
 WHERE LASTNAME = :INPUT_1.LNAME
 AND FIRSTNAME LIKE :INPUT_1.FNAME;

1/***
/* SQL RETURN CODE HANDLING
***** */
1/**

EXEC SQL WHENEVER SQLERROR GOTO P3_DBERROR;
EXEC SQL WHENEVER SQLWARNING GOTO P3_DBERROR;
EXEC SQL WHENEVER NOT FOUND CONTINUE;

1/***
/* MAIN PROGRAM ROUTINE
***** */
1/**

/*INITIALIZATIONS */

OP3_START:
 IN_MESSAGE = '';
 OUT_MESSAGE = '';
 OUT_MESSAGE_LL = 83;
 MODNAME = 'DSN8IPNO';
 FIRST = '1'B;
 ITAB = 0; /* COUNTER */
0 CALL PLITDLI (THREE, GU_FKT, IOPCB, IN_MESSAGE);

 /* IF INVALID DL/I */
 /* STC-CODE ON GU MSG */
 /* PRINT ERROR MESSAGE */

0 IF IOPCB.STC_CODE ^= ' ' THEN
 DO;
 CALL DSN8MPG (MODULE, '064E', OUTMSG);
 ERROR = OUTMSG || IOPCB.STC_CODE;
 CALL P3_SEND;
 END;

1/***
/* SELECT ACTION
***** */
1/**

0 SELECT (ACTION);
 WHEN ('L') DO; /* ACTION - LIST */

1/***
/* REDISPLAY SELECTION SCREEN IF NO CRITERIA ENTERED
***** */
1/**

0 IF INPUT_1.LNAME = ' ' &
 INPUT_1.FNAME = ' ' THEN
 DO;
 CALL P3_SEND;
 GOTO P3_END;
 END;

 MODNAME = 'DSN8IPL0'; /* SELECT "LISTING" PANEL */

1/***
/* LIST ALL EMPLOYEES
***** */
1/**

0 IF INPUT_1.LNAME = '*' THEN /* LIST ALL EMPLOYEES */
 DO;

 EXEC SQL OPEN TELE1; /* OPEN CURSOR */

```

```

 EXEC SQL FETCH TELE1 /* GET FIRST RECORD */
 INTO :PPHONE;

0 IF SQLCODE = 100 THEN /* NO EMPLOYEES FOUND */
DO; /* PRINT ERROR MESSAGE*/
 MODNAME = 'DSN8IPNO';
 CALL DSN8MPG (MODULE, '008I', OUTMSG);
 ERROR = OUTMSG;
 CALL P3_SEND;
 GOTO P3_SELECT_20;
END;

 CALL P3_PREPARE_SCREEN; /* LIST FIRST EMPLOYEE*/

P3_SELECT_10:
 EXEC SQL FETCH TELE1 /* GET NEXT RECORD */
 INTO :PPHONE;

0 IF SQLCODE = 100 THEN /* FINISHED ? */
DO;
 ERROR = '';
 CALL P3_SEND;
 GOTO P3_SELECT_20;
END;

 CALL P3_PREPARE_SCREEN; /* LIST EMPLOYEE */
 /* CONTINUE */ /* */

P3_SELECT_20:
 EXEC SQL CLOSE TELE1; /* CLOSE CURSOR */
 GOTO P3_END;
END; /* END IF */

/***
/* LIST GENERIC EMPLOYEES */
/**/

ELSE DO; /* SELECT EMPLOYEES BY NAME*/
 /* SEARCH ON PART OF NAME? */
 IF INPUT_1.LNAME = '' /* IS NAME BLANK? */
 INPUT_1.LNAME = '%/%/%/%/%/%/%/%'; /* SET PATTERN */
 IF INDEX(INPUT_1.LNAME,'%') > 0 THEN
DO;
 /* YES, SEARCH ON */
 /* PART OF LAST NAME */
 /* (OPT. PART FIRST NAME) */
 /* TRANSLATE IT */
 INPUT_1.LNAME = TRANSLATE(INPUT_1.LNAME,'%', ' ');
 INPUT_1.FNAME = TRANSLATE(INPUT_1.FNAME,'%', ' ');

 EXEC SQL OPEN TELE2; /* OPEN CURSOR */
 /* */

 EXEC SQL FETCH TELE2 /* GET FIRST RECORD */
 INTO :PPHONE;

0 IF SQLCODE = 100 THEN /* NO EMPLOYEES FOUND */
DO; /* PRINT ERROR MESSAGE */
 MODNAME = 'DSN8IPNO';
 CALL DSN8MPG (MODULE, '008I', OUTMSG);
 ERROR = OUTMSG;
 CALL P3_SEND;
 GOTO P3_SELECT_40;
END;

 CALL P3_PREPARE_SCREEN; /* LIST FIRST EMPLOYEE */

P3_SELECT_30:
 EXEC SQL FETCH TELE2 /* GET NEXT RECORD */
 INTO :PPHONE;

0 IF SQLCODE = 100 THEN /* FINISHED? */
DO;
 ERROR = '';
 CALL P3_SEND;
 GOTO P3_SELECT_40;
END;

 CALL P3_PREPARE_SCREEN; /* LIST EMPLOYEE */
 /* CONTINUE */

P3_SELECT_40:
 EXEC SQL CLOSE TELE2; /* CLOSE CURSOR */
 GOTO P3_END;
END; /* END IF */

```

```

/***** LIST SPECIFIC EMPLOYEE(S) *****/
ELSE DO; /* NO - SEARCH ON LAST NAME*/
/*AND OPTIONALLY FIRST NAME*/
INPUT_1.FNAME = TRANSLATE(INPUT_1.FNAME, '%', ' ');
EXEC SQL OPEN TELE3; /* OPEN CURSOR */
EXEC SQL FETCH TELE3 /* GET FIRST RECORD */
INTO :PPHONE;
0 IF SQLCODE = 100 THEN /* EMPLOYEE NOT FOUND */
DO; /* PRINT ERROR MESSAGE */
MODNAME = 'DSN8IPNO';
CALL DSN8MPG (MODULE, '008I', OUTMSG);
ERROR = OUTMSG;
CALL P3_SEND;
GOTO P3_SELECT_60;
END;
CALL P3_PREPARE_SCREEN; /* LIST FIRST EMPLOYEE */
P3_SELECT_50:
EXEC SQL FETCH TELE3 /* GET NEXT RECORD */
INTO :PPHONE;
0 IF SQLCODE = 100 THEN /* FINISHED ? */
DO;
ERROR = '';
CALL P3_SEND;
GOTO P3_SELECT_60;
END;
CALL P3_PREPARE_SCREEN; /* LIST EMPLOYEE */
GOTO P3_SELECT_50; /* CONTINUE */
P3_SELECT_60:
EXEC SQL CLOSE TELE3; /* CLOSE CURSOR */
END; /* END IF */
END; /* END IF */
END; /* END WHEN */
/* CHANGE ERROR HANDLING */
/* FOR UPDATE */
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
/***** UPDATE PHONE NUMBERS FOR EMPLOYEES *****/
/* WHEN ('U') DO; /* TELEPHONE UPDATE */
0 WHEN ('U') DO; /* TELEPHONE UPDATE */
OUT_MESSAGE.LL = 110;
MODNAME = 'DSN8IPNO'; /* FIND WHICH NUMBERS HAVE */
/* BEEN UPDATED */
0 ERROR = ''; /* SET IN CASE NO UPDATES */
DO I = 1 TO 15;
IF INPUT_2.NEWNO(I) = ' ' THEN; /* NO UPDATE ON THIS LINE */
ELSE DO; /* PERFORM UPDATE */
EMPLOYEE_NO = INPUT_2.ENO(I);
PHONE_NO = INPUT_2.NEWNO(I);
0 EXEC SQL UPDATE VEMPLP
SET PHONENUMBER = :PHONE_NO
WHERE EMPLOYEENUMBER = :EMPLOYEE_NO;
/* UPDATE SUCCESSFUL */
/* PRINT CONFIRMATION */
/* MESSAGE */
0 IF SQLCODE = 0 THEN
DO;
CALL DSN8MPG (MODULE, '004I', OUTMSG);
ERROR = OUTMSG;
END; /* UPDATE FAILED */
/* PRINT ERROR MESSAGE */
ELSE
DO;
CALL DSN8MPG (MODULE, '007E', OUTMSG);
ERROR = OUTMSG;
GOTO P3_DBERROR2;

```

```

 END;

 END; /* END IF */
 END; /* END WHEN */
0 CALL P3_SEND;
END;

0 OTHERWISE /* UNFORMATTED SCREEN */
DO;
 OUT_MESSAGE.LL = 110;
 MODNAME = 'DSN8IPNO';
 CALL P3_SEND;
END;
END; /* END SELECT */
0 GOTO P3_END;
1/***/
/* SQL ERROR HANDLING */
/***/
0P3_DBERROR:
CALL DSN8MPG (MODULE, '060E', OUTMSG);
CHAR_SQLCODE = SQLCODE;
ERROR = OUTMSG||CHAR_SQLCOD;
PUT DATA (ERROR,SQLWARN0); /*PRINT ERROR MESSAGE */

0P3_DBERROR2:
0 CALL PLITDLI (THREE,CHNG_FKT,ALTPCB,IOLTERM);

0 IF ALTPCB.STC_CODE ^= ' ' THEN; /* PERFORM ROLLBACK */
ELSE CALL PLITDLI (FOUR,ISRT_FKT,ALTPCB,OUT_MESSAGE,MODNAME);
0 CALL PLITDLI (ONE,ROLL_FKT);
0 GOTO P3_END;

1/***/
/* PRINT INFORMATION ON SCREEN */
/***/
1P3_PREPARE_SCREEN:
PROC;
IF ITAB = 15 THEN /*IF ANOTHER PAGE */
DO; /* PRINT SCROLLING MESSAGE */
 CALL DSN8MPG (MODULE, '058I', OUTMSG);
 ERROR = OUTMSG;
 CALL P3_SEND;
 ITAB = 0; /* INITIALIZE COUNTER */
 OUT_MESSAGE.LL = 83; /* INITIALIZE LINE LENGTH */
END;

ITAB = ITAB + 1; /* INCREMENT COUNTER */
 /* MOVE DATA TO OUTPUT AREA*/
OUTPUT_2 (ITAB) = PPHONE , BY NAME;
OUT_MESSAGE.LL = OUT_MESSAGE.LL + 73;
RETURN;

END P3_PREPARE_SCREEN;
1P3_SEND:
PROC;

IF FIRST THEN
DO;
 CALL PLITDLI (FOUR,ISRT_FKT,IOPCB,OUT_MESSAGE,MODNAME);
 FIRST = '0'B;
END;

ELSE CALL PLITDLI (THREE,ISRT_FKT,IOPCB,OUT_MESSAGE);

0 IF IOPCB.STC_CODE = ' ' THEN RETURN;
 /* INVALID DL/I STC-CODE ON ISRT MSG*/
 /* PRINT ERROR MESSAGE */
 CALL DSN8MPG (MODULE, '065E', OUTMSG);
0 ERROR = OUTMSG||IOPCB.STC_CODE;
0 CALL PLITDLI (THREE,CHNG_FKT,ALTPCB,IOLTERM);

0 IF ALTPCB.STC_CODE ^= ' ' THEN; /* PERFORM ROLLBACK */
ELSE CALL PLITDLI (FOUR,ISRT_FKT,ALTPCB,OUT_MESSAGE,MODNAME);
0 CALL PLITDLI (ONE,ROLL_FKT);
RETURN;
1/***/
0END P3_SEND; /* END OF PROGRAM */
0P3_END:

```

```
END DSN8IP3;
```

## Referencia relacionada

[“Ejemplos de aplicaciones en IMS” en la página 1415](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

## DSNTEJ4C

ESTE JCL REALIZA LA CONFIGURACIÓN DE LA FASE 4 PARA LAS APLICACIONES DE MUESTRA EN SITIOS CON COBOL.

```
//*****
///* NAME = DSNTEJ4C
//*
///* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//* PHASE 4
//* COBOL, IMS
//*
//* LICENSED MATERIALS - PROPERTY OF IBM
//* 5650-DB2
//* (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
//*
//* STATUS = VERSION 12
//*
//* FUNCTION = THIS JCL PERFORMS THE PHASE 4 SETUP FOR THE SAMPLE
//* APPLICATIONS AT SITES WITH COBOL. IT PREPARES THE
//* COBOL IMS PROGRAM.
//*
//* RUN THIS JOB ANYTIME AFTER PHASE 2.
//*
//* CHANGE ACTIVITY =
//* 08/18/2014 Single-phase migration s21938_inst1 s21938
//*
//*****
//*
//JOBLIB DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
//* STEP 1: PREPARE SQL 0 PART OF PROGRAM
//PH04CS01 EXEC DSNHICOB,
// PARM.PC='HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
// NOXREF,'SQL(DB2)', 'DEC(31)',
// PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)'),
// PARM.LKED='XREF,NCAL',
// MEM=DSN8ICO
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8ICO),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT0
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8ICO),
// DISP=SHR
//COB.SYSIN DD DSN=&&DSNHOUT0
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB LOAD(DSN8ICO),
// DISP=SHR
//*
//* STEP 2: PREPARE SQL 1 PART OF PROGRAM
//PH04CS02 EXEC DSNHICOB,
// PARM.PC='HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
// NOXREF,'SQL(DB2)', 'DEC(31)',
// PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)'),
// PARM.LKED='XREF,NCAL',
// COND=(4,LT),
// MEM=DSN8IC1
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8IC1),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT1
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IC1),
// DISP=SHR
//COB.SYSIN DD DSN=&&DSNHOUT1
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB LOAD(DSN8IC1),
// DISP=SHR
//*
```

```

///* STEP 3: PREPARE SQL 2 PART OF PROGRAM
//PH04CS03 EXEC DSNHICOB,
// PARM.PC='HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
// NOXREF,'SQL(DB2)', 'DEC(31)'),
// PARM.COB=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)'),
// PARM.LKED='XREF,NCAL',
// COND=(4,LT),
// MEM=DSN8IC2
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8IC2),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT2
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR
//PC.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IC2),
// DISP=SHR
//COB.SYSIN DD DSN=&&DSNHOUT2
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IC2),
// DISP=SHR
///*
///* STEP 4: LINKEDIT THE ENTIRE PROGRAM
//PH04CS04 EXEC PGM=IEWL,PARM='LIST,XREF,LET',COND=(4,LT)
//SYSLIB DD DISP=SHR,DSN=CEE.V!R!M!.SCEELKED
// DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
// DD DISP=SHR,DSN=IMSVS.RESLIB
//SYSLIN DD DDNAME=SYSIN
//SYSLMOD DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(50,50))
//SYSIN DD *
INCLUDE SYSLIB(DFSLI000)
INCLUDE SYSLMOD(DSN8IC0)
INCLUDE SYSLMOD(DSN8IC1)
INCLUDE SYSLMOD(DSN8IC2)
INCLUDE SYSLMOD(DSN8MCG)
ENTRY DLITCB
NAME DSN8IC0(R)
///*
///* STEP 5: BIND THE PLAN
//PH04CS05 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLIB DD DSN=DSN!!0.DBRLIB.DATA,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 GRANT BIND, EXECUTE ON PLAN DSN8IC0
 TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE (DSN8IC!!) MEMBER(DSN8IC1) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE (DSNBIC!!) MEMBER(DSN8IC2) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8IC0) PKLIST(DSN8IC!!*) -
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END
///*
///* STEP 6: CREATE THE MFS MAPS
//PH04CS06 EXEC MFSUTL,COND=(4,LT)
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IPG),
// DISP=SHR
///*
///* STEP 7: CREATE THE MFS MAPS
//PH04CS07 EXEC MFSUTL,COND=(4,LT)
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IPD),
// DISP=SHR
///*
///* STEP 8: RUN THE PSBGEN
//PH04CS08 EXEC PSBGEN,MBR=DSN8IC0,COND=(4,LT)
//C.SYSIN DD *
 PRINT NOGEN
 PCB TYPE=TP,EXPRESS=YES,ALTRESP=YES,MODIFY=YES,SAMETRM=YES
 PSBGEN PSBNAME=DSN8IC0,LANG=COBOL
 END
///*
///* STEP 9: RUN THE ACBGEN

```

```

//PH04CS09 EXEC ACBGEN,COND=(4,LT)
//G.SYSIN DD *
BUILD PSB=DSN8IC0
//*
//*
 ALSO ADD MEMBER DSN8FIMS TO THE SYSDEF TO ADD TRANSACTIONS
//*

```

### Referencia relacionada

[“Ejemplos de aplicaciones en IMS” en la página 1415](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

## DSNTEJ4P

ESTE JCL REALIZA LA CONFIGURACIÓN DE LA FASE 4 PARA LAS APLICACIONES DE MUESTRA EN SITIOS CON PL/I.

```

//*****
//** NAME = DSNTEJ4P
//*
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 4
//** PL/I, IMS
//*
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM CORP. ALL RIGHTS RESERVED.
//*
//** STATUS = VERSION 12
//*
//** FUNCTION = THIS JCL PERFORMS THE PHASE 4 SETUP FOR THE SAMPLE
//** APPLICATIONS AT SITES WITH PL/I. IT PREPARES THE
//** PL/I IMS PROGRAM.
//*
//** RUN THIS JOB ANYTIME AFTER PHASE 2.
//*
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//*/
//*****
//JOBLIB DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
//*
//** STEP 1: PREPARE SQL 0 PART OF PROGRAM
//PH04PS01 EXEC DSNHPLI,MEM=DSN8IP0,
// PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',
// PARM.PLI='(NOPT,SOURCE,OBJECT,MARGINS(2,72,0))',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS))',
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IP0),
// DISP=SHR
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8IP0),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT0
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT0
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP0),
// DISP=SHR
//*
//** STEP 2: PREPARE SQL 1 PART OF PROGRAM
//PH04PS02 EXEC DSNHPLI,MEM=DSN8IP1,
// COND=(4,LT),
// PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',
// PARM.PLI='(NOPT,SOURCE,OBJECT,MARGINS(2,72,0),NORENT',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS))',
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IP1),
// DISP=SHR
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8IP1),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT1
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR

```

```

//PLI.SYSIN DD DSN=&&DSNHOUT1
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP1),
// DISP=SHR
///*
//** STEP 3: PREPARE SQL 2 PART OF PROGRAM
//PH04PS03 EXEC DSNHPLI,MEM=DSN8IP2,
// COND=(4,LT),
// PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',
// PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0),NORENT',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS)'),
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IP2),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8IP2),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT2
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT2
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP2),
// DISP=SHR
///*
//** STEP 4: PREPARE TELEPHONE PROGRAM
//PH04PS04 EXEC DSNHPLI,MEM=DSN8IP3,
// COND=(4,LT),
// PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',
// PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS)'),
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IP3),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8IP3),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT3
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT3
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP3),
// DISP=SHR
///*
//** STEP 5: PREPARE SQL 0 PART OF PROJECT APPLICATION
//PH04PS05 EXEC DSNHPLI,MEM=DSN8IP6,
// COND=(4,LT),
// PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',
// PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS)'),
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IP6),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8IP6),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT6
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT6
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP6),
// DISP=SHR
///*
//** STEP 6: PREPARE SQL 1 PART OF PROGRAM
//PH04PS06 EXEC DSNHPLI,MEM=DSN8IP7,
// COND=(4,LT),
// PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',
// PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0),NORENT',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS)'),
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IP7),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8IP7),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT7
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR

```

```

// DD DSN=DSN!!0.SSDNSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT7
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP7),
// DISP=SHR
///*
//** STEP 7: PREPARE SQL 2 PART OF PROGRAM
//PH04PS07 EXEC DSNHPLI,MEM=DSN8IP8,
// COND=(4,LT),
// PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),STDSQL(NO)',
// PARM.PLI='(NOPT,SOURCE,OBJECT,MARGINS(2,72,0),NORENT',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(IMS))',
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=DSN!!0.SSDNSAMP(DSN8IP8),
// DISP=SHR
//PC.DBRLIB DD DSN=DSN!!0.DBRLIB.DATA(DSN8IP8),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT8
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SSDNSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT8
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8IP8),
// DISP=SHR
///*
//** STEP 8: LINKEDIT PROGRAM TOGETHER
//PH04PS08 EXEC PGM=IEWL,PARM='LIST,XREF,LET',
// COND=(4,LT)
//SYSLIB DD DISP=SHR,DSN=CEE.V!R!M!.SCEELKED
// DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
// DD DISP=SHR,DSN=IMSVS.RESLIB
//SYSLIN DD DDNAME=SYSIN
//SYSLMOD DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(50,50))
//SYSIN DD *
INCLUDE SYSLIB(DFSLI000)
INCLUDE SYSLMOD(DSN8IP0)
INCLUDE SYSLMOD(DSN8IP1)
INCLUDE SYSLMOD(DSN8IP2)
INCLUDE SYSLMOD(DSN8MPG)
ENTRY CEESTART
NAME DSN8IP0(R)
INCLUDE SYSLIB(DFSLI000)
INCLUDE SYSLMOD(DSN8IP3)
INCLUDE SYSLMOD(DSN8MPG)
ENTRY CEESTART
NAME DSN8IH0(R)
INCLUDE SYSLIB(DFSLI000)
INCLUDE SYSLMOD(DSN8IP6)
INCLUDE SYSLMOD(DSN8IP7)
INCLUDE SYSLMOD(DSN8IP8)
INCLUDE SYSLMOD(DSN8MPG)
ENTRY CEESTART
NAME DSN8IQ0(R)
//*
//** STEP 9: BIND PROGRAMS
//PH04PS09 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRLIB DD DISP=SHR,DSN=DSN!!0.DBRLIB.DATA
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 GRANT BIND, EXECUTE ON PLAN DSN8IP0, DSN8IQ0, DSN8IH0
 TO PUBLIC;
//SYSTGIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE(DSN8IP!!) MEMBER(DSN8IP1) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(DSN8IP!!) MEMBER(DSN8IP2) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(DSN8IP!!) MEMBER(DSN8IP3) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(DSN8IP!!) MEMBER(DSN8IP7) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(DSN8IP!!) MEMBER(DSN8IP8) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8IP0) +

```

```

PKLIST(DSN8IP!!!.DSN8IP1, DSN8IP!!!.DSN8IP2) +
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8IQ0) +
PKLIST(DSN8IP!!!.DSN8IP7, DSN8IP!!!.DSN8IP8) +
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8IH0) PKLIST(DSN8IP!!!.DSN8IP3) +
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END
/***
/** STEP 10: CREATE MFS MAPS FOR ORGANIZATION APPLICATION
//PH04PS10 EXEC MFSUTL,COND=(4,LT)
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IPG),
// DISP=SHR
/***
/** STEP 11: CREATE MFS MAPS FOR ORGANIZATION APPLICATION
//PH04PS11 EXEC MFSUTL,COND=(4,LT)
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IPD),
// DISP=SHR
/***
/** STEP 12: CREATE MFS MAPS FOR PROJECT APPLICATION
//PH04PS12 EXEC MFSUTL,COND=(4,LT)
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IPF),
// DISP=SHR
/***
/** STEP 13: CREATE MFS MAPS FOR PROJECT APPLICATION
//PH04PS13 EXEC MFSUTL,COND=(4,LT)
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IPE),
// DISP=SHR
/***
/** STEP 14: CREATE MFS MAPS FOR TELEPHONE APPLICATION
//PH04PS14 EXEC MFSUTL,COND=(4,LT)
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IPL),
// DISP=SHR
/***
/** STEP 15: CREATE MFS MAPS FOR TELEPHONE APPLICATION
//PH04PS15 EXEC MFSUTL,COND=(4,LT)
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8IPN),
// DISP=SHR
/***
/** STEP 16: RUN PSBGEN
//PH04PS16 EXEC PSBGEN,MBR=DSN8IP0,COND=(4,LT)
//C.SYSIN DD *
 PRINT NOGEN
 PCB TYPE=TP,EXPRESS=YES,ALTRESP=YES,MODIFY=YES,SAMETRM=YES
 PSBGEN PSBNAME=DSN8IP0,LANG=PLI
END
/***
/** STEP 17: RUN ACBGEN
//PH04PS17 EXEC ACBGEN,COND=(4,LT)
//G.SYSIN DD *
 BUILD PSB=DSN8IP0
/***
/** STEP 18: RUN PSBGEN
//PH04PS18 EXEC PSBGEN,MBR=DSN8IQ0,COND=(4,LT)
//C.SYSIN DD *
 PRINT NOGEN
 PCB TYPE=TP,EXPRESS=YES,ALTRESP=YES,MODIFY=YES,SAMETRM=YES
 PSBGEN PSBNAME=DSN8IQ0,LANG=PLI
END
/***
/** STEP 19 : RUN ACBGEN
//PH04PS19 EXEC ACBGEN,COND=(4,LT)
//G.SYSIN DD *
 BUILD PSB=DSN8IQ0
/***
/** STEP 20 : RUN PSBGEN
//PH04PS20 EXEC PSBGEN,MBR=DSN8IH0,COND=(4,LT)
//C.SYSIN DD *
 PRINT NOGEN
 PCB TYPE=TP,EXPRESS=YES,ALTRESP=YES,MODIFY=YES,SAMETRM=YES
 PSBGEN PSBNAME=DSN8IH0,LANG=PLI
END
/***
/** STEP 21 : RUN ACBGEN
//PH04PS21 EXEC ACBGEN,COND=(4,LT)
//G.SYSIN DD *
 BUILD PSB=DSN8IH0

```

```

//*
//* ALSO ADD MEMBER DSN8FIMS TO THE SYSDEF, TO ADD TRANSACTIONS
/*

```

### Referencia relacionada

["Ejemplos de aplicaciones en IMS" en la página 1415](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el IMS entorno.

## Ejemplos de aplicaciones en CICS

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

*Tabla 184. Ejemplos de solicitudes de Db2 para CICS*

Aplicación	Nombre de programa	Nombre de miembro JCL	Descripción
Organización	DSN8CC0	DSNTEJ5C	CICS Aplicación de organización COBOL
	DSN8CC1		
	DSN8CC2		
Organización	DSN8CP0	DSNTEJ5P	CICS Solicitud de organización PL/I
	DSN8CP1		
	DSN8CP2		
Proyecto	DSN8CP6	DSNTEJ5P	CICS Solicitud de proyecto PL/I
	DSN8CP7		
	DSN8CP8		
Teléfono	DSN8CP3	DSNTEJ5P	CICS Aplicación para teléfonos PL/I. Este programa enumera los números de teléfono de los empleados y los actualiza si se solicita.

### Referencia relacionada

[Conjuntos de datos que utiliza el precompilador](#)

Cuando invoca el precompilador, necesita proporcionar conjuntos de datos que contienen entrada para el precompilador, como las sentencias de programación de host o sentencias SQL. También necesita proporcionar conjuntos de datos donde el precompilador pueda almacenar la salida, como el código de origen modificado o los mensajes de diagnóstico.

## DSN8CC0

ESTE MÓDULO EMITE CICS RECEIVE MAP TO RETRIEVE INPUT, CALLS DSN8CC1, AND ISSUE CICS ENVÍO DE MAPA DESPUÉS DE VOLVER.

```

IDENTIFICATION DIVISION.
*-----
* PROGRAM-ID. DSN8CC0.

**** DSN8CC0 - SUBSYSTEM INTERFACE MODULE FOR CICS/VС - COBOL ***
*
* MODULE NAME = DSN8CC0
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
* SUBSYSTEM INTERFACE MODULE
* CICS
* COBOL
*
*Licensed Materials - Property of IBM
*5605-DB2

```

```

*(C) COPYRIGHT 1982, 2010 IBM Corp. All Rights Reserved.
*
*STATUS = Version 10
*
* FUNCTION = THIS MODULE ISSUES CICS RECEIVE MAP TO RETRIEVE
* INPUT, CALLS DSN8CC1, AND ISSUE CICS SEND MAP
* AFTER RETURNING.
*
* NOTES =
* 1. THIS IS A CICS PSEUDO CONVERSATION PROGRAM WHICH
* INITIALIZES ITSELF WHEN A TERMINAL OPERATOR ENTERS
* INPUT AFTER VIEWING THE SCREEN SENT BY PREVIOUS
* ITERATIONS OF THE PROGRAM.
*
* DEPENDENCIES = TWO CICS MAPS(DSECTS) ARE REQUIRED:
* DSN8MCMG AND DSN8MCMD
* MODULE DSN8CC1 IS REQUIRED.
* DCLGEN STRUCTURE DSN8MCCS IS REQUIRED
* INCLUDED COBOL STRUCTURE DSN8MCCA IS
* REQUIRED.
*
* RESTRICTIONS = NONE
*
*
* MODULE TYPE =
* PROCESSOR = DB2 PRECOMPILER,CICS TRANSLATOR,COBOL COMPILE
* MODULE SIZE = SEE LINK-EDIT
* ATTRIBUTES = REUSABLE
*
* ENTRY POINT = DSN8CC0
* PURPOSE = SEE FUNCTION
* LINKAGE = CICS/OS/VS ENTRY
*
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
*
* SYMBOLIC LABEL/NAME = DSN8CCGI
* DESCRIPTION = CICS/OS/VS BMS MAP FOR GENERAL INPUT
*
* SYMBOLIC LABEL/NAME = DSN8CCDI
* DESCRIPTION = CICS/OS/VS BMS MAP FOR DETAIL INPUT
*
* OUTPUT = PARAMETERS EXPLICITLY RETURNED:
*
* SYMBOLIC LABEL/NAME = DSN8CCG0
* DESCRIPTION = CICS/OS/VS BMS MAP FOR GENERAL OUTPUT
*
* SYMBOLIC LABEL/NAME = DSN8CCD0
* DESCRIPTION = CICS/OS/VS BMS MAP FOR DETAIL OUTPUT
*
*
* EXIT-NORMAL = CICS RETURN TRANSID
*
* EXIT-ERROR = SQL ERROR FOR SQL ERRORS
* CICS ABEND FOR CICS PROBLEMS
*
* RETURN CODE = NONE
*
* ABEND CODES = LSCR - LOGICAL SCREEN SET INCORRECTLY
*
* ERROR-MESSAGES = NONE
*
*
* EXTERNAL REFERENCES = COMMON CICS REQUIREMENTS
* ROUTINES/SERVICES =
* CICS/VS SERVICES
* DSN8CC1 - SQL 1 MAINLINE CODE
*
* DATA-AREAS =
* DSN8MCCA - PARAMETER TO BE PASSED TO DSN8CC1
* COMMON AREA
* DSN8MCCS - DECLARE CONVERSATION STATUS
* DSN8MCC2 - COMMON AREA PART 2
* DSN8MCMD - CICS/OS/VS COBOL MAP, ORGANIZATION
* DSN8MCMG - CICS/OS/VS COBOL MAP, ORGANIZATION
*
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA
*
* TABLES = NONE
*
* CHANGE-ACTIVITY =
* - 10/18/2005 PK03311 INITIALIZE UNINITIALIZED STORAGE @01

```

```

* * *PSEUDOCODE*
*
* PROCEDURE
* DECLARATIONS.
* ALLOCATE COBOL WORK AREA FOR COMMAREA.
* PUT MODULE NAME 'DSN8CC0' IN AREA USED BY ERROR-HANDLER.
* PUT CICS EIBTRMID IN PCONVSTA.CONVID TO BE PASSED TO
* DSN8CC1.
* RETRIEVE LASTCR FROM VCONA USING THE CONVID TO DETERMINE
* WHICH OF THE TWO BMS MAPS SHOULD BE USED TO MAP IN DATA.
*
*
* IF RETRIEVAL OF MAPS IS SUCCESSFUL, THEN DO;
* EXEC CICS RECEIVE MAP ACCORDING TO SPECIFIED LASTSCR
*
* IF MAPFAIL CONDITION IS RAISED* THEN DO;
* COMPARM.PFKIN = '00'
* GO TO CC0SEND
* END
*
* ELSE
* PUT DATA FROM MAP INTO COMPARM **
*
* ELSE
* IT IS A NEW CONVERSATION
* AND NO EXEC CICS RECEIVE MAP IS ISSUED.
*
* CC0SEND:
* EXEC CICS LINK PROGRAM('DSN8CC1') COMMAREA(COMMAREA).
* UPON RETURN FROM DSN8CC1, EXEC CICS SEND MAP ACCORDING TO
* THE TYPE SPECIFIED IN PCONVSTA.LASTSCR.
* EXEC CICS RETURN TRANSID(D8CS).
*
* END.
*
* * I.E. LAST CONVERSATION EXISTS, BUT OPERATOR HAD ENTERED
* DATA FROM A CLEARED SCREEN OR HAD ERASED ALL DATA ON A
* FORMATTED SCREEN AND PRESSED ENTER
*
* ** COMPARM.PFKIN = PF KEY ACTUALLY USED I.E. '01' FOR
* PF1 ..
----------*-----*
```

ENVIRONMENT DIVISION.

```

-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
-----*
```

DATA DIVISION.

-----\*

WORKING-STORAGE SECTION.

77 FOUND PIC S99.  
 EXEC SQL INCLUDE SQLCA END-EXEC.  
 EXEC SQL INCLUDE DSN8MCC2 END-EXEC.

01 COMMAREA.  
 EXEC SQL INCLUDE DSN8MCCA END-EXEC.  
 EXEC SQL INCLUDE DSN8MCCS END-EXEC.  
 EXEC SQL INCLUDE DSN8MCMG END-EXEC.  
 EXEC SQL INCLUDE DSN8MCMD END-EXEC.

\*\*\*\*\*  
\* MAPD REDEFINES THE COBOL STRUCTURE ASSOCIATED WITH THE  
\* CICS MAP DSN8CCD.  
\*\*\*\*\*

01 MAPD REDEFINES DSN8CCDI.  
 02 FILLER PIC X(387).  
 02 SUBMAP OCCURS 15 TIMES.  
 03 COL1LEN PIC S9(4) COMP.  
 03 COL1ATTR PIC X(1).  
 03 COL1DATA PIC X(37).  
 03 COL2LEN PIC S9(4) COMP.  
 03 COL2ATTR PIC X(1).  
 03 COL2DATA PIC X(40).

\*\*\*\*\*  
\* PFKEYS IS AN ARRAY OF 24 ELEMENTS REPRESENTING THE DIFFERENT  
\* PFKEYS AS THEY WOULD BE REPRESENTED IN EIBAID.  
\*\*\*\*\*

01 PFKEYS-DUMB.  
 02 PFKEYS-ALL PIC X(24) VALUE '123456789:@ABCDEFGHI>.<' .  
 02 PFKEYS REDEFINES PFKEYS-ALL PIC X(1) OCCURS 24 TIMES.

\*\*\*\*\*  
\* PFK IS AN ARRAY OF 12 TWO-BYTE CHARS REPRESENTING THE PFKEYS  
\* ALLOWED AS INPUT TO DSN8CC1 AND DSN8CC2 ETC.

```

*****-
01 PFK-DUMB.
02 PFK-ALL PIC X(24) VALUE '010203040506070809101112'.
02 PFK REDEFINES PFK-ALL PIC X(2) OCCURS 12 TIMES.

PROCEDURE DIVISION.

*****-
* SQL RETURN CODE HANDLING
*****-
EXEC SQL WHENEVER SQLERROR GO TO DB-ERROR END-EXEC
EXEC SQL WHENEVER SQLWARNING GO TO DB-ERROR END-EXEC.

*****-
* ALLOCATE COBOL WORK AREA / INITIALIZE VARIABLES
*****-
* INIT AREA INCLUDED BY DSN8MCCA
MOVE SPACES TO COMMAREA.

* INIT AREA INCLUDED BY DSN8MCMS @001
MOVE SPACES TO PCONA.

* INIT AREA INCLUDED BY DSN8MCMG @001
MOVE ZEROES TO ATITLEL OF DSN8CCGI.
MOVE SPACES TO ATITLEI OF DSN8CCGI.
MOVE ZEROES TO AMAJSYSL OF DSN8CCGI.
MOVE SPACES TO AMAJSYSI OF DSN8CCGI.
MOVE ZEROES TO AACTIONL OF DSN8CCGI.
MOVE SPACES TO AACTIONI OF DSN8CCGI.
MOVE ZEROES TO ADESCL2L OF DSN8CCGI.
MOVE SPACES TO ADESCL2I OF DSN8CCGI.
MOVE ZEROES TO AOBJECTL OF DSN8CCGI.
MOVE SPACES TO AOBJECTI OF DSN8CCGI.
MOVE ZEROES TO ADESCL3L OF DSN8CCGI.
MOVE SPACES TO ADESCL3I OF DSN8CCGI.
MOVE ZEROES TO ASEARCHL OF DSN8CCGI.
MOVE SPACES TO ASEARCHI OF DSN8CCGI.
MOVE ZEROES TO ADESCL4L OF DSN8CCGI.
MOVE SPACES TO ADESCL4I OF DSN8CCGI.
MOVE ZEROES TO ADATAL OF DSN8CCGI.
MOVE SPACES TO ADATAI OF DSN8CCGI.
MOVE ZEROES TO AMSGL OF DSN8CCGI.
MOVE SPACES TO AMSGI OF DSN8CCGI.
MOVE ZEROES TO ALINEL(1).
MOVE SPACES TO ALINEI(1).
MOVE ZEROES TO ALINEL(2).
MOVE SPACES TO ALINEI(2).
MOVE ZEROES TO ALINEL(3).
MOVE SPACES TO ALINEI(3).
MOVE ZEROES TO ALINEL(4).
MOVE SPACES TO ALINEI(4).
MOVE ZEROES TO ALINEL(5).
MOVE SPACES TO ALINEI(5).
MOVE ZEROES TO ALINEL(6).
MOVE SPACES TO ALINEI(6).
MOVE ZEROES TO ALINEL(7).
MOVE SPACES TO ALINEI(7).
MOVE ZEROES TO ALINEL(8).
MOVE SPACES TO ALINEI(8).
MOVE ZEROES TO ALINEL(9).
MOVE SPACES TO ALINEI(9).
MOVE ZEROES TO ALINEL(10).
MOVE SPACES TO ALINEI(10).
MOVE ZEROES TO ALINEL(11).
MOVE SPACES TO ALINEI(11).
MOVE ZEROES TO ALINEL(12).
MOVE SPACES TO ALINEI(12).
MOVE ZEROES TO ALINEL(13).
MOVE SPACES TO ALINEI(13).
MOVE ZEROES TO ALINEL(14).
MOVE SPACES TO ALINEI(14).
MOVE ZEROES TO ALINEL(15).
MOVE SPACES TO ALINEI(15).
MOVE ZEROES TO APFKEYL OF DSN8CCGI.
MOVE SPACES TO APFKEYI OF DSN8CCGI.

* INIT AREA INCLUDED BY DSN8MCMD @001
MOVE ZEROES TO BTITLEL OF DSN8CCDI.
MOVE SPACES TO BTITLEI OF DSN8CCDI.
MOVE ZEROES TO BMAJSYSL OF DSN8CCDI.
MOVE SPACES TO BMAJSYSI OF DSN8CCDI.
MOVE ZEROES TO BACTIONL OF DSN8CCDI.
MOVE SPACES TO BACTIONI OF DSN8CCDI.
MOVE ZEROES TO BDESCL2L OF DSN8CCDI.

```

```

MOVE SPACES TO BDESCL2I OF DSN8CCDI.
MOVE ZEROES TO BOBJECTL OF DSN8CCDI.
MOVE SPACES TO BOBJECTI OF DSN8CCDI.
MOVE ZEROES TO BDESCL3L OF DSN8CCDI.
MOVE SPACES TO BDESCL3I OF DSN8CCDI.
MOVE ZEROES TO BSEARCHL OF DSN8CCDI.
MOVE SPACES TO BSEARCHI OF DSN8CCDI.
MOVE ZEROES TO BDESCL4L OF DSN8CCDI.
MOVE SPACES TO BDESCL4I OF DSN8CCDI.
MOVE ZEROES TO BDATAL OF DSN8CCDI.
MOVE SPACES TO BDATAI OF DSN8CCDI.
MOVE ZEROES TO BMSGL OF DSN8CCDI.
MOVE SPACES TO BMSGI OF DSN8CCDI.
MOVE ZEROES TO LINE1F1L OF DSN8CCDI.
MOVE SPACES TO LINE1F1I OF DSN8CCDI.
MOVE ZEROES TO LINE1F2L OF DSN8CCDI.
MOVE SPACES TO LINE1F2I OF DSN8CCDI.
MOVE ZEROES TO LINE2F1L OF DSN8CCDI.
MOVE SPACES TO LINE2F1I OF DSN8CCDI.
MOVE ZEROES TO LINE2F2L OF DSN8CCDI.
MOVE SPACES TO LINE2F2I OF DSN8CCDI.
MOVE ZEROES TO LINE3F1L OF DSN8CCDI.
MOVE SPACES TO LINE3F1I OF DSN8CCDI.
MOVE ZEROES TO LINE3F2L OF DSN8CCDI.
MOVE SPACES TO LINE3F2I OF DSN8CCDI.
MOVE ZEROES TO LINE4F1L OF DSN8CCDI.
MOVE SPACES TO LINE4F1I OF DSN8CCDI.
MOVE ZEROES TO LINE4F2L OF DSN8CCDI.
MOVE SPACES TO LINE4F2I OF DSN8CCDI.
MOVE ZEROES TO LINE5F1L OF DSN8CCDI.
MOVE SPACES TO LINE5F1I OF DSN8CCDI.
MOVE ZEROES TO LINE5F2L OF DSN8CCDI.
MOVE SPACES TO LINE5F2I OF DSN8CCDI.
MOVE ZEROES TO LINE6F1L OF DSN8CCDI.
MOVE SPACES TO LINE6F1I OF DSN8CCDI.
MOVE ZEROES TO LINE6F2L OF DSN8CCDI.
MOVE SPACES TO LINE6F2I OF DSN8CCDI.
MOVE ZEROES TO LINE7F1L OF DSN8CCDI.
MOVE SPACES TO LINE7F1I OF DSN8CCDI.
MOVE ZEROES TO LINE7F2L OF DSN8CCDI.
MOVE SPACES TO LINE7F2I OF DSN8CCDI.
MOVE ZEROES TO LINE8F1L OF DSN8CCDI.
MOVE SPACES TO LINE8F1I OF DSN8CCDI.
MOVE ZEROES TO LINE8F2L OF DSN8CCDI.
MOVE SPACES TO LINE8F2I OF DSN8CCDI.
MOVE ZEROES TO LINE9F1L OF DSN8CCDI.
MOVE SPACES TO LINE9F1I OF DSN8CCDI.
MOVE ZEROES TO LINE9F2L OF DSN8CCDI.
MOVE SPACES TO LINE9F2I OF DSN8CCDI.
MOVE ZEROES TO LINEAF1L OF DSN8CCDI.
MOVE SPACES TO LINEAF1I OF DSN8CCDI.
MOVE ZEROES TO LINEAF2L OF DSN8CCDI.
MOVE SPACES TO LINEAF2I OF DSN8CCDI.
MOVE ZEROES TO LINEBF1L OF DSN8CCDI.
MOVE SPACES TO LINEBF1I OF DSN8CCDI.
MOVE ZEROES TO LINEBF2L OF DSN8CCDI.
MOVE SPACES TO LINEBF2I OF DSN8CCDI.
MOVE ZEROES TO LINECF1L OF DSN8CCDI.
MOVE SPACES TO LINECF1I OF DSN8CCDI.
MOVE ZEROES TO LINECF2L OF DSN8CCDI.
MOVE SPACES TO LINECF2I OF DSN8CCDI.
MOVE ZEROES TO LINEDF1L OF DSN8CCDI.
MOVE SPACES TO LINEDF1I OF DSN8CCDI.
MOVE ZEROES TO LINEDF2L OF DSN8CCDI.
MOVE SPACES TO LINEDF2I OF DSN8CCDI.
MOVE ZEROES TO LINEEF1L OF DSN8CCDI.
MOVE SPACES TO LINEEF1I OF DSN8CCDI.
MOVE ZEROES TO LINEEF2L OF DSN8CCDI.
MOVE SPACES TO LINEEF2I OF DSN8CCDI.
MOVE ZEROES TO LINEFF1L OF DSN8CCDI.
MOVE SPACES TO LINEFF1I OF DSN8CCDI.
MOVE ZEROES TO LINEFF2L OF DSN8CCDI.
MOVE SPACES TO LINEFF2I OF DSN8CCDI.
MOVE ZEROES TO BPFKEYL OF DSN8CCDI.
MOVE SPACES TO BPFKEYI OF DSN8CCDI.

*
MOVE 'DSN8CC0' TO MAJOR IN DSN8-MODULE-NAME.
MOVE '0' TO MAJSYS IN OUTAREA.
MOVE '0' TO EXITCODE.
MOVE EIBTRMID TO CICSID OF PCONVSTA.
MOVE CONVID OF PCONVSTA TO SAVE-CONVID.

```

```

*****-

* TRY TO RETRIEVE LAST CONVERSATION. IF SUCCESSFUL, USE THE

* LAST SCREEN SPECIFIED TO RECEIVE INPUT FROM TERMINAL.

*****-

EXEC SQL SELECT LASTSCR

 INTO :PCONA.LASTSCR

 FROM VCONA

 WHERE CONVID = :SAVE-CONVID END-EXEC.

*****-

* IF LAST CONVERSATION DOES NOT EXIST, THEN DO NOT ATTEMPT TO

* RECEIVE INPUT MAP. GO DIRECTLY TO VALIDATION MODULES

* TO GET TITLE ETC. FOR OUTPUT MAP.

*****-

IF SQLCODE = +100 THEN

 GO TO CC0SEND.

*****-

* IF LAST CONVERSATION EXISTS, BUT OPERATOR HAS ENTERED DATA N

* FROM A CLEARED SCREEN OR HAD ERASED ALL DATA ON A FORMATTED N

* SCREEN AND PRESSED ENTER THEN N

* MOVE DATA INTO CORRESPONDING FIELDS IN INAREA AND GO TO N

* VALIDATION MODULES.

*****-

EXEC CICS HANDLE CONDITION MAPFAIL (CC0SEND) END-EXEC.

IF LASTSCR OF PCONA NOT = 'DSN8002' THEN

 GO TO CC0-LABEL1.

* **DSN8002

* **DETAIL MAP

* **MOVE DATA INTO

* **INPUT FIELDS

EXEC CICS RECEIVE MAP ('DSN8CCD') MAPSET ('DSN8CCD')

 END-EXEC.

IF BMAJSYSL NOT = 0 THEN MOVE BMAJSYSL TO MAJSYS OF INAREA

 ELSE MOVE '0' TO MAJSYS OF INAREA.

IF BACTIONL NOT = 0 THEN MOVE BACTIONL TO ACTION OF INAREA

 ELSE MOVE SPACES TO ACTION OF INAREA.

IF BOBJECTL NOT = 0 THEN MOVE BOBJECTL TO OBJFLD OF INAREA

 ELSE MOVE SPACES TO OBJFLD OF INAREA.

IF BSEARCHL NOT = 0 THEN MOVE BSEARCHL TO SRCH OF INAREA

 ELSE MOVE SPACES TO SRCH OF INAREA.

IF BDATAL NOT = 0 THEN MOVE BDATAL TO DATAIN OF INAREA

 ELSE MOVE SPACES TO DATAIN OF INAREA.

MOVE 1 TO I.

* **GO TO VALIDATION MODULES

GO TO CC0-LABELX.

* **ERROR ON LASTSCREEN?

CC0-LABEL1.

IF LASTSCR OF PCONA NOT = 'DSN8001' THEN

 EXEC CICS ABEND ABCODE ('LSCR') END-EXEC

 GOBACK.

* **USING LAST SCREEN

* **SPECIFIED TO RECEIVE

* **INPUT FROM TERMINAL

EXEC CICS RECEIVE MAP ('DSN8CCG') MAPSET('DSN8CCG') END-EXEC.

*****-

* IF DATA IS RECEIVED FOR A FIELD, THEN MOVE THE DATA INTO THE

* CORRESPONDING FIELD IN INAREA, OTHERWISE MOVE BLANKS.

*****-

IF AMAJSYSL NOT = 0 THEN MOVE AMAJSYSL TO MAJSYS OF INAREA

 ELSE MOVE '0' TO MAJSYS OF INAREA.

IF AACTIONL NOT = 0 THEN MOVE AACTIONL TO ACTION OF INAREA

 ELSE MOVE SPACES TO ACTION OF INAREA.

IF AOBJECTL NOT = 0 THEN MOVE AOBJECTL TO OBJFLD OF INAREA

 ELSE MOVE SPACES TO OBJFLD OF INAREA.

IF ASEARCHL NOT = 0 THEN MOVE ASEARCHL TO SRCH OF INAREA

 ELSE MOVE SPACES TO SRCH OF INAREA.

IF ADATAL NOT = 0 THEN MOVE ADATAL TO DATAIN OF INAREA

 ELSE MOVE SPACES TO DATAIN OF INAREA.

```

```

GO TO CC0-LABEL3.

CC0-LABELX.
 IF COL2LEN(I) NOT = 0 THEN MOVE COL2DATA(I) TO TRANDATA(I)
 ELSE MOVE SPACES TO TRANDATA(I).
 ADD 1 TO I.

* ** CC0-LABELX LOOP
CC0-LOOPX.
 PERFORM CC0-LABELX UNTIL I > 15.

CC0-LABEL3.
 MOVE 1 TO I.
 MOVE 0 TO FOUND.

*****-
* CONVERT THE PFKEY INFO IN EIBAID TO THE FORM ACCEPTED
* BY DSN8CC1 AND DSN8CC2 EG. PF1 = '01' AND PF13 = '01'.
*****-
CC0-LABEL4.

*
* **PF KEYS 1-12
 IF PFKEYS(I) = EIBAID THEN MOVE 1 TO FOUND
 ELSE ADD 1 TO I.

* ** CC0-LABEL4 LOOP
CC0-LOOP4.
 PERFORM CC0-LABEL4 UNTIL
 I > 24 OR FOUND = 1.

*
* **PF KEYS > 12
CC0-LABEL5.
 IF I > 12 THEN SUBTRACT 12 FROM I.
 IF FOUND = 1 THEN
 MOVE PFK(I) TO PFKIN OF INAREA
 ELSE MOVE SPACES TO PFKIN OF INAREA.
 GO TO CC0-LABEL6.

*****-
* GO TO DSN8CC1, GET DCLGEN STRUCTURES AND TABLE DCL
*****-

CC0SEND.
 MOVE SPACES TO INAREA.
 MOVE '00' TO PFKIN OF INAREA.

CC0-LABEL6.
 MOVE '0' TO MAJSYS IN INAREA.
 EXEC CICS LINK PROGRAM ('DSN8CC1') COMMAREA(COMMAREA)
 LENGTH(3000) END-EXEC.
 GO TO CC0-NORMAL.
 EXEC SQL INCLUDE DSN8MCXX END-EXEC.

*****-
* AFTER RETURN FROM DSN8CC1, MOVE DATA TO OUTPUT MAP AREA AND
* SEND MAP ACCORDING TO MAP SPECIFIED IN LASTSCR OF PCONVSTA.
*****-
CC0-NORMAL.
 IF LASTSCR OF PCONVSTA = 'DSN8002 ' THEN GO TO CC0-LABEL9.

*
* **MOVE DATA INTO
* **OUTPUT FIELDS
 MOVE HTITLE OF OUTAREA TO ATITLEO.
 MOVE MAJSYS OF OUTAREA TO AMAJSYSO.
 MOVE ACTION OF OUTAREA TO AACTIONO.
 MOVE OBJFLD OF OUTAREA TO AOBJECTO.
 MOVE SRCH OF OUTAREA TO ASEARCHO.
 MOVE DATAOUT TO ADATAO.
 MOVE MSG OF OUTAREA TO AMSGO.
 MOVE DESC2 OF OUTAREA TO ADESCL20.
 MOVE DESC3 OF OUTAREA TO ADESCL30.
 MOVE DESC4 OF OUTAREA TO ADESCL40.
 MOVE PFKTEXT OF OUTAREA TO APFKEYO.
 MOVE 1 TO I.

*
* **SEND MAP ACCORDING TO
* **PREVIOUS SCREEN
CC0-LABEL7.
 MOVE LINE0(I) TO ALINE0(I).
 ADD 1 TO I.

```

```

* **CC0-LABEL7 LOOP
CC0-LOOP7.
 PERFORM CC0-LABEL7 UNTIL
 I > 15.

*****-*
* CREATES A DYNAMIC CURSOR
*****-*

* **SET CURSOR POSITION
CC0-LABEL8.
 MOVE ZEROES TO CURSOR-VALUE.
 IF AACTIONO = SPACES THEN MOVE +179 TO CURSOR-VALUE
 ELSE IF AOBJECTO = SPACES THEN MOVE +259 TO CURSOR-VALUE
 ELSE IF ASEARCHO = SPACES THEN MOVE +339 TO CURSOR-VALUE
 ELSE IF ADATAO = SPACES OR AACTIONO = 'D' OR 'E' THEN
 MOVE +419 TO CURSOR-VALUE.

* **SEND OUTPUT MAP
IF CURSOR-VALUE = ZEROES THEN
 EXEC CICS SEND MAP('DSN8CCG') MAPSET('DSN8CCG') END-EXEC
ELSE
 EXEC CICS SEND MAP('DSN8CCG') MAPSET('DSN8CCG') ERASE
 CURSOR(CURSOR-VALUE) END-EXEC.

* **FINISHED?
IF EXITCODE = '1' THEN GO TO CC0-LABEL12.
EXEC CICS RETURN TRANSID('D8CS') END-EXEC.

*****-*
* MOVES DATA FROM OUTPUT MAP AREA TO
* RECEIVE MAP ACCORDING TO MAP SPECIFIED IN LASTSCR OF PCONVST
*****-*

* **MOVE DATA
* **FROM OUTPUT FIELDS
CC0-LABEL9.
 MOVE HTITLE OF OUTAREA TO BTITLEO.
 MOVE MAJSYS OF OUTAREA TO BMMAJSYSO.
 MOVE ACTION OF OUTAREA TO BACTIONO.
 MOVE OBJFLD OF OUTAREA TO BOBJECTO.
 MOVE SRCH OF OUTAREA TO BSEARCHO.
 MOVE DATAOUT TO BDATAO.
 MOVE MSG OF OUTAREA TO BMSGO.
 MOVE DESC2 OF OUTAREA TO BDDESCL20.
 MOVE DESC3 OF OUTAREA TO BDDESCL30.
 MOVE DESC4 OF OUTAREA TO BDDESCL40.
 MOVE PFKTEXT OF OUTAREA TO BPKEYO.
 MOVE 1 TO I.

* **RECEIVE MAP ACCORDING
* **TO PREVIOUS SCREEN

CC0-LABEL10.
 MOVE FIELD1(I) TO COL1DATA(I) .
* ** CHECK FOR ATTRIBUTE OF X'C0C1'
* IF ATTR(I) = -16191 THEN MOVE -1 TO COL2LEN(I).
 MOVE ATTR2(I) TO COL2ATTR(I) .
 MOVE FIELD2(I) TO COL2DATA(I) .
 ADD 1 TO I.

* ** CC0-LABEL10 LOOP
CC0-LOOP10.
 PERFORM CC0-LABEL10 UNTIL
 I > 15.

CC0-LABEL11.
*****-*
* CREATES A DYNAMIC CURSOR
*****-*

* **SET CURSOR POSITION
 MOVE ZEROES TO CURSOR-VALUE.
 IF BACTIONO = SPACES THEN MOVE +179 TO CURSOR-VALUE
 ELSE IF BOBJECTO = SPACES THEN MOVE +259 TO CURSOR-VALUE
 ELSE IF BSEARCHO = SPACES THEN MOVE +339 TO CURSOR-VALUE
 ELSE IF BDATAO = SPACES OR BACTIONO = 'D' OR 'E' THEN
 MOVE +419 TO CURSOR-VALUE.

* **SEND INPUT MAP

```

```

IF CURSOR-VALUE = ZEROES THEN
 EXEC CICS SEND MAP('DSN8CCD') MAPSET('DSN8CCD') END-EXEC
ELSE
 EXEC CICS SEND MAP('DSN8CCD') MAPSET('DSN8CCD') ERASE
 CURSOR(CURSOR-VALUE) END-EXEC.

*
* **FINISHED?

IF EXITCODE = '1' THEN GO TO CC0-LABEL12.
EXEC CICS RETURN TRANSID('D8CS') END-EXEC.
GOBACK.

*
* **RETURN
CC0-LABEL12.
EXEC CICS RETURN END-EXEC.
GOBACK.

```

### Referencia relacionada

[“Ejemplos de aplicaciones en CICS” en la página 1461](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

## DSN8CC1

ESTE MÓDULO REALIZA LOS INCLUIDOS PARA INTRODUCIR LAS ESTRUCTURAS DE TABLA SQL DCLS Y DCLGEN, ASÍ COMO EL ÁREA DE PARÁMETROS.

```

IDENTIFICATION DIVISION.

PROGRAM-ID. DSN8CC1.

***** DSN8CC1 - SQL 1 MAINLINE FOR CICS - COBOL ****
*
* MODULE NAME = DSN8CC1
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
* SQL 1 MAINLINE
* CICS
* COBOL
*
*Licensed Materials - Property of IBM
*5605-DB2
*(C) COPYRIGHT 1982, 2010 IBM Corp. All Rights Reserved.
*
*STATUS = Version 10
*
* FUNCTION = THIS MODULE PERFORMS THE INCLUDES TO BRING IN THE
* SQL TABLE DCLS AND DCLGEN STRUCTURES AS WELL AS
* THE PARAMETER AREA.
*
* NOTES =
* DEPENDENCIES = CALLED BY DSN8CC0, CALLS DSN8CC2(CICS LINKS).
* RESTRICTIONS = NONE
*
* MODULE TYPE =
* PROCESSOR = DB2 PRECOMPILER,CICS TRANSLATOR,COBOL COMPILER
* MODULE SIZE = SEE LINK_EDIT
* ATTRIBUTES = REUSABLE
*
* ENTRY POINT = DSN8CC1
* PURPOSE = SEE FUNCTION
* LINKAGE = INCLUDED BY MODULE DSN8MC1
*
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
*
* SYMBOLIC LABEL/NAME = NONE
* DESCRIPTION = NONE
*
* OUTPUT = PARAMETERS EXPLICITLY RETURNED:
*
* SYMBOLIC LABEL/NAME = NONE
* DESCRIPTION = NONE
*
* EXIT-NORMAL = DSN8CC0
*
* EXIT-ERROR = DSN8CC0
*
```

```

* RETURN CODE = NONE
*
* ABEND CODES = NONE
*
* ERROR-MESSAGES = NONE
*
* EXTERNAL REFERENCES =
* ROUTINES/SERVICES = DSN8CC2
*
* DATA-AREAS =
* DSN8MCCA - COBOL STRUCTURE FOR DFHCOMMAREA
* DSN8MCCS - VCONA TABLE DCL AND PCONA DCLGEN
* DSN8MCC2 - COMMON AREA PART 2
* DSN8MCOV - VOPTVAL TABLE DCL & POPTVAL DCLGEN
* DSN8MCVO - FINDS VALID OPTIONS FOR ACTION,
* OBJECT, SEARCH CRITERIA
* DSN8MC1 - SQL1 COMMON MODULE FOR IMS AND CICS
* DSN8MC3 - DSN8MC5 - VALIDATION MODULES CALLED BY DSN8MC0
* DSN8MCXX - SQL ERROR HANDLER
*
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA
*
* TABLES = NONE
*
* CHANGE-ACTIVITY =
* - 10/18/2005 PK03311 INITIALIZE UNINITIALIZED STORAGE @01
*
* *PSEUDOCODE*
*
* PROCEDURE
* INCLUDE DECLARATIONS.
* INCLUDE DSN8MC1.
* INCLUDE ERROR HANDLER.
*
* CC1EXIT: (REFERENCED BY DSN8MC1)
* EXEC CICS RETURN.
*
* CC1CALL: (REFERENCED BY DSN8MC1)
* EXEC CICS LINK PROGRAM('DSN8CC2')
* COMMAREA(DFHCOMMAREA).
* GO TO MC1SAVE. (LABEL IN DSN8MC1)
*
* INCLUDE VALIDATION MODULES.
*
* END.
*****ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.
***** * DECLARE FIELD PASSED TO MESSAGE ROUTINE
* * DECLARE CONVERSATION STATUS
* * DECLARE MESSAGE TEXT
* * DECLARE OPTION VALIDATION
* * DECLARE COMMON AREA AND COMMON AREA PART 2

01 MSGCODE PIC X(04).

01 OUTMSG PIC X(69).

EXEC SQL INCLUDE DSN8MCCS END-EXEC.
EXEC SQL INCLUDE DSN8MCOV END-EXEC.
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC SQL INCLUDE DSN8MCC2 END-EXEC.

LINKAGE SECTION.
01 DFHCOMMAREA.
EXEC SQL INCLUDE DSN8MCCA END-EXEC.

PROCEDURE DIVISION.

* INIT AREA INCLUDED BY DSN8MCCS @001
* MOVE SPACES TO PCONA.
* INIT AREA INCLUDED BY DSN8MCOV @001
* MOVE SPACES TO POPTVAL.

```

```

* SQL RETURN CODE HANDLING *
***** EXEC SQL WHENEVER SQLERROR GO TO DB-ERROR END-EXEC
 EXEC SQL WHENEVER SQLWARNING GO TO DB-ERROR END-EXEC.

MOVE 'DSN8CC1' TO MAJOR IN DSN8-MODULE-NAME.

* FIND VALID OPTIONS FOR ACTION, OBJECT, SEARCH CRITERION*
* RETRIEVE CONVERSATION, VALIDATE, CALL SQL2 *

EXEC SQL INCLUDE DSN8MCV0 END-EXEC.
EXEC SQL INCLUDE DSN8MC1 END-EXEC.

*
* **RETURN
CC1-EXIT.
 EXEC CICS RETURN END-EXEC.

* VALIDATE ACTION, OBJECT, SEARCH CRITERIA *
* HANDLE ERRORS *

CC1-CALL.
 EXEC CICS LINK PROGRAM('DSN8CC2') COMMAREA(DFHCOMMAREA)
 LENGTH(3000) END-EXEC.
 GO TO MC1-SAVE.
 EXEC SQL INCLUDE DSN8MC3 END-EXEC.
 EXEC SQL INCLUDE DSN8MC4 END-EXEC.
 EXEC SQL INCLUDE DSN8MC5 END-EXEC.
 EXEC SQL INCLUDE DSN8MCXX END-EXEC.
 GOBACK.

```

### Referencia relacionada

["Ejemplos de aplicaciones en CICS"](#) en la página 1461

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

## DSN8CC2

ROUTER PARA SELECCIÓN SECUNDARIA Y/O LLAMADAS DE PROCESAMIENTO DE DETALLES SELECCIÓN SECUNDARIA MÓDULOS DSN8MCA DSN8MCM LLAMADAS MÓDULOS DETALLES DSN8MCD DSN8MCE DSN8MCF DSN8MCT DSN8MCV DSN8MCW DSN8MCX DSN8MCZ LLAMADAS POR DSN8MC1 (SQL1).

```

IDENTIFICATION DIVISION.

PROGRAM-ID. DSN8CC2.

***** DSN8CC2 - COMMON MODULE FOR CICS - COBOL*****
*
* MODULE NAME = DSN8CC2
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
* SQL 2 COMMON MODULE
* CICS
* COBOL
*
* Licensed Materials - Property of IBM
*5605-DB2
*(C) COPYRIGHT 1982, 2010 IBM Corp. All Rights Reserved.
*
*STATUS = Version 10
*
*
* FUNCTION = ROUTER FOR SECONDARY SELECTION AND/OR
* DETAIL PROCESSING
* CALLS SECONDARY SELECTION MODULES
* DSN8MCA DSN8MCM
* CALLS DETAIL MODULES
* DSN8MCD DSN8MCE DSN8MCF
* DSN8MCT DSN8MCV DSN8MCW DSN8MCX DSN8MCZ
* CALLED BY DSN8MC1 (SQL1)
*
* NOTES = NONE

```

```

*
*
* MODULE TYPE =
* PROCESSOR = DB2 PRECOMPILER, CICS TRANSLATOR,
* VS COBOL
* MODULE SIZE = SEE LINKEDIT
* ATTRIBUTES = REUSABLE
*
* ENTRY POINT = DSN8CC2
* PURPOSE = SEE FUNCTION
* LINKAGE = NONE
* INPUT =
*
* SYMBOLIC LABEL/NAME = COMMPTR
* DESCRIPTION = POINTER TO COMMAREA
* (COMMUNICATION AREA)
*
* OUTPUT =
*
* SYMBOLIC LABEL/NAME = COMMPTR
* DESCRIPTION = POINTER TO COMMAREA
* (COMMUNICATION AREA)
*
* EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION
*
* EXIT-ERROR =
* IF SQLERROR OR SQLWARNING, SQL WHENEVER CONDITION
* SPECIFIED IN DSN8CC2 WILL BE RAISED AND PROGRAM
* WILL GO TO THE LABEL DB-ERROR.
*
*
* RETURN CODE = NONE
*
*
* ABEND CODES = NONE
*
*
* ERROR-MESSAGES =
* DSN8062E-AN OBJECT WAS NOT SELECTED
* DSN8066E-UNSUPPORTED PFK OR LOGIC ERROR
* DSN8072E-INVALID SELECTION ON SECONDARY SCREEN
*
*
* EXTERNAL REFERENCES = NONE
* ROUTINES/SERVICES = 10 MODULES LISTED ABOVE
* DSN8MCG - ERROR MESSAGE ROUTINE
*
*
* DATA-AREAS =
* DSN8MCA - SECONDARY SELECTION FOR ORGANIZATION
* DSN8MCAD - DECLARE ADMINISTRATION DETAIL
* DSN8MCAE - CURSOR EMPLOYEE LIST
* DSN8MCAL - CURSOR ADMINISTRATION LIST
* DSN8MCA2 - DECLARE ADMINISTRATION DETAIL
* DSN8MCCA - COMMON AREA
* DSN8MCC2 - COMMON AREA PART 2
* DSN8MCD - DEPARTMENT STRUCTURE DETAIL
* DSN8MCDA - CURSOR ADMINISTRATION DETAIL
* DSN8MCDH - CURSOR FOR DISPLAY TEXT FROM
* TDSPXTT TABLE
* DSN8MCDM - DECLARE DEPARTMENT MANAGER
* DSN8MCDP - DECLARE DEPARTMENT
* DSN8MCDT - DECLARE DISPLAY TEXT
* DSN8MCE - DEPARTMENT DETAIL
* DSN8MCEM - DECLARE EMPLOYEE
* DSN8MCED - DECLARE EMPLOYEE-DEPARTMENT
* DSN8MCF - EMPLOYEE DETAIL
* DSN8MCOV - DECLARE OPTION VALIDATION
* DSN8MCXX - ERROR HANDLER
*
*
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA
*
*
* TABLES = NONE
*
*
* CHANGE-ACTIVITY =
* - ADD NEW VARIABLES FOR REFERENTIAL INTEGRITY V2R1 *
* - 10/18/2005 PK03311 INITIALIZE UNINITIALIZED STORAGE @01 *
*
* *PSEUDOCODE*
*
* THIS MODULE DETERMINES WHICH SECONDARY SELECTION AND/OR
* DETAIL MODULE(S) ARE TO BE CALLED IN THE CICS/COBOL
* ENVIRONMENT.
*

```

```

* WHAT HAS HAPPENED SO FAR?.....THE SUBSYSTEM *
* DEPENDENT MODULE (IMS,CICS,TSO) OR (SQL 0) HAS *
* READ THE INPUT SCREEN, FORMATTED THE INPUT AND PASSED CONTROL *
* TO SQL 1. SQL 1 PERFORMS VALIDATION ON THE SYSTEM DEPENDENT *
* FIELDS (MAJOR SYSTEM, ACTION, OBJECT, SEARCH CRITERIA). IF *
* ALL SYSTEM FIELDS ARE VALID SQL 1 PASSED CONTROL TO THIS *
* MODULE. PASSED PARAMETERS CONSIST ONLY OF A POINTER WHICH *
* POINTS TO A COMMUNICATION CONTROL AREA USED TO COMMUNICATE *
* BETWEEN SQL 0 , SQL 1, SQL 2 AND THE SECONDARY SELECTION *
* AND DETAIL MODULES. *
*
*WHAT IS INCLUDED IN THIS MODULE?..... *
* ALL SECONDARY SELECTION AND DETAIL MODULES ARE 'INCLUDED'. *
* ALL VARIABLES KNOWN IN THIS PROCEDURE ARE KNOWN IN THE *
* SUB PROCEDURES. ALL SQL CURSOR DEFINITIONS AND *
* SQL 'INCLUDES' ARE DONE IN THIS PROCEDURE. BECAUSE OF THE *
* RESTRICTION THAT CURSOR HOST VARIABLES MUST BE DECLARED BEFORE*
* THE CURSOR DEFINITION ALL CURSOR HOST VARIABLES ARE DECLARED *
* IN THIS PROCEDURE. *
*
* PROCEDURE *
* IF ANSWER TO DETAIL SCREEN AND DETAIL PROCESSOR *
* IS NOT WILLING TO ACCEPT AN ANSWER THEN *
* NEW REQUEST* *
*
* ELSE *
* IF ANSWER TO A SECONDARY SELECTION THEN *
* DETERMINE IF NEW REQUEST. *
*
*
* CASE (NEW REQUEST) *
*
* SUBCASE ('ADD') *
* DETAIL PROCESSOR *
* RETURN TO SQL 1 *
* ENDSUB *
*
* SUBCASE ('ERASE','DISPLAY','UPDATE') *
* CALL SECONDARY SELECTION *
* IF # OF POSSIBLE CHOICES IS ^= 1 THEN *
* RETURN TO SQL 1 *
* ELSE *
* CALL THE DETAIL PROCESSOR *
* RETURN TO SQL 1 *
* ENDSUB *
*
* ENDCASE *
*
* IF ANSWER TO SECONDARY SELECTION AND A SELECTION HAS *
* ACTUALLY BEEN MADE THEN *
* IF IT IS A VALID SELECTION NUMBER THEN *
* CALL DETAIL PROCESSOR *
* RETURN TO SQL 1 *
* END *
*
* ELSE *
* PRINT ERROR MSG *
* RETURN TO SQL 1 *
* END. *
*
* IF ANSWER TO SECONDARY SELECTION THEN *
* CALL SECONDARY SELECTION *
* RETURN TO SQL 1 *
* END. *
*
* IF ANSWER TO DETAIL THEN *
* CALL DETAIL PROCESSOR *
* RETURN TO SQL 1 *
* END. *
*
* RETURN TO SQL 1. *
*
* END. *
*
* *EXAMPLE- A ROW IS SUCCESSFULLY ADDED, THE OPERATOR RECEIVES*
* THE SUCCESSFULLY ADDED MESSAGE AND JUST HITS ENTER. *
----------*-----*-----*-----*
```

ENVIRONMENT DIVISION.

```

DATA DIVISION.

 WORKING-STORAGE SECTION.

***** * FIELDS SENT TO MESSAGE ROUTINE *
***** 01 MSGCODE PIC X(04).

 01 OUTMSG PIC X(69).

***** * NULL INDICATOR *
***** 01 NULLIND1 PIC S9(4) COMP-4.
 01 NULLIND2 PIC S9(4) COMP-4.
 01 NULLIND3 PIC S9(4) COMP-4.
 01 NULLIND4 PIC S9(4) COMP-4.
 01 NULLIND5 PIC S9(4) COMP-4.
 01 NULLARRY.
 03 NULLARRY1 PIC S9(4) USAGE COMP OCCURS 13 TIMES.

 EXEC SQL INCLUDE SQLCA END-EXEC.

 EXEC SQL INCLUDE DSN8MCC2 END-EXEC.
 EXEC SQL INCLUDE DSN8MCMDP END-EXEC.
 EXEC SQL INCLUDE DSN8MCEM END-EXEC.
 EXEC SQL INCLUDE DSN8MCMDM END-EXEC.
 EXEC SQL INCLUDE DSN8MCAD END-EXEC.
 EXEC SQL INCLUDE DSN8MCA2 END-EXEC.
 EXEC SQL INCLUDE DSN8MCOV END-EXEC.
 EXEC SQL INCLUDE DSN8MCDT END-EXEC.
 EXEC SQL INCLUDE DSN8MCED END-EXEC.

 01 CONSTRAINTS.
 03 PARM-LENGTH PIC S9(4) COMP-4.
 03 REF-CONSTRAINT PIC X(08).
 03 FILLER PIC X(62).
 01 MGRNO-CONSTRAINT PIC X(08) VALUE 'RDE' .

LINKAGE SECTION.
 01 DFHCOMMAREA.
 EXEC SQL INCLUDE DSN8MCMA END-EXEC.

PROCEDURE DIVISION.

 EXEC SQL INCLUDE DSN8MCAE END-EXEC.
 EXEC SQL INCLUDE DSN8MCAL END-EXEC.
 EXEC SQL INCLUDE DSN8MCDF END-EXEC.
 EXEC SQL INCLUDE DSN8MCDA END-EXEC.

***** * SQL RETURN CODE HANDLING *
***** EXEC SQL WHENEVER SQLERROR GO TO DB-ERROR END-EXEC
 EXEC SQL WHENEVER SQLWARNING GO TO DB-ERROR END-EXEC.

***** * INITIALIZATIONS *
 MOVE 'DSN8CC2' TO MAJOR.
 MOVE SPACES TO MINOR.
* INIT AREA INCLUDED BY DSN8MCMDP @001
 MOVE SPACES TO PDEPT.
* INIT AREA INCLUDED BY DSN8MCEM @001
 MOVE SPACES TO PEMP.
* INIT AREA INCLUDED BY DSN8MCDF @001
 MOVE SPACES TO PDEPMGR.
* INIT AREA INCLUDED BY DSN8MCAD @001
 MOVE SPACES TO PASTRDET.
* INIT AREA INCLUDED BY DSN8MCA2 @001
 MOVE SPACES TO PASTRDE2.
* INIT AREA INCLUDED BY DSN8MCDF @001
 MOVE SPACES TO POPTVAL.
* INIT AREA INCLUDED BY DSN8MCDF @001
 MOVE SPACES TO PDSPTXT.
* INIT AREA INCLUDED BY DSN8MCED @001
 MOVE SPACES TO PEMPDPT1.

```

```

* DETERMINES WHETHER NEW REQUEST OR NOT *

* IC200B.

*

IF PREV OF PCONVSTA = ' ' THEN

 MOVE 'Y' TO NEWREQ OF COMPARM.

*

IF NEWREQ OF COMPARM = 'N' AND PREV OF PCONVSTA = 'S'

 AND DATA01 NOT = ''

 AND PFKIN NOT = '08'

 THEN MOVE 'Y' TO NEWREQ OF COMPARM.

*

IF NEWREQ OF COMPARM NOT = 'Y' THEN

 GO TO IC2010.

*

* IF NEW REQUEST AND ACTION IS 'ADD' THEN *

* CALL DETAIL PROCESSOR *

* ELSE CALL SECONDARY SELECTION *

IF ACTION OF INAREA = 'A' THEN

* **DETAILED PROCESSOR

* GO TO DETAIL0.

* **SECONDARY SELECTION

* PERFORM SECSEL THRU END-SECSEL.

* **IF NO. OF CHOICES = 1

* **GO TO DETAIL PROCESSOR

* IF MAXSEL = 1 THEN

* GO TO DETAIL0.

* GO TO EXIT0.

* DETERMINES IF VALID SELECTION NUMBER *

IC2010.

* **VALID SELECTION NO. GIVEN

IF PREV OF PCONVSTA NOT = 'S' OR

 MAXSEL < 1 OR

 PFKIN = '08' OR

 DATA2 = DAT02 THEN

 GO TO IC201.

* **DETAILED SELECTION GIVEN

IF DAT1 NUMERIC AND DAT2 = ' ' THEN

 MOVE DAT1 TO DAT2

 MOVE '0' TO DAT1.

IF DATA2 NUMERIC

 AND DATA2 > '00' AND DATA2 NOT > MAXSEL THEN

 MOVE 'Y' TO NEWREQ OF COMPARM

 GO TO DETAIL0.

* **INVALID SELECTION NO.

* **PRINT ERROR MESSAGE

MOVE '072E' TO MSGCODE.

CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.

MOVE OUTMSG TO MSG OF OUTAREA.

GO TO EXIT0.

* DETERMINES WHETHER SECONDARY SELECTION OR DETAIL *

IC201.

* **SECONDARY SELECTION

IF PREV OF PCONVSTA = 'S' THEN

 PERFORM SECSEL THRU END-SECSEL

 GO TO EXIT0

ELSE

* **DETAILED PROCESSOR

 IF PREV OF PCONVSTA = 'D' THEN GO TO DETAIL0.

* **LOGIC ERROR

* **PRINT ERROR MESSAGE

MOVE '066E' TO MSGCODE.

CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.

MOVE OUTMSG TO MSG OF OUTAREA.

GO TO EXIT0.

* **HANDLES ERRORS

EXEC SQL INCLUDE DSN8MCXX END-EXEC.

GO TO EXIT0.

* CALLS SECONDARY SELECTION AND RETURNS TO SQL 1 *


```

```

SECSEL.
 MOVE 'DSN8001' TO LASTSCR IN PCONVSTA.
*
* **ADMINISTRATIVE
* **DEPARTMENT STRUCTURE
 IF OBJFLD OF INAREA = 'DS' THEN
 PERFORM DSN8MCA THRU END-DSN8MCA
 ELSE
* **INDIVIDUAL DEPARTMENT
* **PROCESSING
 IF OBJFLD OF INAREA = 'DE' THEN
 PERFORM DSN8MCA THRU END-DSN8MCA
 ELSE
* **INDIVIDUAL EMPLOYEE
* **PROCESSING
 IF OBJFLD OF INAREA = 'EM' THEN
 PERFORM DSN8MCA THRU END-DSN8MCA
 ELSE
* **ERROR MESSAGE
* **MISSING SECONDARY SEL
 MOVE '062E' TO MSGCODE
 CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
 MOVE OUTMSG TO MSG OF OUTAREA
 GO TO EXITO.
 END-SECSEL.

* CALLS DETAIL PROCESSOR AND RETURNS TO SQL 1

DETAIL0.
 MOVE 'DSN8002' TO LASTSCR IN PCONVSTA.
*
* **ADMINISTRATIVE
* **DEPARTMENT STRUCTURE
 IF OBJFLD OF INAREA = 'DS' THEN
 PERFORM DSN8MCD THRU END-DSN8MCD
 ELSE
* **INDIVIDUAL DEPARTMENT
* **PROCESSING
 IF OBJFLD OF INAREA = 'DE' THEN
 PERFORM DSN8MCE THRU END-DSN8MCE
 ELSE
* **INDIVIDUAL EMPLOYEE
* **PROCESSING
 IF OBJFLD OF INAREA = 'EM' THEN
 PERFORM DSN8MCF THRU END-DSN8MCF
 ELSE
* **ERROR MESSAGE
* **MISSING DETAIL MODULE
 MOVE '062E' TO MSGCODE
 CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
 MOVE OUTMSG TO MSG OF OUTAREA.
 GO TO EXITO.

* **RETURNS TO SQL 1
EXITO.
 EXEC CICS RETURN END-EXEC.

 EXEC SQL INCLUDE DSN8MCA END-EXEC.
 EXEC SQL INCLUDE DSN8MCD END-EXEC.
 EXEC SQL INCLUDE DSN8MCE END-EXEC.
 EXEC SQL INCLUDE DSN8MCF END-EXEC.
 GOBACK.

```

### Referencia relacionada

[“Ejemplos de aplicaciones en CICS” en la página 1461](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

### DSN8CPO

ESTE MÓDULO EMITE UN CICS MAPA DE RECEPCIÓN PARA RECUPERAR ENTRADAS, LLAMADAS DSN8CP1, Y EMITE UN CICS MAPA DE ENVÍO TRAS EL REGRESO.

DSN8CPO: PROC OPTIONS (MAIN);	00010000
/*****	00020000
*	* 00030000

```

* MODULE NAME = DSN8CP0 * 00040000
* * 00050000
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION * 00060000
* SUBSYSTEM INTERFACE MODULE * 00070000
* CICS * 00080000
* PL/I * 00090000
* ORGANIZATION APPLICATION * 00100000
* * 00110000
* LICENSED MATERIALS - PROPERTY OF IBM 5655-DB2 * 00120000
* (C) COPYRIGHT 1982, 2010 IBM CORP. ALL RIGHTS RESERVED. * 00130000
* * 00140000
* STATUS = VERSION 10 * 00150000
* * 00160000
* FUNCTION = THIS MODULE ISSUES A CICS RECEIVE MAP TO RETRIEVE * 00170000
* INPUT, CALLS DSN8CP1, AND ISSUES A CICS SEND * 00180000
* MAP AFTER RETURNING. * 00190000
* * 00200000
* NOTES = * 00210000
* 1. THIS IS A CICS PSEUDO CONVERSATION PROGRAM WHICH * 00220000
* INITIALIZES ITSELF WHEN TERMINAL OPERATOR ENTERS * 00230000
* INPUT AFTER VIEWING THE SCREEN SENT BY PREVIOUS * 00240000
* ITERATIONS OF THE PROGRAM. * 00250000
* * 00260000
* DEPENDENCIES = TWO CICS MAPS(DSECTS) ARE REQUIRED: * 00270000
* DSN8MCMG AND DSN8MCMD * 00280000
* MODULES DSN8CP1 IS REQUIRED. * 00290000
* DCLGEN STRUCTURE DSN8MPCS IS REQUIRED. * 00300000
* INCLUDED PLI STRUCTURE DSN8MPCA IS REQUIRED. * 00310000
* * 00320000
* RESTRICTIONS = NONE * 00330000
* * 00340000
* * 00350000
* MODULE TYPE = PL/I PROC OPTIONS(MAIN) * 00360000
* PROCESSOR = DB2 PRECOMPLIER, CICS TRANSLATOR, PL/I OPTIMIZE * 00370000
* MODULE SIZE = SEE LINK-EDIT * 00380000
* ATTRIBUTES = REUSABLE * 00390000
* * 00400000
* ENTRY POINT = DSN8CP0 * 00410000
* PURPOSE = SEE FUNCTION * 00420000
* LINKAGE = CICS/OS/VS ENTRY * 00430000
* * 00440000
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION: * 00450000
* SYMBOLIC LABEL/NAME = DSN8CPDI * 00460000
* DESCRIPTION = CICS BMS MAP FOR DETAIL INPUT * 00470000
* * 00480000
* SYMBOLIC LABEL/NAME = DSN8CPGI * 00490000
* DESCRIPTION = CICS BMS MAP FOR GENERAL INPUT * 00500000
* * 00510000
* OUTPUT = PARAMETERS EXPLICITLY RETURNED: * 00520000
* SYMBOLIC LABEL/NAME = DSN8CPDO * 00530000
* DESCRIPTION = CICS BMS MAP FOR DETAIL OUTPUT * 00540000
* * 00550000
* SYMBOLIC LABEL/NAME = DSN8CPGO * 00560000
* DESCRIPTION = CICS BMS MAP FOR GENERAL OUTPUT * 00570000
* * 00580000
* EXIT-NORMAL = CICS RETURN TRANSID(D8PS). * 00590000
* * 00600000
* EXIT-ERROR = DB_ERROR FOR SQL ERRORS. * 00610000
* NO PL/I ON CONDITIONS. * 00620000
* * 00630000
* RETURN CODE = NONE * 00640000
* * 00650000
* ABEND CODES = * 00660000
* CICS ABEND FOR CICS PROBLEMS. * 00670000
* MAPI - LASTSCREEN IS WRONG NAME ON INPUT * 00680000
* MAPO - SQL1 DID NOT PASS BACK VALID LASTSCREEN NAME * 00690000
* * 00700000
* ERROR-MESSAGES = NONE * 00710000
* * 00720000
* EXTERNAL REFERENCES = COMMON CICS REQUIREMENTS * 00730000
* ROUTINES/SERVICES = DSN8CP1 * 00740000
* * 00750000
* DATA-AREAS = * 00760000
* DSN8MPCA = - PARAMETER TO BE PASSED TO DSN8CP1 * 00770000
* COMMON AREA * 00780000
* DSN8MPCS = - DECLARE CONVERSATION STATUS * 00790000
* DSN8MPMD = - CICS/OS/VS PL/I MAP, ORGANIZATION * 00800000
* DSN8MPMG = - CICS/OS/VS PL/I MAP, ORGANIZATION * 00810000
* * 00820000
* CONTROL-BLOCKS = * 00830000
* SQLCA = - SQL COMMUNICATION AREA * 00840000
* * 00850000

```

```

* TABLES = NONE * 00860000
* * 00870000
* CHANGE-ACTIVITY = NONE * 00880000
* * 00890000
* * 00900000
* * 00910000
* * 00920000
* * 00930000
* * 00940000
* PROCEDURE * 00950000
* DECLARATIONS.
* ALLOCATE PLI WORK AREA FOR COMMAREA.
* PUT MODULE NAME 'DSN8CP0' IN AREA USED BY ERROR-HANDLER. * 00960000
* PUT CICS EIBTRMID IN PCONVSTA.CONVID TO BE PASSED TO DSN8CP1 00970000
* RETRIEVE LASTSCR FROM VCONA USING THE CONVID TO DETERMINE * 00980000
* WHICH OF THE TWO BMS MAPS SHOULD BE USED TO MAP IN DATA. * 00990000
* * 01000000
* IF RETRIEVAL IS SUCCESSFUL, THEN DO.
* EXEC CICS RECEIVE MAP ACCORDING TO SPECIFIED LASTSCR.
* IF MAPFAIL CONDITION IS RAISED* THEN DO.
* COMPARM.PFKIN = '00'
* GO TO CP0CP1
* END
* ELSE
* PUT DATA FROM MAP INTO COMPARM **
* ELSE
* IT IS A NEW CONVERSATION, AND NO EXEC CICS
* RECEIVE MAP IS ISSUED.
*
* CP0CP1:
* EXEC CICS LINK PROGRAM('DSN8CP1') COMMAREA(COMMAREA).
* UPON RETURN FROM DSN8CP1, EXEC CICS SEND MAP ACCORDING TO
* THE TYPE SPECIFIED IN PCONVSTA.LASTSCR.
* EXEC CICS RETURN TRANSID(D8PS).
*
* END.
*
* * I.E. LAST CONVERSATION EXISTS, BUT OPERATOR HAD ENTERED
* DATA FROM A CLEARED SCREEN OR HAD ERASED ALL DATA ON
* SCREEN AND PRESSED ENTER.
*
* ** COMPARM.PFKIN = PF KEY ACTUALLY USED I.E. '01' FOR
* PF1...
*
-----/ 01280000
-----/ %PAGE; 01290000
/*-----*/ 01300000
/*-----*/ 01310000
/*-----*/ 01320000
/*-----*/ 01330000
/*-----*/ 01340000
/*-----*/ 01350000
/*-----*/ 01360000
/*-----*/ 01370000
/*-----*/ 01380000
/*-----*/ 01390000
/*-----*/ 01400000
/*-----*/ 01410000
/*-----*/ 01420000
/*-----*/ 01430000
/*-----*/ 01440000
/*-----*/ 01450000
/*-----*/ 01460000
/*-----*/ 01470000
/*-----*/ 01480000
/*-----*/ 01490000
/*-----*/ 01500000
/*-----*/ 01510000
DCL STRING BUILTIN; 01520000
DCL J FIXED BIN; 01530000
DCL SAVE_CONVID CHAR(16); 01540000
DCL (SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1); /* DECLARE CONTROL FLAGS */ 01550000
DCL DSN8MPG EXTERNAL ENTRY; 01560000
DCL MODULE CHAR (07); 01570000
DCL OUTMSG CHAR (69); 01580000
DCL DSN8MPG EXTERNAL ENTRY; 01590000
DCL DSN8MPG EXTERNAL ENTRY; 01600000
DCL DSN8MPG EXTERNAL ENTRY; 01610000
DCL DSN8MPG EXTERNAL ENTRY; 01620000
DCL DSN8MPG EXTERNAL ENTRY; 01630000
DCL DSN8MPG EXTERNAL ENTRY; 01640000
DCL DSN8MPG EXTERNAL ENTRY; 01650000
DCL DSN8MPG EXTERNAL ENTRY; 01660000
DCL DSN8MPG EXTERNAL ENTRY; 01670000

```

```

/* CICS MAP DSN8CPD. */01680000
/****** */01690000
0DCL MAP1PTR PTR,01700000
 MAP2PTR PTR;01710000
 DCL IOAREA AREA(2048);01720000
0DCL 1 SUBMAP(15) BASED (ADDR(DSN8CPDI.LINE1F1L)) UNALIGNED,01730000
 2 COL1LEN FIXED BIN (15,0) ,01740000
 2 COL1ATTR CHAR (1) ,01750000
 2 COL1DATA CHAR (37) ,01760000
 2 COL2LEN FIXED BIN (15,0) ,01770000
 2 COL2ATTR CHAR (1) ,01780000
 2 COL2DATA CHAR (40) ;01790000
0/***** */01800000
/* PFSTRG IS AN ARRAY OF 24 ELEMENTS REPRESENTING THE DIFFERENT */01810000
/* PFKEYS AS THEY WOULD BE REPRESENTED IN EIBAID. */01820000
/****** */01830000
0DCL CONVID CHAR(16) ;01840000
 DCL PFSTRG CHAR(24) INIT ('123456789:#@ABCDEFGHI.<') ,01850000
0/***** */01860000
/* PFK IS AN ARRAY OF 12 TWO-BYTE CHARS REPRESENTING THE PFKEYS */01870000
/* ALLOWED AS INPUT TO DSN8CP1 AND DSN8CP2 ETC. */01880000
/****** */01890000
 PFK(12) CHAR(2) INIT ('01','02','03','04','05','06',01900000
 '07','08','09','10','11','12'),01910000
 N FIXED BIN;01920000
 01930000
 01940000
0/***** */01950000
/* SQL RETURN CODE HANDLING */01960000
/****** */01970000
EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR;01980000
EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR;01990000
 02000000
0/***** */02010000
/* ALLOCATE PL/I WORK AREA / INITIALIZE VARIABLES */02020000
/****** */02030000
 02040000
0ALLOCATE COMMAREA SET(COMMPTR); /*ALLOCATE COMMON AREA */02050000
MAP1PTR = ADDR(IOAREA); /* SET THE POINTER FOR THE GENERAL MAP */02060000
MAP2PTR = ADDR(IOAREA); /* SET THE POINTER FOR THE DETAIL MAP */02070000
COMMAREA = '' ; /*CLEAR COMMON AREA */02080000
DSN8_MODULE_NAME.MAJOR = 'DSN8CP0 ' ; /*GET MODULE NAME */02090000
 02100000
 /*CONSTRUCT CICS CONVERSATION ID */
 /*A 4 CHAR TERMINAL ID CON-*/
 /*CATENATED WITH 12 BLANKS*/02110000
 02120000
 02130000
CONVID,PCONVSTA.CONVID = EIBTRMID || ' ' ;02140000
OUTAREA.MAJSYS = '0'; /*SET MAJOR SYSTEM TO 0-ORGANIZATION*/02150000
EXITCODE = '0'; /*CLEAR EXIT CODE */02160000
 02170000
EXEC CICS HANDLE CONDITION MAPFAIL(CP0SEND);02180000
 02190000
0/***** */02200000
/* TRY TO RETRIEVE LAST CONVERSATION. IF SUCCESSFUL, USE THE */02210000
/* LAST SCREEN SPECIFIED TO RECEIVE INPUT FROM TERMINAL. */02220000
/****** */02230000
 02240000
0 EXEC SQL SELECT LASTSCR02250000
 INTO :PCONA.LASTSCR02260000
 FROM VCONA02270000
 WHERE CONVID = :CONVID ;02280000
 02290000
0/***** */02300000
/* IF LAST CONVERSATION DOES NOT EXIST, THEN DO NOT ATTEMPT TO */02310000
/* RECEIVE INPUT MAP. GO DIRECTLY TO VALIDATION MODULES */02320000
/* TO GET TITLE ETC. FOR OUTPUT MAP. */02330000
/****** */02340000
 02350000
0 IF SQLCODE = +100 THEN GO TO CP0SEND;02360000
 02370000
0/***** */02380000
/* IF DATA IS RECEIVED FOR A FIELD, THENMOVE THE DATA */02390000
/* INTO THE CORRESPONDING FIELDS IN INAREA, OTHERWISE MOVE BLANKS. */02400000
/* */02410000
/* IF LAST CONVERSATION EXISTS, BUT OPERATOR HAS ENTERED DATA */02420000
/* FROM A CLEARED SCREEN OR HAD ERASED ALL DATA ON A FORMATTED */02430000
/* SCREEN AND PRESSED ENTER THEN */02440000
/* MOVE DATA INTO CORRESPONDING FIELDS IN INAREA AND GO TO */02450000
/* VALIDATION MODULES. */02460000
/****** */02470000
 02480000
IF PCONA.LASTSCR = 'DSN8001 ' THEN02490000

```

```

DO; /*USING LAST SCREEN */ 02500000
 /*SPECIFIED TO RECEIVE*/ 02510000
 /*INPUT FROM TERMINAL*/ 02520000
EXEC CICS RECEIVE MAP ('DSN8CPG') MAPSET ('DSN8CPG') ; 02530000
 02540000
 02550000
IF AMAJSYSL ^= 0 THEN COMPARM.MAJSYS = AMAJSYSI; 02560000
ELSE COMPARM.MAJSYS = '0'; 02570000
IF AACTIONL ^= 0 THEN COMPARM.ACTION = AACTIONI; 02580000
ELSE COMPARM.ACTION = ' '; 02590000
IF AOBJECTL ^= 0 THEN COMPARM.OBJFLD = AOBJECTI; 02600000
ELSE COMPARM.OBJFLD = ' '; 02610000
IF ASEARCHL ^= 0 THEN COMPARM.SEARCH = ASEARCHI; 02620000
ELSE COMPARM.SEARCH = ' '; 02630000
IF ADATAL ^= 0 THEN COMPARM.DATA = ADATAI ; 02640000
ELSE COMPARM.DATA = ' '; 02650000
 02660000
 02670000
0 ELSE IF PCONA.LASTSCR = 'DSN8002 ' THEN 02680000
DO; /*MOVE DATA INTO */ 02690000
 /*INPUT FIELDS */ 02700000
EXEC CICS RECEIVE MAP ('DSN8CPD') MAPSET('DSN8CPD') ; 02710000
 02720000
 02730000
IF BMAJSYSL ^= 0 THEN COMPARM.MAJSYS = BMAJSYSI; 02740000
ELSE COMPARM.MAJSYS = '0'; 02750000
IF BACTIONL ^= 0 THEN COMPARM.ACTION = BACTIONI; 02760000
ELSE COMPARM.ACTION = ' '; 02770000
IF BOBJECTL ^= 0 THEN COMPARM.OBJFLD = BOBJECTI; 02780000
ELSE COMPARM.OBJFLD = ' '; 02790000
IF BSEARCHL ^= 0 THEN COMPARM.SEARCH = BSEARCHI; 02800000
ELSE COMPARM.SEARCH = ' '; 02810000
IF BDATAL ^= 0 THEN COMPARM.DATA = BDATAI ; 02820000
ELSE COMPARM.DATA = ' '; 02830000
 02840000
DO I = 1 TO 15; 02850000
IF SUBMAP.COL2LEN(I) ^= 0 THEN
 COMPARM.TRANDATA(I) = SUBMAP.COL2DATA(I) ; 02860000
ELSE COMPARM.TRANDATA(I) = ' ' ; 02870000
 02880000
END; 02890000
 02900000
 02910000
0 ELSE /* WRONG LASTSCREEN NAME*/ 02920000
DO;
 EXEC CICS ABEND ABCODE ('MAPI'); 02930000
END; 02940000
 02950000
 02960000
0*****/*CONVERT THE PFKEY INFO IN EIBAID TO THE FORM ACCEPTED */02970000
/* BY DSN8CP1 AND DSN8CP2 ETC. EG. PF1 = '01' AND PF13 = '01'. */02980000
/******0299000003000000030100000302000003030000030400000305000003060000030700000308000003090000031000000311000003120000031300000314000003150000031600000317000003180000031900000320000003210000032200000323000003240000032500000326000003270000032800000329000003300000033100000
CP0CP1 : 03220000
INAREA.MAJSYS = '0'; /*SET MAJOR SYSTEM TO O-ORGANIZATION */ 03230000
 03240000
 03250000
 03260000
 03270000
 03280000
0 EXEC SQL INCLUDE DSN8MPXX; /*GET DCLGEN STRUCTURES*/ 03290000
 03300000
 03310000
*****03310000

```

```

/*
/* AFTER RETURN FROM DSN8CP1 (SQL1), THE PROGRAM EXAMINES DATA */03320000
/* PASSED BACK IN PCONVSTA TO SEE WHAT KIND OF SCREEN SHOULD BE */03330000
/* SENT. PUT THAT DATA INTO THE OUTPUT MAP AND SEND OUTPUT. */03340000
/* IF A SQL ERROR OR WARNING HAD OCCURRED PREVIOUSLY, THE ERROR */03350000
/* MESSAGES ARE EXPECTED TO HAVE BEEN PUT INTO PCONVSTA. */03360000
*/
/*MESSAGES ARE EXPECTED TO HAVE BEEN PUT INTO PCONVSTA. */03370000
*/
/******03380000
*****03390000
03400000
IF PCONVSTA.LASTSCR = 'DSN8001' THEN /*MOVE DATA INTO */03410000
DO; /*OUTPUT FIELDS */03420000
 ATITLEO = PCONVSTA.TITLE ;03430000
 AMAJSYSO= PCONVSTA.MAJSYS;03440000
 AACTIONO= PCONVSTA.ACTION;03450000
 AOBJECTO= PCONVSTA.OBJFLD;03460000
 ASEARCHO= PCONVSTA.SEARCH;03470000
 ADATAAO = PCONVSTA.DATA ;03480000
 AMSGO = PCONVSTA.MSG ;03490000
 ADESCL20= PCONVSTA.DESC2 ;03500000
 ADESCL30= PCONVSTA.DESC3 ;03510000
 ADESCL40= PCONVSTA.DESC4 ;03520000
 APFKEYO = PCONVSTA.PFKTEXT;03530000
 03540000
 DO I = 1 TO 15; /*SEND MAP ACCORDING TO */03550000
 ALINEO(I) = PCONVSTA.OUTPUT.LINE(I); /*PREVIOUS SCREEN*/03560000
 END;03570000
 03580000
0/*****03590000
/* CREATES A DYNAMIC CURSOR */03600000
/*****03610000
 /SET CURSOR POSITION */03620000
 CURSOR_VALUE = 0; /*CLEAR CURSOR*/03630000
 IF AACTIONO = ' ' THEN /*CURSOR SET TO*/03640000
 CURSOR_VALUE = 179; /*ACTION POSITION*/03650000
 ELSE03660000
 IF AOBJECTO = ' ' THEN /*CURSOR SET TO*/03670000
 CURSOR_VALUE = 259; /*OBJFLD POSITION*/03680000
 ELSE03690000
 IF ASEARCHO = ' ' THEN /*CURSOR SET TO*/03700000
 CURSOR_VALUE = 339; /*SEARCH CRITERIA POSITION*/03710000
 ELSE03720000
 IF ADATAAO = ' ' |03730000
 (AACTIONO = 'D' |03740000
 AACTIONO = 'E') THEN /*CURSOR SET TO */03750000
 CURSOR_VALUE = 419; /*DATA POSITION*/03760000
 03770000
 IF CURSOR_VALUE = 0 THEN /*SEND OUTPUT MAP */03780000
 EXEC CICS SEND MAP('DSN8CPG') MAPSET('DSN8CPG') ERASE;03790000
 ELSE03800000
 EXEC CICS SEND MAP('DSN8CPG') MAPSET('DSN8CPG') ERASE03810000
 CURSOR(CURSOR_VALUE);03820000
 03830000
 IF EXITCODE = '1' THEN /* FINISHED ? */03840000
 EXEC CICS RETURN ; /* EXIT, DON'T REINVOKE TRANSACTION */03850000
 ELSE03860000
 EXEC CICS RETURN TRANSID('D8PS'); /* STANDARD EXIT */03870000
 END;03880000
 03890000
0/*****03900000
/* MOVES DATA FROM OUTPUT MAP AREA TO */03910000
/* RECEIVE MAP ACCORDING TO MAP SPECIFIED IN LASTSCR OF PCONVST */03920000
/*****03930000
0 ELSE IF PCONVSTA.LASTSCR = 'DSN8002' THEN03940000
DO;03950000
 BTITLEO = PCONVSTA.TITLE ;03960000
 BMMAJSYSO= PCONVSTA.MAJSYS;03970000
 BACTIONO= PCONVSTA.ACTION;04000000
 BOBJECTO= PCONVSTA.OBJFLD;04010000
 BSEARCHO= PCONVSTA.SEARCH;04020000
 BDATAAO = PCONVSTA.DATA ;04030000
 BMSGO = PCONVSTA.MSG ;04040000
 BDDESCL20= PCONVSTA.DESC2 ;04050000
 BDDESCL30= PCONVSTA.DESC3 ;04060000
 BDDESCL40= PCONVSTA.DESC4 ;04070000
 BPFKEYO = PCONVSTA.PFKTEXT;04080000
 04090000
 DO I = 1 TO 15 ; /*RECEIVE MAP ACCORDING*/04100000
 SUBMAP.COL1DATA(I) = REOUT.FIELD1(I); /*TO PREVIOUS SCREEN*/04110000
0 /-----*/04120000
/* */04130000

```

```

/*
/* MODULES DSN8MPE, DSN8MPF ETC. IN SQL2 HAVE PUT THE */ 04140000
/* ATTRIBUTE BYTE AND CURSOR CONTROL INFO IN IMS MFS */ 04150000
/* FORM - HEX'C0' FOR DYNAMIC CURSOR WITH 2 BYTES OF */ 04160000
/* ATTRIBUTE INFORMATION TO FOLLOW. THIS PROGRAM CHECKS */ 04170000
/* FOR THE HEX'C0' AND INSERTS '-1' INTO */ 04180000
/* THE LENGTH FIELD ASSOCIATED WITH THE DATA TO CONFORM */ 04190000
/* WITH THE STANDARD WAY OF HANDLING DYNAMIC CURSORS IN */ 04200000
/* CICS. SIMILARLY, ONLY THE SECOND OF THE TWO ATTRIBUTE */ 04210000
/* BYTES IS MOVED INTO THE CICS ATTRIBUTE BYTE. THE */ 04220000
/* FIRST TWO BITS OF THE ATTRIBUTE BYTE IS DIFFERENT */ 04230000
/* BETWEEN IMS AND CICS STANDARD REPRESENTATIONS, HOWEVER */ 04240000
/* 3270 MANUALS INDICATE THAT ON OUTPUT, THE FIRST */ 04250000
/* TWO BITS ARE IGNORED. THUS THE SAME ATTRIBUTE BYTE */ 04260000
/* IS USED BETWEEN IMS AND CICS MODULES. */ 04270000
*/
*/
0 IF UNSPEC(REOUT.ATTR1(I)) = '11000000'B /* X'C0' ATTR */ 04300000
 THEN SUBMAP.COL2LEN(I) = -1; 04310000
 SUBMAP.COL2ATTR(I) = REOUT.ATTR2(I); 04320000
 SUBMAP.COL2DATA(I) = REOUT.FIELD2(I); 04330000
END; 04340000
0*****/* *****/04350000
/* CREATES A DYNAMIC CURSOR */04360000
0*****/* *****/04370000
 /*SET CURSOR POSITION */ 04380000
 CURSOR_VALUE = 0; /*CLEAR CURSOR */ 04390000
 IF BACTIONO = ' ' THEN /*CURSOR SET TO*/ 04400000
 CURSOR_VALUE = 179; /*ACTION POSITION*/ 04410000
 ELSE 04420000
 IF BOBJECTO = ' ' THEN /*CURSOR SET TO*/ 04430000
 CURSOR_VALUE = 259; /*OBJFLD POSITION*/ 04440000
 ELSE 04450000
 IF BSEARCHO = ' ' THEN /*CURSOR SET TO*/ 04460000
 CURSOR_VALUE = 339; /*SEARCH CRITERIA*/ 04470000
 ELSE 04480000
 IF BDATAO = ' ' | 04490000
 (BACTIONO = 'D' | 04500000
 BACTIONO = 'E') THEN /*CURSOR SET TO */ 04510000
 CURSOR_VALUE = 419; /*DATA POSITION*/ 04520000
 END; 04530000
 IF CURSOR_VALUE = 0 THEN /*SEND INPUT MAP */ 04540000
 EXEC CICS SEND MAP('DSN8CPD') MAPSET('DSN8CPD') ERASE; 04550000
 ELSE 04560000
 EXEC CICS SEND MAP('DSN8CPD') MAPSET('DSN8CPD') ERASE 04570000
 CURSOR(CURSOR_VALUE); 04580000
 04590000
 IF EXITCODE = '1' THEN /*FINISHED ? */ 04600000
 EXEC CICS RETURN ; /* EXIT, DON'T REINVOKE TRANSACTION */ 04610000
 ELSE 04620000
 EXEC CICS RETURN TRANSID('D8PS'); /* STANDARD EXIT */ 04630000
 END; 04640000
0 /* SQL1 DID NOT PASS BACK VALID LASTSCREEN NAME */ 04650000
 ELSE EXEC CICS ABEND ABCODE ('MAPO'); 04660000
END; 04670000

```

### Referencia relacionada

[“Ejemplos de aplicaciones en CICS” en la página 1461](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

## DSN8CP1

ESTE MÓDULO REALIZA LOS INCLUYE PARA INTRODUCIR LAS ESTRUCTURAS DCLS Y DCLGEN DE LA TABLA SQL, ASÍ COMO EL ÁREA DE PARÁMETROS.

```

DSN8CP1:PROC (COMMTR) OPTIONS(MAIN);
*****/* *****/
* MODULE NAME = DSN8CP1 *
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION *
* SQL 1 MAINLINE *
* CICS *
* PL/I *
* ORGANIZATION APPLICATION *
* COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1985 *
* REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083 *
* *

```

```

* STATUS = RELEASE 2, LEVEL 0 *
*
* FUNCTION = THIS MODULE PERFORMS THE INCLUDES TO BRING IN THE *
* SQL TABLE DCLS AND DCLGEN STRUCURES AS WELL AS *
* PARAMETER AREA. *
* INCLUDE DSN8MP1. *
* CALL DSN8CP2. *
* RETURN TO DSN8CP0. *
*
* NOTES = *
* DEPENDENCIES = CALLED BY DSN8CP0, CALLS DSN8CP2 (CICS LINKS). *
* RESTRICTIONS = NONE *
*
* MODULE TYPE = PL/I PROC(COMMTPTR) OPTIONS. *
* PROCESSOR = DB2 PRECOMPILER, CICS TRANSLATOR, PL/I OPTIMIZER *
* MODULE SIZE = SEE LINK-EDIT *
* ATTRIBUTES = REUSABLE *
*
* ENTRY POINT = DSN8CP1 *
* PURPOSE = SEE FUNCTION *
* LINKAGE = NONE *
*
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION: *
* SYMBOLIC LABEL/NAME = COMMTPTR (POINTER TO COMMAREA) *
* DESCRIPTION = NONE *
*
* OUTPUT = PARAMETERS EXPLICITLY RETURNED: *
* SYMBOLIC LABEL/NAME = NONE *
* DESCRIPTION = NONE *
*
* EXIT-NORMAL = DSN8CP0 *
*
* EXIT-ERROR = DSN8CP0 *
*
* RETURN CODE = NONE *
*
* ABEND CODES = NONE *
*
* ERROR-MESSAGES = NONE *
*
* EXTERNAL REFERENCES = *
* ROUTINES/SERVICES = DSN8CP2 *
*
* DATA-AREAS = *
* DSN8MPCA - PLI STRUCTURE FOR COMMAREA *
* DSN8MPCS - DECLARE CONVERSATION STATUS *
* DSN8MPOV - DECLARE OPTION VALIDATION *
* DSN8MPVO - FIND VALID OPTIONS FOR ACTION, *
* OBJECT, SEARCH CRITERIA *
* DSN8MP1 - RETRIEVE LAST CONVERSATION, *
* VALIDATE, CALL SQL2 *
* DSN8MP3 -- DSN8MP5 - VALIDATION MODULES CALLED BY DSN8MP1 *
* DSN8MPXX - SQL ERROR HANDLER *
*
* CONTROL-BLOCKS = *
* SQLCA - SQL COMMUNICATION AREA *
*
* TABLES = NONE *
*
* CHANGE-ACTIVITY = NONE *
*
* *PSEUDOCODE*
*
* PROCEDURE *
* INCLUDE DECLARATIONS. *
* INCLUDE DSN8MP1. *
* INCLUDE ERROR HANDLER. *
*
* CP1EXIT: (REFERENCED BY DSN8MP1) *
* EXEC CICS RETURN. *
*
* CP1CALL: (REFERENCED BY DSN8MP1) *
* EXEC CICS LINK PROGRAM('DSN8CP2') COMMAREA(COMMAREA) *
* LENGTH(3000). *
* GO TO MP1SAVE. (LABEL IN DSN8MP1) *
*
* INCLUDE VALIDATION MODULES. *
*
* END. *

```

```

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*

** DCLGENS AND INITIALIZATIONS

*/
DCL STRING BUILTIN;
DCL J FIXED BIN;
DCL SAVE_CONVID CHAR(16);
DCL (SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1);
DCL MODULE CHAR (07);
DCL OUTMSG CHAR (69);
DCL DSN8MPG EXTERNAL ENTRY;
EXEC SQL INCLUDE DSN8MPCA; /* INCLUDE COMMAREA */
DSN8_MODULE_NAME.MAJOR = 'DSN8CP1 ' ; /* INITIALIZE MODULE NAME */
EXEC SQL INCLUDE DSN8MPCS; /* INCLUDE PCONA */
EXEC SQL INCLUDE DSN8MPOV; /* INCLUDE POPTVAL */
EXEC SQL INCLUDE DSN8MPVO; /* INCLUDE CURSOR */
EXEC SQL INCLUDE SQLCA; /* INCLUDE SQL COMMAREA */
EXEC SQL INCLUDE DSN8MP1; /* INCLUDE SQL1 MAIN */
EXEC SQL INCLUDE DSN8MPXX; /* INCLUDE ERRORHANDLER */

CP1EXIT :
 EXEC CICS RETURN; /* STANDARD EXIT */

CP1CALL :
 EXEC CICS LINK PROGRAM('DSN8CP2') COMMAREA(COMMAREA) LENGTH(3000);
 GO TO MP1SAVE;

EXEC SQL INCLUDE DSN8MP3; /* INCLUDE ACTION VALIDATION */
EXEC SQL INCLUDE DSN8MP4; /* INCLUDE OBJECT VALIDATION */
EXEC SQL INCLUDE DSN8MP5; /* INCLUDE SEARCH CRITERIA */
END; /* VALIDATION */

```

### **Referencia relacionada**

“Ejemplos de aplicaciones en CICS” en la página 1461

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

DSN8CP2

ROUTER PARA SELECCIÓN SECUNDARIA Y/O LLAMADAS DE PROCESAMIENTO DE DETALLES SELECCIÓN SECUNDARIA MÓDULOS DSN8MPA DSN8MPM LLAMADAS MÓDULOS DETALLES DSN8MPD DSN8MPE DSN8MPF DSN8MPT DSN8MPV DSN8MPW DSN8MPX DSN8MPZ LLAMADAS POR DSN8MP1 (SQL1).

```
DSN8CP2: PROC(COMMPTR) OPTIONS(MAIN); /* SQL 2 FOR CICS AND PLI */ 00010000
 %PAGE; 00020000
***** 00030000
* 00040000
MODULE NAME = DSN8CP2 00050000
* 00060000
DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION 00070000
SQL 2 COMMON MODULE 00080000
CICS 00090000
PL/I 00100000
```

```

* ORGANIZATION APPLICATION * 00110000
*
* LICENSED MATERIALS - PROPERTY OF IBM * 00120000
* 5695-DB2 * 00130000
* (C) COPYRIGHT 1982, 1995 IBM CORP. ALL RIGHTS RESERVED. * 00136000
*
* STATUS = VERSION 4 * 00143000
* * 00150000
* * 00160000
* * 00170000
*
* FUNCTION = ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSING 00180000
* CALLS SECONDARY SELECTION MODULES * 00190000
* DSN8MPA DSN8MPM * 00200000
* CALLS DETAIL MODULES * 00210000
* DSN8MPD DSN8MPE DSN8MPF * 00220000
* DSN8MPT DSN8MPV DSN8MPW DSN8MPX DSN8MPZ * 00230000
* CALLED BY DSN8MP1 (SQL1) * 00240000
* * 00250000
* NOTES = NONE * 00260000
* * 00270000
* MODULE TYPE = BLOCK OF PL/I CODE * 00280000
* PROCESSOR = DB2 PRECOMPLIER, PL/I OPTIMIZER * 00290000
* MODULE SIZE = SEE LINKEDIT * 00300000
* ATTRIBUTES = REUSABLE * 00310000
* * 00320000
* ENTRY POINT = DSN8CP2 * 00330000
* PURPOSE = SEE FUNCTION * 00340000
* LINKAGE = NONE * 00350000
* INPUT = * 00360000
* * 00370000
* SYMBOLIC LABEL/NAME = COMMPTR * 00380000
* DESCRIPTION = POINTER TO COMMAREA * 00390000
* (COMMUNICATION AREA) * 00400000
* * 00410000
* OUTPUT = * 00420000
* * 00430000
* SYMBOLIC LABEL/NAME = COMMPTR * 00440000
* DESCRIPTION = POINTER TO COMMAREA * 00450000
* (COMMUNICATION AREA) * 00460000
* * 00470000
* EXIT-NORMAL =
* EXIT-ERROR = IF SQLERROR OR SQLWARNING, SQL WHENEVER CONDITION * 00480000
* SPECIFIED IN DSN8CP2 WILL BE RAISED AND PROGRAM * 00490000
* WILL GO TO THE LABEL DB_ERROR. * 00500000
* * 00510000
* * 00520000
* * 00530000
* * 00540000
* RETURN CODE = NONE * 00550000
* ABEND CODES = NONE * 00560000
* ERROR-MESSAGES =
* DSN8062E-AN OBJECT WAS NOT SELECTED * 00570000
* DSN8066E-UNSUPPORTED PFK OR LOGIC ERROR * 00580000
* DSN8072E-INVALID SELECTION ON SECONDARY SCREEN * 00590000
* * 00600000
* * 00620000
* * 00630000
* * 00640000
* EXTERNAL REFERENCES = NONE * 00650000
* ROUTINES/SERVICES = 10 MODULES LISTED ABOVE * 00660000
* DSN8MPG - ERROR MESSAGE ROUTINE * 00670000
* * 00680000
* DATA-AREAS =
* DSN8MPA - SECONDARY SELECTION FOR ORGANIZATION * 00690000
* DSN8MPAD - DECLARE ADMINISTRATIVE DETAIL * 00700000
* DSN8MPAE - CURSOR EMPLOYEE LIST * 00710000
* DSN8MPAL - CURSOR ADMINISTRATION LIST * 00720000
* DSN8MPA2 - DECLARE ADMINISTRATIVE DETAIL * 00730000
* DSN8MPCA - DECLARE SQL COMMON AREA * 00740000
* DSN8MPD - DEPARTMENT STRUCTURE DETAIL * 00750000
* DSN8MPDA - CURSOR ADMINISTRATION LIST * 00760000
* DSN8MPDH - CURSOR FOR DISPLAY TEXT FROM * 00770000
* TDSPTXT TABLE * 00780000
* DSN8MPDM - DECLARE DEPARTMENT MANAGER * 00790000
* DSN8MPDP - DECLARE DEPARTMENT * 00800000
* DSN8MPDT - DECLARE DISPLAY TEXT * 00810000
* DSN8MPE - DEPARTMENT DETAIL * 00820000
* DSN8MPEM - DECLARE EMPLOYEE * 00830000
* DSN8MPED - DECLARE EMPLOYEE-DEPARTMENT * 00840000
* DSN8MPF - EMPLOYEE DETAIL * 00850000
* DSN8MPOV - DECLARE OPTION VALIDATION * 00860000
* DSN8MPXX - ERROR HANDLER * 00870000
* * 00880000
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA * 00890000
* * 00900000
* * 00910000

```

```

* TABLES = NONE * 00920000
* * 00930000
* CHANGE-ACTIVITY = NONE * 00940000
* * 00950000
* * 00960000
* * 00970000
* * 00980000
* * 00990000
* * 01000000
* * 01010000
* * 01020000
* * 01030000
* * 01040000
* * 01050000
* * 01060000
* * 01070000
* * 01080000
* * 01090000
* * 01100000
* * 01110000
* * 01120000
* * 01130000
* * 01140000
* * 01150000
* * 01160000
* * 01170000
* * 01180000
* * 01190000
* * 01200000
* * 01210000
* * 01220000
* * 01230000
* * 01240000
* * 01250000
* * 01260000
* * 01270000
* * 01280000
* * 01290000
* * 01300000
* * 01310000
* * 01320000
* * 01330000
* * 01340000
* * 01350000
* * 01360000
* * 01370000
* * 01380000
* * 01390000
* * 01400000
* * 01410000
* * 01420000
* * 01430000
* * 01440000
* * 01450000
* * 01460000
* * 01470000
* * 01480000
* * 01490000
* * 01500000
* * 01510000
* * 01520000
* * 01530000
* * 01540000
* * 01550000
* * 01560000
* * 01570000
* * 01580000
* * 01590000
* * 01600000
* * 01610000
* * 01620000
* * 01630000
* * 01640000
* * 01650000
* * 01660000
* * 01670000
* * 01680000
* * 01690000
* * 01700000
* * 01710000
* * 01720000
* * 01730000
-----*/
```

```

/* INCLUDE DECLARES */ 01740000
EXEC SQL INCLUDE DSN8MPCA; /*COMMUNICATION AREA BETWEEN MODULES */ 01750000
EXEC SQL INCLUDE SQLCA; /*SQL COMMUNICATION AREA */ 01760000
EXEC SQL INCLUDE DSN8MPDP; /* DCLGEN FOR DEPARTMENT */ 01770000
EXEC SQL INCLUDE DSN8MPEM; /* DCLGEN FOR EMPLOYEE */ 01780000
EXEC SQL INCLUDE DSN8MPED; /* DCLGEN FOR EMPLOYEE-DEPARTMENT */ 01790000
EXEC SQL INCLUDE DSN8MPDM; /* DCLGEN FOR DEPARTMENT/MANAGER */ 01800000
EXEC SQL INCLUDE DSN8MPAD; /* DCLGEN FOR ADMINISTRATION DETAIL */ 01810000
EXEC SQL INCLUDE DSN8MPA2; /* DCLGEN FOR ADMINISTRATION DETAIL */ 01815000
EXEC SQL INCLUDE DSN8MPOV; /* DCLGEN FOR OPTION VALIDATION */ 01820000
EXEC SQL INCLUDE DSN8MPDT; /* DCLGEN FOR DISPLAY TEXT TABLE */ 01830000
EXEC SQL INCLUDE DSN8MPAL; /* MAJSYS O - SEC SEL FOR DS AND DE */ 01840000
EXEC SQL INCLUDE DSN8MPAE; /* MAJSYS O - SEC SEL FOR EM */ 01850000
EXEC SQL INCLUDE DSN8MPDA; /* MAJSYS O - DETAIL FOR DS */ 01860000
EXEC SQL INCLUDE DSN8MPDH; /* PROG TABLES - DISPLAY HEADINGS */ 01870000
/* CURSORS */ 01880000
DCL VERIFY BUILTIN; 01890000
DCL UNSPEC BUILTIN; 01900000
DCL DSN8MPG EXTERNAL ENTRY; 01910000
DCL STRING BUILTIN; 01920000
DCL J FIXED BIN; 01930000
DCL SAVE_CONVID CHAR(16); 01940000
DCL (SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1); 01950000
DCL /* FIELDS SENT TO MESSAGE ROUTINE */ 01960000
DCL /* DCLGENS AND INITIALIZATIONS */ 01970000
DCL /* PROGRAMMING TABLES */ 01980000
DCL /* STRING BUILTIN */ 01990000
DCL /* DCLGENS AND INITIALIZATIONS */ 02000000
DCL /* PROGRAMMING TABLES */ 02010000
DCL /* DCLGENS AND INITIALIZATIONS */ 02020000
DCL STRING BUILTIN; 02030000
DCL J FIXED BIN; 02040000
DCL SAVE_CONVID CHAR(16); 02050000
DCL (SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1); 02060000
DCL /* FIELDS SENT TO MESSAGE ROUTINE */ 02070000
DCL /* DCLGENS AND INITIALIZATIONS */ 02080000
DCL /* PROGRAMMING TABLES */ 02090000
DCL /* FIELDS SENT TO MESSAGE ROUTINE */ 02100000
DCL /* DCLGENS AND INITIALIZATIONS */ 02110000
DCL /* PROGRAMMING TABLES */ 02120000
DCL MODULE CHAR (07) INIT('DSN8CP2'); 02130000
DCL OUTMSG CHAR (69); 02140000
DCL /* SQL RETURN CODE HANDLING */ 02150000
DCL /* SQL RETURN CODE HANDLING */ 02160000
DCL /* SQL RETURN CODE HANDLING */ 02170000
DCL /* SQL RETURN CODE HANDLING */ 02180000
DCL /* SQL RETURN CODE HANDLING */ 02190000
EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR; 02200000
EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR; 02210000
DCL /* INITIALIZATIONS */ 02220000
DCL /* INITIALIZATIONS */ 02230000
DCL /* INITIALIZATIONS */ 02240000
DCL /* INITIALIZATIONS */ 02250000
DCL /* INITIALIZATIONS */ 02260000
DSN8_MODULE_NAME.MAJOR='DSN8CP2'; 02270000
DSN8_MODULE_NAME.MINOR=' ' ; 02280000
DCL /* DETERMINES WHETHER NEW REQUEST OR NOT */ 02290000
DCL /* DETERMINES WHETHER NEW REQUEST OR NOT */ 02300000
DCL /* DETERMINES WHETHER NEW REQUEST OR NOT */ 02310000
DCL /* DETERMINES WHETHER NEW REQUEST OR NOT */ 02320000
DCL /* IF 'NO ANSWER POSSIBLE' SET BY DETAIL PROCESSOR THEN FORCE A */ 02330000
DCL /* NEW REQUEST. */ 02340000
DCL /* NEW REQUEST. */ 02350000
DCL /* NEW REQUEST. */ 02360000
IF PCONVSTA.PREV = ' ' THEN 02370000
 COMPARM.NEWREQ = 'Y';
DCL /* IF ANSWER TO SECONDARY SELECTION THEN DETERMINE IF REALLY A */ 02380000
DCL /* NEW REQUEST. IT WILL BE CONSIDERED A NEW REQUEST IF POSITIONS*/ 02390000
DCL /* 3 TO 60 ARE NOT ALL BLANK AND THE ENTERED DATA IF NOT 'NEXT' */ 02400000
DCL /* 3 TO 60 ARE NOT ALL BLANK AND THE ENTERED DATA IF NOT 'NEXT' */ 02410000
DCL /* 3 TO 60 ARE NOT ALL BLANK AND THE ENTERED DATA IF NOT 'NEXT' */ 02420000
DCL /* 3 TO 60 ARE NOT ALL BLANK AND THE ENTERED DATA IF NOT 'NEXT' */ 02430000
IF COMPARM.NEWREQ = 'N' & PCONVSTA.PREV = 'S' & 02440000
 SUBSTR(COMPARM.DATA,3,58) ^= ' ' & 02450000
 COMPARM.PFKIN ^= '08' 02460000
 THEN COMPARM.NEWREQ = 'Y'; 02470000
DCL /* IF NEW REQUEST AND ACTION IS 'ADD' THEN */ 02480000
DCL /* CALL DETAIL PROCESSOR */ 02490000
DCL /* ELSE CALL SECONDARY SELECTION */ 02500000
DCL /* ELSE CALL SECONDARY SELECTION */ 02510000
DCL /* ELSE CALL SECONDARY SELECTION */ 02520000
DCL /* ELSE CALL SECONDARY SELECTION */ 02530000
DCL /* ELSE CALL SECONDARY SELECTION */ 02540000

```

```

IF COMPARM.NEWREQ='Y' THEN 02550000
 DO; 02560000
 IF COMPARM.ACTION = 'A' THEN 02570000
 DO;
 CALL DETAIL; /*CALL DETAIL PROCESSOR */
 GO TO EXIT; /* RETURN */
 END;
 CALL SECSEL; /*CALL SECONDARY SELECTION*/
 IF MAXSEL = 1 THEN /* IF NO. OF CHOICES = 1 */
 CALL DETAIL; /* CALL DETAIL PROCESSOR */
 GO TO EXIT; /* RETURN */
 END;
 /* IF ANSWER TO SECONDARY SELECTION AND NOT A SCROLLING REQUEST */
 /* (INPUT NOT EQUAL TO 'NEXT') AND THE POSITIONS */
 /* 1 TO 2 IN INPUT DATA FIELD NOT EQUAL TO POSITIONS 1 TO 2 */
 /* IN OUTPUT DATA FIELD THEN SEE IF VALID SELECTION. */

 /* DETERMINES IF VALID SELECTION NUMBER */

IF PCONVSTA.PREV ^= 'S' THEN GO TO IP201; /* TO SECONDARY SEL */ 02790000
IF PCONVSTA.MAXSEL < 1 THEN GO TO IP201; /* NO VALID CHOICES */ 02810000
IF COMPARM.PFKIN = '08' THEN GO TO IP201; /* SCROLL REQUEST */ 02830000
IF SUBSTR(COMPARM.DATA,1,2) = SUBSTR(PCONVSTA.DATA,1,2) 02840000
 THEN GO TO IP201; /* NO CHANGE ON INPUT SCREEN */
IF SUBSTR(COMPARM.DATA,2,1) = ' ' THEN /* SECOND CHAR BLANK */ 02880000
 IF VERIFY(SUBSTR(COMPARM.DATA,1,1),'123456789') = 0 THEN 02890000
 DO;
 SUBSTR(COMPARM.DATA,2,1) = SUBSTR(COMPARM.DATA,1,1); 02910000
 SUBSTR(COMPARM.DATA,1,1) = '0'; 02920000
 END;
 IF VERIFY(SUBSTR(COMPARM.DATA,1,2),'0123456789') = 0 & 02950000
 SUBSTR(COMPARM.DATA,1,2) > '00' THEN 02960000
 IF SUBSTR(COMPARM.DATA,1,2) <= PCONVSTA.MAXSEL THEN 02980000
 DO;
 COMPARM.NEWREQ = 'Y'; /*TELL DETAIL PROCESSOR NEW REQ */
 CALL DETAIL; /* CALL DETAIL PROCESSOR*/
 GO TO EXIT; /* RETURN*/
 END;
 /*INVALID SELECTION NO.*/
 /*PRINT ERROR MESSAGE */
 CALL DSN8MPG (MODULE, '072E', OUTMSG);
 PCONVSTA.MSG= OUTMSG;
 GO TO EXIT; /* RETURN */

 /* DETERMINES WHETHER SECONDARY SELECTION OR DETAIL */

/* MUST BE ANY ANSWER TO EITHER SEC SEL OR DETAIL */
IP201:
 IF PCONVSTA.PREV = 'S' THEN 03170000
 DO;
 CALL SECSEL; /*SECONDARY SELECTION*/
 GO TO EXIT; /* RETURN */
 END;
 IF PCONVSTA.PREV = 'D' THEN 03260000
 DO;
 CALL DETAIL; /* DETAIL PROCESSOR */
 GO TO EXIT; /* RETURN */
 END;
 /*LOGIC ERROR */
 CALL DSN8MPG (MODULE, '066E', OUTMSG);
 PCONVSTA.MSG= OUTMSG;
 GO TO EXIT;

```

```

EXEC SQL INCLUDE DSN8MPXX; /*HANDLES SQL ERRORS*/ 03370000
GO TO EXIT;

/***/
/* CALLS SECONDARY SELECTION AND RETURNS TO SQL 1 */
/* NOTE - SAME SECONDARY SELECTION MODULE FOR DS, DE AND EM */
/***/
03380000
03390000
03400000
03410000
03420000
03430000
03440000
03450000
03460000
03470000
03480003
03490000
03500000
03510000
03520000
03530000
03540003
03550000
03560000
03570000
03580000
03590000
03600003
03610000
03620000
03630000
03640000
03650000
03660000
03670000
03680000
03690000
03700000
03710000
03720000
03730000
03740000
03750000
03760000
03770000
03780000
03790000
03800003
03810000
03820000
03830000
03840000
03850000
03860000
03870000
03880000
03890000
03900000
03910000
03920000
03930000
03940000
03950000
03960000
03970000
03980000
03990000
04000000
04010000
04020000
04030000
04040000

SECSEL: PROC; /*CALL APPROPRIATE SECONDARY SEL */ 03450000
PCONVSTA.LASTSCR = 'DSN8001'; /* NOTE GENERAL MAP */ 03460000
DO;
CALL DSN8MPA;
RETURN;
END;

IF COMPARM.OBJFLD='DS' THEN /*ADMINISTRATIVE */ 03480003
DO;
CALL DSN8MPA;
RETURN;
END;

IF COMPARM.OBJFLD='DE' THEN /*DEPARTMENT STRUCTURE */ 03490000
DO;
CALL DSN8MPA;
RETURN;
END;

IF COMPARM.OBJFLD='EM' THEN /*INDIVIDUAL DEPARTMENT*/ 03540003
DO;
CALL DSN8MPA;
RETURN;
END;

CALL DSN8MPG (MODULE, '062E', OUTMSG); /*MISSING SECONDARY SEL*/ 03650000
PCONVSTA.MSG= OUTMSG; /*PRINT ERROR MESSAGE */ 03660000
/*PRINT ERROR MESSAGE*/ 03670000
/*PRINT ERROR MESSAGE*/ 03680000
/*RETURN */ 03690000
/*RETURN */ 03700000
/*RETURN */ 03710000
/*RETURN */ 03720000

GO TO EXIT;
END SECSEL;

/***/
/* CALLS DETAIL PROCESSOR AND RETURNS TO SQL 1 */
/***/
03730000
03740000
03750000
03760000
03770000
03780000
03790000
03800003
03810000
03820000
03830000
03840000
03850000
03860000
03870000
03880000
03890000
03900000
03910000
03920000
03930000
03940000
03950000
03960000
03970000
03980000
03990000
04000000
04010000
04020000
04030000
04040000

DETAIL: PROC; /* CALL APPROPRIATE DETAIL MODULE */ 03770000
PCONVSTA.LASTSCR = 'DSN8002'; /* NOTE DETAIL MAP */ 03780000
SELECT (COMPARM.OBJFLD);
WHEN('DS') CALL DSN8MPD; /*DEPARTMENT STRUCTURE */ 03820000
WHEN('DE') CALL DSN8MPE; /*DEPARTMENT*/ 03840000
WHEN('EM') CALL DSN8MPF; /*EMPLOYEE*/ 03860000
OTHERWISE /*MISSING DETAIL MODULE*/ 03880000
DO;
CALL DSN8MPG (MODULE, '062E', OUTMSG); /*PRINT ERROR MESSAGE */ 03890000
PCONVSTA.MSG= OUTMSG;
END;
END;
END DETAIL;

/*RETURNS TO SQL 1*/ 03970000
03980000
03990000
04000000
04010000
04020000
04030000
04040000

EXIT: EXEC CICS RETURN;

EXEC SQL INCLUDE DSN8MPA; /* SEC SEL - ADMIN STRUCTURE */ 04000000
EXEC SQL INCLUDE DSN8MPD; /* DETAIL - ADMIN STRUCTURE */ 04010000
EXEC SQL INCLUDE DSN8MPE; /* DETAIL - DEPARTMENTS */ 04020000
EXEC SQL INCLUDE DSN8MPF; /* DETAIL - EMPLOYEES */ 04030000
END DSN8CP2; 04040000

```

## Referencia relacionada

[“Ejemplos de aplicaciones en CICS” en la página 1461](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

## DSN8CP6

ESTE MÓDULO EMITE UN CICS MAPA DE RECEPCIÓN PARA RECUPERAR ENTRADAS, LLAMADAS DSN8CP7, Y EMITE UN CICS MAPA DE ENVÍO TRAS EL REGRESO.

```

DSN8CP6 : PROC OPTIONS (MAIN); 00010000
/****** 00020000
* * 00030000
* MODULE NAME = DSN8CP6 * 00040000
* * 00050000
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION * 00060000
* SUBSYSTEM INTERFACE MODULE * 00070000
* CICS * 00080000
* PL/I * 00090000
* PROJECT APPLICATION * 00100000
* * 00110000
* LICENSED MATERIALS - PROPERTY OF IBM 5605-DB2 * 00120000
* (C) COPYRIGHT 1982, 2010 IBM CORP. ALL RIGHTS RESERVED. * 00130000
* * 00140000
* STATUS = VERSION 10 * 00150000
* * 00160000
* FUNCTION = THIS MODULE ISSUES A CICS RECEIVE MAP TO RETRIEVE * 00170000
* INPUT, CALLS DSN8CP7, AND ISSUES A CICS SEND * 00180000
* MAP AFTER RETURNING. * 00190000
* NOTES = * 00200000
* 1. INITIALIZES ITSELF WHEN TERMINAL OPERATOR ENTER INPUT * 00210000
* AFTER VIEWING THE SCREEN SENT BY THE PREVIOUS * 00220000
* ITERATION OF THE PROGRAM. * 00230000
* * 00240000
* DEPENDENCIES = TWO CICS MAPS(DSECTS) ARE REQUIRED : * 00250000
* DSN8MCME AND DSN8MCMF. * 00260000
* MODULES DSN8CP7 IS REQUIRED. * 00270000
* DCLGEN STRUCTURE DSN8MPCS IS REQUIRED. * 00280000
* INCLUDED PLI STRUCTURE DSN8MPCA IS REQUIRED. * 00290000
* * 00300000
* RESTRICTIONS = NONE * 00310000
* * 00320000
* * 00330000
* MODULE TYPE = PL/I PROC OPTIONS(MAIN) * 00340000
* PROCESSOR = DB2 PRECOMPILER, CICS TRANSLATOR, PL/I OPTIMIZER * 00350000
* MODULE SIZE = SEE LINK-EDIT * 00360000
* ATTRIBUTES = REUSABLE * 00370000
* * 00380000
* ENTRY POINT = DSN8CP6 * 00390000
* PURPOSE = SEE FUNCTION * 00400000
* LINKAGE = NONE * 00410000
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION: * 00420000
* * 00430000
* SYMBOLIC LABEL/NAME = NONE * 00440000
* DESCRIPTION = NONE * 00450000
* * 00460000
* OUTPUT = PARAMETERS EXPLICITLY RETURNED: * 00470000
* * 00480000
* SYMBOLIC LABEL/NAME = NONE * 00490000
* DESCRIPTION = NONE * 00500000
* * 00510000
* * 00520000
* EXIT-NORMAL = CICS RETURN TRANSID(D8PP). * 00530000
* * 00540000
* EXIT-ERROR = DB_ERROR FOR SQL ERRORS. * 00550000
* CICS ABEND FOR CICS PROBLEMS. * 00560000
* NO PL/I ON CONDITIONS. * 00570000
* * 00580000
* RETURN CODE = NONE * 00590000
* * 00600000
* ABEND CODES = NONE * 00610000
* * 00620000
* ERROR-MESSAGES = NONE * 00630000
* * 00640000
* EXTERNAL REFERENCES = COMMON CICS REQUIREMENTS * 00650000
* ROUTINES/SERVICES = DSN8CP7 * 00660000
* * 00670000
* DATA-AREAS = * 00680000
* DSN8MPCA - PARAMETER TO BE PASSED TO DSN8CP7 * 00690000
* COMMON AREA * 00700000
* DSN8MPCS - DECLARE CONVERSATION STATUS * 00710000
* DSN8MPMF - CICS/OS/VS PL/I MAP, PROJECTS * 00720000
* DSN8MPME - CICS/OS/VS PL/I MAP, PROJECTS * 00730000
* * 00740000
* CONTROL-BLOCKS = * 00750000
* SQLCA - SQL COMMUNICATION AREA * 00760000
* * 00770000
* TABLES = NONE * 00780000
* * 00790000
* CHANGE-ACTIVITY = NONE * 00800000
* * 00810000

```

```

* * 00820000
* * 00830000
* * 00840000
* * 00850000
* * 00860000
* * 00870000
* * 00880000
* * 00890000
* * 00900000
* * 00910000
* * 00920000
* * 00930000
* * 00940000
* * 00950000
* * 00960000
* * 00970000
* * 00980000
* * 00990000
* * 01000000
* * 01010000
* * 01020000
* * 01030000
* * 01040000
* * 01050000
* * 01060000
* * 01070000
* * 01080000
* * 01090000
* * 01100000
* * 01110000
* * 01120000
* * 01130000
* * 01140000
* * 01150000
* * 01160000
* * 01170000
* * 01180000
* * 01190000
/-----/
/* */ 01200000
/* */ 01210000
/* */ 01220000
/* */ 01230000
/* */ 01240000
/* */ 01250000
/* */ 01260000
/* */ 01270000
/* */ 01280000
/* */ 01290000
/* */ 01300000
/-----/ 01310000
EXEC SQL INCLUDE DSN8MPCA; /* COMMAREA */ 01320000
EXEC SQL INCLUDE DSN8MPMF; /* 1ST MAP, BUILT FROM DSN8CPF */ 01330000
EXEC SQL INCLUDE DSN8MPME; /* 2ND MAP, BUILT FROM DSN8CPE */ 01340000
EXEC SQL INCLUDE SQLCA; /* SQL COMMUNICATION AREA */ 01350000
EXEC SQL INCLUDE DSN8MPCS; /* PCONA */ 01360000
 01370000
0/*****01380000
/* SUBMAP REDEFINES THE PL/I STRUCTURE ASSOCIATED WITH THE */ 01390000
/* CICS MAP DSN8CPE. */ 01400000
/*****01410000
01420000
ODCL MAP1PTR PTR;
MAP2PTR PTR;
DCL IOAREA AREA(2048);
ODCL 1 SUBMAP(15) BASED (ADDR(DSN8CPEI,LINE1F1L)) UNALIGNED,
2 COL1LEN FIXED BIN (15,0) , 01430000
2 COL1ATTR CHAR (1) , 01440000
2 COL1DATA CHAR (37) , 01450000
2 COL2LEN FIXED BIN (15,0) , 01460000
2 COL2ATTR CHAR (1) , 01470000
2 COL2DATA CHAR (40) ; 01480000
 01490000
 01500000
 01510000
 01520000
 01530000
0/*****01540000
/* PFSTRG IS AN ARRAY OF 24 ELEMENTS REPRESENTING THE DIFFERENT */ 01550000
/* PFKEYS AS THEY WOULD BE REPRESENTED IN EIBAID. */ 01560000
/*****01570000
01580000
ODCL CONVID CHAR(16); 01590000
DCL PFSTRG CHAR(24) INIT ('123456789:#@ABCDEFGHI.<'), 01600000
 01610000
0/*****01620000
/* PFK IS AN ARRAY OF 12 TWO-BYTE CHARS REPRESENTING THE PFKEYS */ 01630000

```

```

/* ALLOWED AS INPUT TO DSN8CP7 AND DSN8CP8 ETC. */01640000
/****** */01650000
01660000
PFK(12) CHAR(2) INIT ('01','02','03','04','05','06',
'07','08','09','10','11','12'), 01670000
N FIXED BIN; 01680000
01690000
01700000
/****** */01710000
/* ** DCLGENS AND INITIALIZATIONS */01720000
/****** */01730000
01740000
01750000
01760000
01770000
01780000
DCL STRING BUILTIN; 01790000
DCL J FIXED BIN; 01800000
DCL SAVE_CONVID CHAR(16); 01810000
/* DECLARE CONTROL FLAGS */01820000
DCL (SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1); 01830000
/****** */01840000
01850000
01860000
01870000
01880000
DCL DSN8MPG EXTERNAL ENTRY; 01890000
01900000
/* SQL RETURN CODE HANDLING */01910000
/****** */01920000
01930000
EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR; 01940000
EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR; 01950000
01960000
01970000
/* ALLOCATE PL/I WORK AREA / INITIALIZE VARIABLES */01980000
/****** */01990000
02000000
0ALLOCATE COMMAREA SET(COMMPTR); /*ALLOCATE COMMON AREA */02010000
MAP1PTR = ADDR(IOAREA); /* SET THE POINTER FOR THE GENERAL MAP */02020000
MAP2PTR = ADDR(IOAREA); /* SET THE POINTER FOR THE DETAIL MAP */02030000
COMMAREA = '' ; /*CLEAR COMMON AREA */02040000
DSN8_MODULE_NAME.MAJOR = 'DSN8CP6 ' ; /*GET MODULE NAME */02050000
/*CONSTRUCT CICS CONVERSATION ID */02060000
/*A 4 CHAR TERMINAL ID CON-*/02070000
/*CATENATED WITH 12 BLANKS*/02080000
CONVID,PCONVSTA.CONVID = EIBTRMID || ' ' ; 02090000
OUTAREA.MAJSYS = 'P'; /*SET MAJOR SYSTEM TO P-PROJECT */02100000
EXITCODE = '0'; /*CLEAR EXIT CODE */02110000
02120000
02130000
EXEC CICS HANDLE CONDITION MAPFAIL(CP6SEND); 02140000
02150000
/* TRY TO RETRIEVE LAST CONVERSATION. IF SUCCESSFUL, USE THE */02160000
/* LAST SCREEN SPECIFIED TO RECEIVE INPUT FROM TERMINAL. */02170000
/****** */02180000
02190000
0 EXEC SQL SELECT LASTSCR
INTO :PCONA.LASTSCR 02200000
FROM VCONA 02210000
WHERE CONVID = :CONVID ; 02220000
02230000
02240000
02250000
/* IF LAST CONVERSATION DOES NOT EXIST, THEN DO NOT ATTEMPT TO */02260000
/* RECEIVE INPUT MAP. GO DIRECTLY TO VALIDATION MODULES */02270000
/* TO GET TITLE ETC. FOR OUTPUT MAP. */02280000
/****** */02290000
02300000
0 IF SQLCODE = +100 THEN GO TO CP6SEND; 02310000
02320000
02330000
/* IF DATA IS RECEIVED FOR A FIELD, THENMOVE THE DATA */02340000
/* INTO THE CORRESPONDING FIELDS IN INAREA, OTHERWISE MOVE BLANKS. */02350000
*/
/* IF LAST CONVERSATION EXISTS, BUT OPERATOR HAS ENTERED DATA */02370000
/* FROM A CLEARED SCREEN OR HAD ERASED ALL DATA ON A FORMATTED */02380000
/* SCREEN AND PRESSED ENTER THEN */02390000
/* MOVE DATA INTO CORRESPONDING FIELDS IN INAREA AND GO TO */02400000
/* VALIDATION MODULES. */02410000
/****** */02420000
02430000
IF PCONA.LASTSCR = 'DSN8001 ' THEN 02440000
DO; 02450000

```

```

 /*USING LAST SCREEN */ 02460000
 /*SPECIFIED TO RECEIVE*/ 02470000
 /*INPUT FROM TERMINAL*/ 02480000
EXEC CICS RECEIVE MAP ('DSN8CPF') MAPSET ('DSN8CPF') ; 02490000
 IF AMAJSYSL ^= 0 THEN COMPARM.MAJSYS = AMAJSYSI; 02500000
 ELSE COMPARM.MAJSYS = 'P'; 02510000
 IF AACTIONL ^= 0 THEN COMPARM.ACTION = AACTIONI; 02520000
 ELSE COMPARM.ACTION = ' '; 02530000
 IF AOBJECTL ^= 0 THEN COMPARM.OBJFLD = AOBJECTI; 02540000
 ELSE COMPARM.OBJFLD = ' '; 02550000
 IF ASEARCHL ^= 0 THEN COMPARM.SEARCH = ASEARCHI; 02560000
 ELSE COMPARM.SEARCH = ' '; 02570000
 IF ADATAL ^= 0 THEN COMPARM.DATA = ADATAI ; 02580000
 ELSE COMPARM.DATA = ' '; 02590000
END; 02600000
02610000
0 ELSE IF PCONA.LASTSCR = 'DSN8002' THEN 02620000
 DO; 02630000
 /*MOVE DATA INTO */ 02640000
 /*INPUT FIELDS */ 02650000
 EXEC CICS RECEIVE MAP ('DSN8CPE') MAPSET('DSN8CPE') ; 02660000
 IF BMAJSYSL ^= 0 THEN COMPARM.MAJSYS = BMAJSYSI; 02670000
 ELSE COMPARM.MAJSYS = 'P'; 02680000
 IF BACTIONL ^= 0 THEN COMPARM.ACTION = BACTIONI; 02690000
 ELSE COMPARM.ACTION = ' '; 02700000
 IF BOBJECTL ^= 0 THEN COMPARM.OBJFLD = BOBJECTI; 02710000
 ELSE COMPARM.OBJFLD = ' '; 02720000
 IF BSEARCHL ^= 0 THEN COMPARM.SEARCH = BSEARCHI; 02730000
 ELSE COMPARM.SEARCH = ' '; 02740000
 IF BDATAL ^= 0 THEN COMPARM.DATA = BDATAI ; 02750000
 ELSE COMPARM.DATA = ' '; 02760000
 END; 02770000
 DO I = 1 TO 15; 02780000
 IF SUBMAP.COL2LEN(I) ^= 0 THEN 02790000
 COMPARM.TRANDATA(I) = SUBMAP.COL2DATA(I) ; 02800000
 ELSE COMPARM.TRANDATA(I) = ' '; 02810000
 END; 02820000
 END; 02830000
02840000
0 ELSE /* WRONG LASTSCREEN NAME*/ 02850000
 DO; 02860000
 EXEC CICS ABEND ABCODE ('MAPI'); 02870000
 END; 02880000
02890000
0/******/02900000
/* CONVERT THE PFKEY INFO IN EIBAID TO THE FORM ACCEPTED */02910000
/* BY DSN8CP7 AND DSN8CP8 ETC. EG. PF1 = '01' AND PF13 = '01'. */02920000
/******/02930000
02940000
0 N = INDEX (PFSTRG , EIBAID) ; 02950000
 IF N ^= 0 THEN /* IF PF KEY USED */ 02960000
 DO; 02970000
 IF N > 12 THEN N = N - 12 ; /* PF13 = PF1 ETC. */ 02980000
 COMPARM.PFKIN = PFK(N) ; 02990000
 END; 03000000
 ELSE COMPARM.PFKIN = ' ' ; /* IF ENTER | PAKEYS */ 03020000
 GO TO CP6CP7; 03030000
03040000
/******/03050000
/*
*/03060000
/* GO TO DSN8CP7, GET DCLGEN STRUCTURES AND TABLE DCL */03070000
/*
*/03080000
/******/03090000
03100000
CP6SEND:
 INAREA = ' ' ; /*BLANK OUT INAREA */ 03110000
 COMPARM.PFKIN = '00' ; /*PUT '00' INTO PFKIN*/ 03120000
03130000
03140000
CP6CP7 :
 INAREA.MAJSYS = 'P'; /*SET MAJOR SYSTEM TO P-PROJECT */ 03150000
 /* GO TO DSN8CP7 */ 03160000
 EXEC CICS LINK PROGRAM ('DSN8CP7') COMMAREA(COMMAREA)
 LENGTH(3000); 03170000
03180000
03190000
03200000
03210000
0 EXEC SQL INCLUDE DSN8MPXX; /*GET DCLGEN STRUCTURES*/ 03220000
03230000
 %PAGE; 03240000
/******/03250000
/*
*/03260000
/* AFTER RETURN FROM DSN8CP7 (SQL1), THE PROGRAM EXAMINES DATA */03270000

```

```

/*
 PASSED BACK IN PCONVSTA TO SEE WHAT KIND OF SCREEN SHOULD BE */03280000
/* SENT. PUT THAT DATA INTO THE OUTPUT MAP AND SEND OUTPUT. */03290000
/* IF A SQL ERROR OR WARNING HAD OCCURRED PREVIOUSLY, THE ERROR */03300000
/* MESSAGES ARE EXPECTED TO HAVE BEEN PUT INTO PCONVSTA. */03310000
/*
 *****/03320000
 *****/03330000
 03340000
IF PCONVSTA.LASTSCR = 'DSN8001' THEN /*MOVE DATA INTO */ 03350000
DO; /*OUTPUT FIELDS */ 03360000
 ATITLEO = PCONVSTA.TITLE ; 03370000
 AMAJSYS0= PCONVSTA.MAJSYS; 03380000
 AACTIONO= PCONVSTA.ACTION; 03390000
 AOBJECTO= PCONVSTA.OBJFLD; 03400000
 ASEARCHO= PCONVSTA.SEARCH; 03410000
 ADATAAO = PCONVSTA.DATA ; 03420000
 AMSGO = PCONVSTA.MSG ; 03430000
 ADESCL20= PCONVSTA.DESC2 ; 03440000
 ADESCL30= PCONVSTA.DESC3 ; 03450000
 ADESCL40= PCONVSTA.DESC4 ; 03460000
 APFKEYO = PCONVSTA.PFKTEXT; 03470000
 03480000
DO I = 1 TO 15; /*SEND MAP ACCORDING TO*/ 03490000
 ALINEO(I) = PCONVSTA.OUTPUT.LINE(I); /*PREVIOUS SCREEN*/ 03500000
END; 03510000
 03520000
0/*****03530000
/* CREATES A DYNAMIC CURSOR */03540000
/*****03550000
 03560000
 /*SET CURSOR POSITION */ 03570000
CURSOR_VALUE = 0; /*CLEAR CURSOR */ 03580000
IF AACTIONO = ' ' THEN /*CURSOR SET TO*/ 03590000
 CURSOR_VALUE = 179; /*ACTION POSITION*/ 03600000
ELSE
 IF AOBJECTO = ' ' THEN /*CURSOR SET TO*/ 03620000
 CURSOR_VALUE = 259; /*OBJECT POSITION*/ 03630000
 ELSE
 IF ASEARCHO = ' ' THEN /*CURSOR SET TO*/ 03650000
 CURSOR_VALUE = 339; /*SEARCH POSITION*/ 03660000
 ELSE
 IF ADATAAO = ' ' | /*CURSOR SET TO*/ 03680000
 (AACTIONO = 'D' | /*DATA POSITION*/ 03690000
 AACTIONO = 'U' | 03700000
 AACTIONO = 'A' | 03710000
 AACTIONO = 'E') THEN 03720000
 CURSOR_VALUE = 419; 03730000
 03740000
 03750000
 IF CURSOR_VALUE = 0 THEN /*SEND OUTPUT MAP */ 03760000
 EXEC CICS SEND MAP('DSN8CPF') MAPSET('DSN8CPF'); 03770000
 ELSE
 EXEC CICS SEND MAP('DSN8CPF') MAPSET('DSN8CPF') ERASE
 03780000
 CURSOR(CURSOR_VALUE); 03790000
 03800000
 03810000
 IF EXITCODE = '1' THEN /*FINISHED ? */ 03820000
 EXEC CICS RETURN; /* RETURN, DON'T REINVOKE TRANSACTION*/ 03830000
 ELSE
 EXEC CICS RETURN TRANSID('D8PP'); /* STANDARD RETURN */ 03850000
END; 03860000
 03870000
0/*****03880000
/* MOVES DATA FROM OUTPUT MAP AREA TO */03890000
/* RECEIVE MAP ACCORDING TO MAP SPECIFIED IN LASTSCR OF PCONVST */03900000
/*****03910000
 03920000
 /*MOVE DATA*/ 03930000
 /*FROM OUTPUT FIELDS*/ 03940000
0 ELSE IF PCONVSTA.LASTSCR = 'DSN8002' THEN
DO;
 BTITLEO = PCONVSTA.TITLE ; 03970000
 BMAJSYS0= PCONVSTA.MAJSYS; 03980000
 BACTIONO= PCONVSTA.ACTION; 03990000
 BOBJECTO= PCONVSTA.OBJFLD; 04000000
 BSEARCHO= PCONVSTA.SEARCH; 04010000
 BDATAAO = PCONVSTA.DATA ; 04020000
 BMSGO = PCONVSTA.MSG ; 04030000
 BDESCL20= PCONVSTA.DESC2 ; 04040000
 BDESCL30= PCONVSTA.DESC3 ; 04050000
 BDESCL40= PCONVSTA.DESC4 ; 04060000
 BPFKEYO = PCONVSTA.PFKTEXT; 04070000
 04080000
DO I = 1 TO 15 ; /*RECEIVE MAP ACCORDING TO */ 04090000

```

```

SUBMAP.COL1DATA(I) = REOUT.FIELD1(I); /*PREVIOUS SCREEN */ 04100000
 04110000
0 /*----- */ 04120000
/* */ 04130000
/* MODULES DSN8MPE, DSN8MPF ETC. IN SQL2 HAVE PUT THE */ 04140000
/* ATTRIBUTE BYTE AND CURSOR CONTROL INFO IN IMS MFS */ 04150000
/* FORM - HEX'C0' FOR DYNAMIC CURSOR WITH 2 BYTES OF */ 04160000
/* ATTRIBUTE INFORMATION TO FOLLOW. THIS PROGRAM CHECKS */ 04170000
/* FOR THE HEX'C0' AND INSERTS -1 INTO */ 04180000
/* THE LENGTH FIELD ASSOCIATED WITH THE DATA TO CONFORM */ 04190000
/* WITH THE STANDARD WAY OF HANDLING DYNAMIC CURSORS IN */ 04200000
/* CICS. SIMILARLY, ONLY THE SECOND OF THE TWO ATTRIBUTE */ 04210000
/* BYTES IS MOVED INTO THE CICS ATTRIBUTE BYTE. THE */ 04220000
/* FIRST TWO BITS OF THE ATTRIBUTE BYTE IS DIFFERENT */ 04230000
/* BETWEEN IMS AND CICS STANDARD REPRESENTATIONS, HOWEVER */ 04240000
/* 3270 MANUALS INDICATE THAT ON OUTPUT, THE FIRST */ 04250000
/* TWO BITS ARE IGNORED. THUS THE SAME ATTRIBUTE BYTE */ 04260000
/* IS USED BETWEEN IMS AND CICS MODULES. */ 04270000
/* */ 04280000
/*----- */ 04290000
04300000
0 IF UNSPEC(REOUT.ATTR1(I)) = '11000000'B /* X'C0' ATTR */ 04310000
 THEN SUBMAP.COL2LEN(I) = -1; 04320000
 SUBMAP.COL2ATTR(I) = REOUT.ATTR2(I); 04330000
 SUBMAP.COL2DATA(I) = REOUT.FIELD2(I); 04340000
END; 04350000
04360000
0/***** */ 04370000
/* CREATES A DYNAMIC CURSOR */ 04380000
/***** */ 04390000
04400000
CURSOR_VALUE = 0; /*SET CURSOR POSITION */ 04410000
IF BACTIONO = ' ' THEN /*CLEAR CURSOR */ 04420000
 CURSOR_VALUE = 179; /*CURSOR SET TO*/ 04430000
ELSE /*ACTION POSITION*/ 04440000
 IF BOBJECTO = ' ' THEN /*CURSOR SET TO*/ 04450000
 CURSOR_VALUE = 259; /*OBJECT POSITION*/ 04460000
 ELSE /*CURSOR SET TO*/ 04470000
 IF BSEARCHO = ' ' THEN /*CURSOR SET TO*/ 04480000
 CURSOR_VALUE = 339; /*SEARCH POSITION*/ 04490000
 ELSE /*CURSOR SET TO*/ 04500000
 IF BDATAO = ' ' | /*CURSOR SET TO*/ 04510000
 (BACTIONO = 'D' | /*DATA POSITION*/ 04520000
 BACTIONO = 'U' | 04530000
 BACTIONO = 'A' | 04540000
 BACTIONO = 'E') THEN 04550000
 CURSOR_VALUE = 419; 04560000
 END; /*SEND INPUT MAP */ 04570000
 END; /*SEND INPUT MAP */ 04580000
 END; /*SEND INPUT MAP */ 04590000
 END; /*SEND INPUT MAP */ 04600000
 EXEC CICS SEND MAP('DSN8CPE') MAPSET('DSN8CPE'); 04610000
ELSE /*SEND INPUT MAP */ 04620000
 EXEC CICS SEND MAP('DSN8CPE') MAPSET('DSN8CPE') ERASE 04630000
 CURSOR(CURSOR_VALUE); 04640000
04650000
04660000
IF EXITCODE = '1' THEN /*FINISHED ? */ 04670000
 EXEC CICS RETURN; /* RETURN, DON'T REINVOKE TRANSACTION */ 04680000
ELSE /*STANDARD RETURN */ 04690000
 EXEC CICS RETURN TRANSID('D8PP'); /* STANDARD RETURN */ 04700000
END; 04710000
04720000
0 /* SQL1 DID NOT PASS BACK VALID LASTSCREEN NAME */ 04730000
ELSE EXEC CICS ABEND ABCODE ('MAPO'); 04740000
END; 04750000

```

### Referencia relacionada

[“Ejemplos de aplicaciones en CICS” en la página 1461](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

### DSN8CP7

ESTE MÓDULO REALIZA LOS INCLUIDOS PARA INTRODUCIR LAS ESTRUCTURAS DE TABLA SQL DCLS Y DCLGEN, ASÍ COMO EL ÁREA DE PARÁMETROS.

```

DSN8CP7:PROC (COMMTPR) OPTIONS(MAIN);
/*****

```

```

*
* MODULE NAME = DSN8CP7
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
* SQL 1 MAINLINE
* CICS
* PL/I
* PROJECT APPLICATION
*
* COPYRIGHT = 5740-XYR (C) COPYRIGHT IBM CORP 1982, 1985
* REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
*
* STATUS = RELEASE 2, LEVEL 0
*
* FUNCTION = THIS MODULE PERFORMS THE INCLUDES TO BRING IN THE
* SQL TABLE DCLS AND DCLGEN STRUCTURES AS WELL AS
* THE PARAMETER AREA.
* INCLUDE DSN8MP1.
* CALL DSN8CP8.
* RETURN TO DSN8CP6.
*
* NOTES =
* DEPENDENCIES = CALLED BY DSN8CP6, CALLS DSN8CP8 (CICS LINKS).
* RESTRICTIONS = NONE
*
* MODULE TYPE = PL/I PROC(COMMPTR) OPTIONS.
* PROCESSOR = DB2 PRECOMPILER, CICS TRANSLATOR, PL/I OPTIMIZER
* MODULE SIZE = SEE LINK-EDIT
* ATTRIBUTES = REUSABLE
*
* ENTRY POINT = DSN8CP7
* PURPOSE = SEE FUNCTION
* LINKAGE = NONE
*
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
*
* SYMBOLIC LABEL/NAME = COMMTPR (POINTER TO COMMAREA)
* DESCRIPTION = NONE
*
* OUTPUT = PARAMETERS EXPLICITLY RETURNED:
*
* SYMBOLIC LABEL/NAME = NONE
* DESCRIPTION = NONE
*
* EXIT-NORMAL = DSN8CP6
*
* EXIT-ERROR = DSN8CP6
*
* RETURN CODE = NONE
*
* ABEND CODES = NONE
*
* ERROR-MESSAGES = NONE
*
* EXTERNAL REFERENCES =
* ROUTINES/SERVICES = DSN8CP8
*
* DATA-AREAS =
* DSN8MPCA - PLI STRUCTURE FOR COMMAREA
* DSN8MPCS - DECLARE CONVERSATION STATUS
* DSN8MPOV - DECLARE OPTION VALIDATION
* DSN8MPVO - FIND VALID OPTIONS FOR ACTION,
* OBJECT, SEARCH CRITERIA
* DSN8MP1 - RETRIEVE LAST CONVERSATION,
* VALIDATE, CALL SQL2
* DSN8MP3 -- DSN8MP5 - VALIDATION MODULES CALLED BY DSN8MP1
* DSN8MPXX - SQL ERROR HANDLER
*
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA
*
* TABLES = NONE
*
* CHANGE-ACTIVITY = NONE
*
* *PSEUDOCODE*
*
* PROCEDURE
* INCLUDE DECLARATIONS.
* INCLUDE DSN8MP1.

```

#### **Referencia relacionada**

“Ejemplos de aplicaciones en CICS” en la página 1461

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

## DSN8CP8

ROUTER PARA SELECCIÓN SECUNDARIA Y/O PROCESAMIENTO DETALLADO LLAMA A MÓDULOS DE SELECCIÓN SECUNDARIA DSN8MPM LLAMA A MÓDULOS DETALLADOS DSN8MPT DSN8MPV DSN8MPW DSN8MPX DSN8MPZ LLAMADA POR DSN8CP7 (SQL1).

```
DSN8CP8: PROC(COMMPTR) OPTIONS(MAIN); 00010000
 %PAGE; 00020000
/***** 00030000
* * 00040000
* MODULE NAME = DSN8CP8 * 00050000
* * 00060000
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION * 00070000
* SQL 2 COMMON MODULE * 00080000
* CICS * 00090000
* PL/I * 00100000
* PROJECT APPLICATION * 00110000
*
* LICENSED MATERIALS - PROPERTY OF IBM * 00120000
* 5695-DB2 * 00130000
* (C) COPYRIGHT 1982, 1995 IBM CORP. ALL RIGHTS RESERVED. * 00140000
*
* STATUS = VERSION 4 * 00146000
* * 00153000
* * 00160000
* * 00170000
* * 00180000
* FUNCTION = ROUTER FOR SECONDARY SELECTION AND/OR DETAIL PROCESSIN* 00190000
* CALLS SECONDARY SELECTION MODULES * 00200000
* DSN8MPM * 00210000
* CALLS DETAIL MODULES * 00220000
* DSN8MPT DSN8MPV DSN8MPW DSN8MPX DSN8MPZ * 00230000
* CALLED BY DSN8CP7 (SQL1) * 00240000
* * 00250000
* NOTES = NONE * 00260000
* * 00270000
* MODULE TYPE = BLOCK OF PL/I CODE * 00280000
* PROCESSOR = DB2 PRECOMPLIER, PL/I OPTIMIZER * 00290000
* MODULE SIZE = SEE LINKEDIT * 00300000
* ATTRIBUTES = REUSABLE * 00310000
* * 00320000
* ENTRY POINT = DSN8CP8 * 00330000
* PURPOSE = SEE FUNCTION * 00340000
* LINKAGE = NONE * 00350000
* INPUT = * 00360000
* * 00370000
* SYMBOLIC LABEL/NAME = COMMPTR * 00380000
* DESCRIPTION = POINTER TO COMMAREA * 00390000
* (COMMUNICATION AREA) * 00400000
* * 00410000
* OUTPUT = * 00420000
* * 00430000
* SYMBOLIC LABEL/NAME = COMMPTR * 00440000
* DESCRIPTION = POINTER TO COMMAREA * 00450000
* (COMMUNICATION AREA) * 00460000
* * 00470000
* * 00480000
* EXIT-NORMAL = * 00490000
* * 00500000
* EXIT-ERROR = IF SQLERROR OR SQLWARNING, SQL WHENEVER CONDITION * 00510000
* SPECIFIED IN DSN8CP8 WILL BE RAISED AND PROGRAM * 00520000
* WILL GO TO THE LABEL DB_ERROR. * 00530000
* * 00540000
* * 00550000
* RETURN CODE = NONE * 00560000
* * 00570000
* ABEND CODES = NONE * 00580000
* * 00590000
* ERROR-MESSAGES = * 00600000
* DSN8062E-AN OBJECT WAS NOT SELECTED * 00610000
* DSN8066E-UNSUPPORTED PFK OR LOGIC ERROR * 00630000
* DSN8072E-INVALID SELECTION ON SECONDARY SCREEN * 00640000
* * 00650000
* EXTERNAL REFERENCES = NONE * 00660000
* ROUTINES/SERVICES = 6 MODULES LISTED ABOVE * 00670000
* DSN8MPG - ERROR MESSAGE ROUTINE * 00680000
* * 00690000
* DATA-AREAS = * 00700000
* DSN8MPAC - DCLGEN FOR ACTIVITY TYPES * 00710000
* DSN8MPAS - CURSOR SECONDARY SELECTION FOR STAFF * 00720000
* * 00730000
```

```

* DSN8MPCA - COMMUNICATION AREA BETWEEN MODULES * 00740000
* DSN8MPDH - CURSOR FOR DISPLAY TEXT FROM * 00750000
* TDSPTXT TABLE * 00760000
* DSN8MPDP - DCLGEN FOR DEPARTMENT * 00770000
* DSN8MPDT - DCLGEN FOR DISPLAY TEXT TABLE * 00780000
* DSN8MPEM - DCLGEN FOR EMPLOYEE * 00790000
* DSN8MPEP - DCLGEN FOR PROJECT/STAFFING * 00800000
* DSN8MPES - CURSOR SECONDARY SELECTION FOR * 00810000
* ESTIMATES * 00820000
* DSN8MPOV - DCLGEN FOR OPTION VALIDATION * 00830000
* DSN8MPPA - DCLGEN FOR PROJECT/ACTIVITIES * 00840000
* DSN8MPPD - DCLGEN FOR PROJ STRUCTURE DETAIL * 00850000
* DSN8MPP2 - DCLGEN FOR PROJ STRUCTURE DETAIL * 00855000
* DSN8MPPE - CURSOR PROJECT DETAIL * 00860000
* DSN8MPPJ - DCLGEN FOR PROJECTS * 00870000
* DSN8MPPL - CURSOR PROJECT LIST * 00880000
* DSN8MPPR - DCLGEN FOR PROJ/RESP EMPLOYEE * 00890000
* DSN8MPSA - DCLGEN FOR PROJ ACTIVITY LISTING * 00910000
* DSN8MPSL - CURSOR STAFFING LIST * 00920000
* DSN8MPS2 - DCLGEN FOR PROJ ACTIVITY LISTING * 00930000
* DSN8MPFP - DCLGEN FOR PROJECT-EMPLOYEE * 00935000
* DSN8MPED - DCLGEN FOR EMPLOYEE-DEPT * 00937000
* DSN8MPM - SECONDARY SELECTION FOR PROJECTS * 00940000
* DSN8MPT - PROJECT ACTIVITY LIST * 00950000
* DSN8MPV - PROJECT STRUCTURE DETAIL * 00960000
* DSN8MPW - ACTIVITY STAFFING DETAIL * 00970000
* DSN8MPX - ACTIVITY ESTIMATE DETAIL * 00980000
* DSN8MPZ - PROJECT DETAIL * 00990000
* * 01000000
* CONTROL-BLOCKS = - SQL COMMUNICATION AREA * 01010000
* SQLCA * 01020000
* * 01030000
* TABLES = NONE * 01040000
* CHANGE-ACTIVITY = NONE * 01050000
* * 01060000
* * 01070000
* * 01080000
* *PSEUDOCODE*
* THIS MODULE DETERMINES WHICH SECONDARY SELECTION AND/OR
* DETAIL MODULE(S) ARE TO BE CALLED IN THE CICS/PL/I
* ENVIRONMENT. * 01110000
* * 01120000
* * 01130000
* * 01140000
* WHAT HAS HAPPENED SO FAR?.....THE SUBSYSTEM
* DEPENDENT MODULE (IMS,CICS,TSO) OR (SQL 0) HAS * 01150000
* READ THE INPUT SCREEN, FORMATTED THE INPUT AND PASSED CONTROL * 01160000
* TO SQL 1. SQL 1 PERFORMS VALIDATION ON THE SYSTEM DEPENDENT * 01170000
* FIELDS (MAJOR SYSTEM, ACTION, OBJECT, SEARCH CRITERIA). IF * 01180000
* ALL SYSTEM FIELDS ARE VALID SQL 1 PASSED CONTROL TO THIS * 01190000
* MODULE. PASSED PARAMETERS CONSIST ONLY OF A POINTER WHICH * 01200000
* POINTS TO A COMMUNICATION CONTROL AREA USED TO COMMUNICATE * 01210000
* BETWEEN SQL 0 , SQL 1, SQL 2 AND THE SECONDARY SELECTION * 01220000
* AND DETAIL MODULES. * 01230000
* * 01240000
* * 01250000
* WHAT IS INCLUDED IN THIS MODULE?.....
* ALL SECONDARY SELECTION AND DETAIL MODULES ARE 'INCLUDED'. * 01260000
* ALL VARIABLES KNOWN IN THIS PROCEDURE ARE KNOWN IN THE * 01270000
* SUB PROCEDURES. ALL SQL CURSOR DEFINITIONS AND * 01280000
* SQL 'INCLUDES' ARE DONE IN THIS PROCEDURE. BECAUSE OF THE * 01290000
* RESTRICTION THAT CURSOR HOST VARIABLES MUST BE DECLARED BEFORE * 01310000
* THE CURSOR DEFINITION ALL CURSOR HOST VARIABLES ARE DECLARED * 01320000
* IN THIS PROCEDURE. * 01330000
* * 01340000
* PROCEDURE
* IF ANSWER TO DETAIL SCREEN & DETAIL PROCESSOR * 01350000
* IS NOT WILLING TO ACCEPT AN ANSWER THEN
* NEW REQUEST*
* ELSE
* IF ANSWER TO A SECONDARY SELECTION THEN * 01360000
* DETERMINE IF NEW REQUEST. * 01370000
* CASE (NEW REQUEST)
* SUBCASE ('ADD')
* DETAIL PROCESSOR * 01380000
* RETURN TO SQL 1 * 01390000
* ENDSUB
* SUBCASE ('DISPLAY', 'ERASE', 'UPDATE') * 01400000
* * 01410000
* * 01420000
* * 01430000
* * 01440000
* * 01450000
* CASE (NEW REQUEST)
* SUBCASE ('DISPLAY', 'ERASE', 'UPDATE') * 01460000
* * 01470000
* ENDSUB
* SUBCASE ('DISPLAY', 'ERASE', 'UPDATE') * 01480000
* * 01490000
* ENDSUB
* SUBCASE ('DISPLAY', 'ERASE', 'UPDATE') * 01500000
* * 01510000
* * 01520000
* ENDSUB
* SUBCASE ('DISPLAY', 'ERASE', 'UPDATE') * 01530000

```

```

* CALL SECONDARY SELECTION * 01540000
* IF # OF POSSIBLE CHOICES IS ^= 1 THEN * 01550000
* RETURN TO SQL 1 * 01560000
* ELSE * 01570000
* CALL THE DETAIL PROCESSOR * 01580000
* RETURN TO SQL 1 * 01590000
* ENDSub * 01600000
* * 01610000
* ENDCASE * 01620000
* * 01630000
* IF ANSWER TO SECONDARY SELECTION AND A SELECTION HAS
* ACTUALLY BEEN MADE THEN * 01640000
* VALID SELECTION #?
* IF IT IS VALID THEN * 01650000
* CALL DETAIL PROCESSOR * 01660000
* RETURN TO SQL 1 * 01670000
* ELSE * 01680000
* PRINT ERROR MSG * 01690000
* RETURN TO SQL 1. * 01700000
* * 01710000
* IF ANSWER TO SECONDARY SELECTION THEN
* CALL SECONDARY SELECTION * 01720000
* RETURN TO SQL 1. * 01730000
* * 01740000
* IF ANSWER TO DETAIL THEN * 01750000
* CALL DETAIL PROCESSOR * 01760000
* RETURN TO SQL 1. * 01770000
* * 01780000
* IF ANSWER TO DETAIL THEN * 01790000
* CALL DETAIL PROCESSOR * 01800000
* RETURN TO SQL 1. * 01810000
* END.
* * 01820000
* * 01830000
* *EXAMPLE- A ROW IS SUCCESSFULLY ADDED, THE OPERATOR RECEIVES* 01840000
* THE SUCCESSFULLY ADDED MESSAGE AND JUST HITS ENTER. * 01850000
* * 01860000
-----/ 01870000
-----/ 01880000
EXEC SQL INCLUDE DSN8MPCA; /*COMMUNICATION AREA BETWEEN MODULES */ 01890000
EXEC SQL INCLUDE SQLCA; /*SQL COMMUNICATION AREA */ 01900000
 01910000
EXEC SQL INCLUDE DSN8MPDP; /* DCLGEN FOR DEPARTMENT */ 01920000
EXEC SQL INCLUDE DSN8MPEM; /* DCLGEN FOR EMPLOYEE */ 01930000
EXEC SQL INCLUDE DSN8MPPJ; /* DCLGEN FOR PROJECTS */ 01940000
EXEC SQL INCLUDE DSN8MPAC; /* DCLGEN FOR ACTIVITY TYPES */ 01950000
EXEC SQL INCLUDE DSN8MPPA; /* DCLGEN FOR PROJECT/ACTIVITIES */ 01960000
EXEC SQL INCLUDE DSN8MPEP; /* DCLGEN FOR PROJECT/STAFFING */ 01970000
EXEC SQL INCLUDE DSN8MPPR; /* DCLGEN FOR PROJ/RESP EMPLOYEE */ 01980000
EXEC SQL INCLUDE DSN8MPD; /* DCLGEN FOR PROJ STRUCTURE DETAIL */ 01990000
EXEC SQL INCLUDE DSN8MPP2; /* DCLGEN FOR PROJ STRUCTURE DETAIL */ 02000000
EXEC SQL INCLUDE DSN8MPSA; /* DCLGEN FOR PROJ ACTIVITY LISTING */ 02010000
EXEC SQL INCLUDE DSN8MPS2; /* DCLGEN FOR PROJ ACTIVITY LISTING */ 02020000
EXEC SQL INCLUDE DSN8MPFP; /* DCLGEN FOR PROJECT-EMPLOYEE */ 02025000
EXEC SQL INCLUDE DSN8MPED; /* DCLGEN FOR EMPLOYEE-DEPT */ 02027000
 /* PROGRAMMING TABLES */ 02030000
EXEC SQL INCLUDE DSN8MPOV; /* DCLGEN FOR OPTION VALIDATION */ 02040000
EXEC SQL INCLUDE DSN8MPDT; /* DCLGEN FOR DISPLAY TEXT TABLE */ 02050000
 02060000
 02070000
/* CURSORS */
EXEC SQL INCLUDE DSN8MPPL; /* MAJSYS P - SEC SEL FOR PS, AL, PR*/ 02080000
EXEC SQL INCLUDE DSN8MPES; /* MAJSYS P - SEC SEL FOR AE */ 02090000
EXEC SQL INCLUDE DSN8MPAS; /* MAJSYS P - SEC SEL FOR AS */ 02100000
EXEC SQL INCLUDE DSN8MPPE; /* MAJSYS P - DETAIL FOR PS */ 02110000
EXEC SQL INCLUDE DSN8MPSL; /* MAJSYS P - DETAIL FOR AL */ 02120000
EXEC SQL INCLUDE DSN8MPDH; /* PROG TABLES - DISPLAY HEADINGS */ 02130000
 02140000
/***/ 02150000
/* SQL RETURN CODE HANDLING */ 02160000
/***/ 02170000
 02180000
EXEC SQL WHENEVER SQLERROR GO TO DB_ERROR; 02190000
EXEC SQL WHENEVER SQLWARNING GO TO DB_ERROR; 02200000
 02210000
④ DCL UNSPEC BUILTIN; 02220000
DCL VERIFY BUILTIN; 02230000
 02240000
DCL LENGTH BUILTIN; 02250000
DCL DSN8MPG EXTERNAL ENTRY; 02260000
 02270000
/***/ 02280000
/* ** DCLGENS AND INITIALIZATIONS */ 02290000
/***/ 02300000
 02310000
DCL STRING BUILTIN; 02320000
DCL J FIXED BIN; 02330000

```

```

DCL SAVE_CONVID CHAR(16); 02340000
 /* DECLARE CONTROL FLAGS */
DCL (SENDBIT, ENDBIT, NEXTBIT, ON, OFF) BIT(1); 02350000
 02360000
 02370000
/***** 02380000
/* FIELDS SENT TO MESSAGE ROUTINE */ 02390000
/***** 02400000
 02410000
DCL MODULE CHAR (07) INIT('DSN8CP8'); 02420000
DCL OUTMSG CHAR (69); 02430000
 02440000
/***** 02450000
/* INITIALIZATIONS */ */ 02460000
/***** 02470000
 02480000
DSN8_MODULE_NAME.MAJOR='DSN8CP8'; 02490000
DSN8_MODULE_NAME.MINOR=' '; 02500000
 02510000
/***** 02520000
/* DETERMINES WHETHER NEW REQUEST OR NOT */ 02530000
/***** 02540000
 02550000
/* IF 'NO ANSWER POSSIBLE' SET BY DETAIL PROCESSOR THEN FORCE A */ 02560000
/* NEW REQUEST. */ */ 02570000
 02580000
IF PCONVSTA.PREV = ' ' THEN 02590000
 COMPARM.NEWREQ = 'Y';
 02600000
 02610000
/* IF ANSWER TO SECONDARY SELECTION THEN DETERMINE IF REALLY A */ 02620000
/* NEW REQUEST. IT WILL BE CONSIDERED A NEW REQUEST IF POSITIONS*/ 02630000
/* * 3 TO 60 ARE NOT ALL BLANK AND THE ENTERED DATA IF NOT 'NEXT' */ 02640000
 02650000
IF COMPARM.NEWREQ = 'N' & PCONVSTA.PREV = 'S' & 02660000
 SUBSTR(COMPARM.DATA,3,58) ^= ' ' & 02670000
 COMPARM.DATA ^= 'NEXT'; 02680000
 THEN COMPARM.NEWREQ = 'Y';
 02690000
 02700000
/***** 02710000
/* IF NEW REQUEST AND ACTION IS 'ADD' THEN */ 02720000
/* CALL DETAIL PROCESSOR */ */ 02730000
/* ELSE CALL SECONDARY SELECTION */ */ 02740000
/***** 02750000
IF COMPARM.NEWREQ='Y' THEN 02760000
DO;
 02770000
 02780000
IF COMPARM.ACTION = 'A' THEN 02790000
DO;
 CALL DETAIL; /* CALL DETAIL PROCESSOR*/ 02810000
 GO TO EXIT; /* RETURN */ 02820000
END;
 02830000
 02840000
CALL SECSEL; /* CALL SECONDARY SELECTION */ 02850000
 02860000
IF MAXSEL = 1 THEN /* IF NO. OF CHOICES = 1 */ 02870000
 CALL DETAIL; /* CALL DETAIL PROCESSOR */ 02880000
 GO TO EXIT; /* RETURN */ 02890000
END;
 02900000
 02910000
/* IF ANSWER TO SECONDARY SELECTION AND NOT A SCROLLING REQUEST */ 02920000
/* (INPUT NOT EQUAL TO 'NEXT') AND THE POSITIONS */ 02930000
/* * 1 TO 2 IN INPUT DATA FIELD NOT EQUAL TO POSITIONS 1 TO 2 */ 02940000
/* IN OUTPUT DATA FIELD THEN SEE IF VALID SELECTION. */ 02950000
 02960000
/***** 02970000
/* DETERMINES IF VALID SELECTION NUMBER */ 02980000
/***** 02990000
 03000000
IF PCONVSTA.PREV ^= 'S' THEN GO TO IP201; /* TO SECONDARY SEL */ 03010000
 03020000
IF PCONVSTA.MAXSEL < 1 THEN GO TO IP201; /* NO VALID CHOICES */ 03030000
 03040000
IF COMPARM.DATA = 'NEXT' THEN GO TO IP201; /* SCROL REQUEST*/ 03050000
 03060000
IF SUBSTR(COMPARM.DATA,1,2) = SUBSTR(PCONVSTA.DATA,1,2) 03070000
 THEN GO TO IP201; /* NO CHANGE ON INPUT SCREEN */ 03080000
 03090000
IF SUBSTR(COMPARM.DATA,2,1) = ' ' THEN /* SECOND CHAR BLANK */ 03100000
 IF VERIFY(SUBSTR(COMPARM.DATA,1,1),'123456789') = 0 THEN 03110000
 DO;
 SUBSTR(COMPARM.DATA,2,1) = SUBSTR(COMPARM.DATA,1,1);
 SUBSTR(COMPARM.DATA,1,1) = '0';
 END;
 03120000
 03130000
 03140000
 03150000

```

```

IF VERIFY(SUBSTR(COPARM.DATA,1,2),'0123456789') = 0 & 03160000
 SUBSTR(COPARM.DATA,1,2) > '00' THEN 03170000
 IF DATAP <= PCONVSTA.MAXSEL THEN 03180000
 DO;
 COPARM.NEWREQ = 'Y'; /* TELL DETAIL PROCESSOR NEW REQ */ 03190000
 CALL DETAIL; /* CALL DETAIL PROCESSOR */ 03200000
 GO TO EXIT; /* RETURN */ 03210000
 END;
 /* INVALID SELECTION NO. */ 03220000
 CALL DSN8MPG (MODULE, '072E', OUTMSG); /* PRINT ERROR MSG */ 03230000
 PCONVSTA.MSG = OUTMSG;
 PCONVSTA.PREV = ' '; /* NOT SEC SELECTION, ERROR */ 03240000
 GO TO EXIT; /* RETURN */ 03250000
 03260000
 /* **** */
 /* DETERMINES WHETHER SECONDARY SELECTION OR DETAIL */ 03270000
 /* **** */
/* MUST BE ANY ANSWER TO EITHER SEC SEL OR DETAIL */ 03280000
IP201: 03290000
 IF PCONVSTA.PREV = 'S' THEN 03300000
 DO;
 CALL SECSEL; /* CALL SECONDARY SELECTION */ 03310000
 GO TO EXIT; /* RETURN */ 03320000
 END;
 IF PCONVSTA.PREV = 'D' THEN 03330000
 DO;
 CALL DETAIL; /* CALL DETAIL PROCESSOR */ 03340000
 GO TO EXIT; /* RETURN */ 03350000
 END;
 /* LOGIC ERROR */
 /* PRINT ERROR MESSAGE */ 03360000
 CALL DSN8MPG (MODULE, '066E', OUTMSG); 03370000
 PCONVSTA.MSG= OUTMSG;
 PCONVSTA.PREV = ' '; /* NOT SEC SELECTION, ERROR */ 03380000
 GO TO EXIT; /* RETURN */ 03390000
 EXEC SQL INCLUDE DSN8MPXX; /* HANDLES SQL ERRORS */ 03400000
 GO TO EXIT; /* RETURN */ 03410000
 /* **** */
 /* CALLS SECONDARY SELECTION AND RETURNS TO SQL 1 */
 /* NOTE - SAME SECONDARY SELECTION MODULE FOR DS, DE AND EM */
 /* **** */
SECSEL: PROC; /* CALL APPROPRIATE SECONDARY SELECTION MODULE */ 03420000
 PCONVSTA.LASTSCR = 'DSN8001'; /* SET FOR GENERAL MAP */ 03430000
 IF COPARM.OBJFLD='AE' | /*ACTIVITY ESTIMATE */ 03440000
 COPARM.OBJFLD='AL' | /*PROJECT ACTIVITY LISTING */ 03450000
 COPARM.OBJFLD='AS' | /*INDIVIDUAL PROJECT STAFFING*/ 03460000
 COPARM.OBJFLD='PR' | /*INDIVIDUAL PROJECT PROCESSING*/ 03470000
 COPARM.OBJFLD='PS' THEN /*PROJECT STRUCTURE */ 03480000
 DO;
 CALL DSN8MPM; /*SECONDARY SELECTION FOR PROJECTS*/ 03490000
 RETURN; /*RETURN */ 03500000
 END;
 /*MISSING SECONDARY SEL*/
 /*PRINT ERROR MESSAGE */ 03510000
 CALL DSN8MPG (MODULE, '062E', OUTMSG); 03520000
 PCONVSTA.MSG= OUTMSG;
 PCONVSTA.PREV = ' '; /* NOT SEC SELECTION, ERROR */ 03530000
 GO TO EXIT; /*RETURN */ 03540000
END SECSEL;

 /* **** */
 /* CALLS DETAIL PROCESSOR AND RETURNS TO SQL 1 */
 /* **** */
DETAIL: PROC; /* CALL APPROPRIATE DETAIL MODULE */ 03550000
 PCONVSTA.LASTSCR = 'DSN8002'; /* SET FOR DETAIL MAP */ 03560000
 IF COPARM.OBJFLD='PS' THEN 03570000
 DO;
 CALL DSN8MPV; /* PROJECT STRUCTURE DETAIL */ 03580000
 RETURN; /*PROJECT STRUCTURE DETAIL */ 03590000
 END;
 /*MISSING SECONDARY SEL*/
 /*PRINT ERROR MESSAGE */ 03600000
 /* **** */
 /* **** */
03610000
03620000
03630000
03640000
03650000
03660000
03670000
03680000
03690002
03700002
03710002
03720002
03730002
03740000
03750000
03760000
03770000
03780000
03790000
03800000
03810000
03820000
03830000
03840000
03850000
03860000
03870000
03880000
03890000
03900000
03910000
03920000
03930000
03940002
03950000
03960000
03970000

```

```

 END; 03980000
IF COMPARM.OBJFLD='AL' THEN 03990000
 DO;
 CALL DSN8MPT; /* PROJECT ACTIVITY LIST */ 04000002
 RETURN;
 END;

IF COMPARM.OBJFLD='PR' THEN 04010000
 DO;
 CALL DSN8MPZ; /* PROJECT DETAIL */ 04020000
 RETURN;
 END;

IF COMPARM.OBJFLD='AE' THEN 04030000
 DO;
 CALL DSN8MPX; /* ACTIVITY ESTIMATE DETAIL */ 04040000
 RETURN;
 END;

IF COMPARM.OBJFLD='AS' THEN 04050000
 DO;
 CALL DSN8MPW; /* ACTIVITY STAFFING DETAIL */ 04060002
 RETURN;
 END;

CALL DSN8MPG (MODULE, '062E', OUTMSG); 04070000
PCONVSTA.MSG= OUTMSG;
PCONVSTA.PREV = ' ' ; /* NOT SEC SELECTION, ERROR */ 04080000
GO TO EXIT;
END DETAIL;

/*MISSING DETAIL MODULE*/
/*PRINT ERROR MESSAGE */
04090000
04100000
04110000
04120002
04130000
04140000
04150000
04160000
04170000
04180002
04190000
04200000
04210000
04220000
04230000
04240000
04250000
04260000
04270000
04280000
04290000
04300000
04310000
04320000
04330000
04340000
04350000
04360000
04370000
04380000
04390000
04400000
04410000

EXIT: EXEC CICS RETURN;
 /* PROJECTS */
 EXEC SQL INCLUDE DSN8MPM; /* SEC SEL - PROJECTS */ 04420000
 EXEC SQL INCLUDE DSN8MPT; /* DETAIL - PROJ ACT LISTING*/ 04430000
 EXEC SQL INCLUDE DSN8MPV; /* DETAIL - PROJ STRUCTURE */ 04440000
 EXEC SQL INCLUDE DSN8MPW; /* DETAIL - INDIVID STAFFING*/ 04450000
 EXEC SQL INCLUDE DSN8MPX; /* DETAIL - INDIVID ACTIVITY*/ 04460000
 EXEC SQL INCLUDE DSN8MPZ; /* DETAIL - INDIVIDUAL PROJ */ 04470000
END DSN8CP8;

```

### Referencia relacionada

["Ejemplos de aplicaciones en CICS"](#) en la página 1461

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

## DSN8CP3

ESTE MÓDULO LISTA LOS NÚMEROS DE TELÉFONO DE EMPLEADOS Y LOS ACTUALIZA SI LO DESEA.

```

DSN8CP3: PROC OPTIONS (MAIN);
/***
*
* MODULE NAME = DSN8CP3
*
* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
* PHONE APPLICATION
* CICS
* PL/I
*
* Licensed Materials - Property of IBM
* 5635-DB2
* (C) COPYRIGHT 1982, 2006 IBM Corp. All Rights Reserved.
*
* STATUS = Version 9
*
* FUNCTION = THIS MODULE LISTS EMPLOYEE PHONE NUMBERS AND
* UPDATES THEM IF DESIRED.
*
* NOTES =
* DEPENDENCIES = THREE CICS MAPS(DSECTS) ARE REQUIRED:
* DSN8MPMN, DSN8MPML, AND DSN8MPMU
* RESTRICTIONS = NONE
*
* MODULE TYPE = PL/I PROC OPTIONS(MAIN)
*

```

```

* PROCESSOR = DB2 PRECOMPILER, CICS TRANSLATOR, PL/I OPTIMIZER*
* MODULE SIZE = SEE LINKEDIT
* ATTRIBUTES = REENTRANT
*
* ENTRY POINT = DSN8CP3
* PURPOSE = SEE FUNCTION
* LINKAGE = INVOKED FROM CICS
*
* INPUT = PARAMETERS EXPLICITLY PASSED TO THIS FUNCTION:
* INPUT-MESSAGE:
*
* SYMBOLIC LABEL/NAME = DSN8CPNI
* DESCRIPTION = PHONE MENU 1 (SELECT)
*
* SYMBOLIC LABEL/NAME = DSN8CPLI
* DESCRIPTION = PHONE MENU 2 (LIST)
*
* SYMBOLIC LABEL/NAME = DSN8CPUI
* DESCRIPTION = PHONE MENU 3 (UPDATE)
*
* SYMBOLIC LABEL/NAME = VPHONE
* DESCRIPTION = VIEW OF TELEPHONE INFORMATION
*
* SYMBOLIC LABEL/NAME = VEMPLP
* DESCRIPTION = VIEW OF EMPLOYEE INFORMATION
*
* OUTPUT = PARAMETERS EXPLICITLY RETURNED:
* OUTPUT-MESSAGE:
*
* SYMBOLIC LABEL/NAME = DSN8CPNO
* DESCRIPTION = PHONE MENU 1 (SELECT)
*
* SYMBOLIC LABEL/NAME = DSN8CPL0
* DESCRIPTION = PHONE MENU 2 (LIST)
*
* SYMBOLIC LABEL/NAME = DSN8CPU0
* DESCRIPTION = PHONE MENU 3 (UPDATE)
*
* EXIT-NORMAL = RETURN CODE 0 NORMAL COMPLETION
*
* EXIT-ERROR =
*
* RETURN CODE = NONE
*
* ABEND CODES = NONE
*
*
* ERROR-MESSAGES =
* DSN8004I - EMPLOYEE SUCCESSFULLY UPDATED
* DSN8007E - EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE
* DSN8008I - NO EMPLOYEE FOUND IN TABLE
* DSN8057I - FURTHER ENTRIES IN TABLE - UPDATE POSSIBLE
* DSN8060E - SQL ERROR, RETURN CODE IS:
*
* EXTERNAL REFERENCES =
* ROUTINES/SERVICES =
* DSN8MPG - ERROR MESSAGE ROUTINE
*
* DATA-AREAS =
* IN_MESSAGE - VIA BMS, SEE INPUT PARAMETERS
* OUT_MESSAGE - VIA BMS, SEE OUTPUT PARAMTERS
* DSN8MPML - DECLARE FOR DSN8CPL CICS MAP
* DSN8MPMN - DECLARE FOR DSN8CPN CICS MAP
* DSN8MPMU - DECLARE FOR DSN8CPU CICS MAP
*
* CONTROL-BLOCKS =
* SQLCA - SQL COMMUNICATION AREA
*
* TABLES = NONE
*
*
* CHANGE-ACTIVITY =
* PQ92146 09/07/04 CHANGE DECLARED LENGTH OF BMS_IO FROM 32767 @01*
* TO 1408 TO STOP IBM2402I COMPILE-TIME ERROR @01*
*
*
* *PSEUDOCODE*
*
* PROCEDURE
* GET FIRST INPUT
* DO WHILE MORE INPUT
* GET REPORT HEADING

```

```

*
* CASE (ACTION)
*
* SUBCASE ('L')
* IF LASTNAME IS '*'
* LIST ALL EMPLOYEES
* ELSE
* IF LASTNAME CONTAINS '%'
* LIST EMPLOYEES GENERIC
* ELSE
* LIST EMPLOYEES SPECIFIC
* ENDSub
*
* SUBCASE ('U')
* UPDATE PHONENUMBER FOR EMPLOYEE
* WRITE CONFIRMATION MESSAGE
* OTHERWISE
* INVALID REQUEST
* ENDSUB
*
* GET NEXT INPUT
* ENDCASE
*
* IF SQL ERROR OCCURS THEN
* FORMAT ERROR MESSAGE
* ROLLBACK
* END
* END.
*/
/*
* MODULE NAME = DSN8CP3
* KDB0010
*
*/
1/***/
/* DECLARATION FOR INPUT / OUTPUT */
/***/
EXEC SQL INCLUDE DSN8MPMN ;
EXEC SQL INCLUDE DSN8MPML ;
EXEC SQL INCLUDE DSN8MPMU ;
ODCL 1 SUBMAPI(15) UNALIGNED BASED(ADDR(DSN8CU2I.NEWNO1L)),
2 NEWNOL FIXED BIN(15,0),
2 NEWNOA CHAR(1),
2 NEWNOD CHAR(4),
2 ENOL FIXED BIN(15,0),
2 ENOA CHAR(1),
2 ENOD CHAR(6);
ODCL 1 SUBMAPO(15) UNALIGNED BASED(ADDR(DSN8CL2I.FNAME1L)),
2 FNAMEL FIXED BIN(15,0),
2 FNAMEA CHAR(1),
2 FNAMED CHAR(12),
2 MINITL FIXED BIN(15,0),
2 MINITA CHAR(1),
2 MINITD CHAR(1),
2 LNAMEL FIXED BIN(15,0),
2 LNAMEA CHAR(1),
2 LNAMED CHAR(15),
2 PNOL FIXED BIN(15,0),
2 PNOA CHAR(1),
2 PNODE CHAR(4),
2 ENOL FIXED BIN(15,0),
2 ENOA CHAR(1),
2 ENOD CHAR(6),
2 WDEPTL FIXED BIN(15,0),
2 WDEPTA CHAR(1),
2 WDEPTD CHAR(3),
2 WNAMEL FIXED BIN(15,0),
2 WNAMEA CHAR(1),
2 WNAMED CHAR(31);

***** HOLDS BYTE-COUNT OF STORAGE ALLOCATED TO BMS OUTPUT AREA *****/
DCL BMS_LL BIN FIXED(31) INIT(STG(DSN8CL2I)); /*@EDVG*/
***** MASK/OVERLAY OF STORAGE ALLOCATED TO BMS OUTPUT AREA *****/
DCL BMS_IO CHAR(1408) BASED(ADDR(DSN8CL2I)); /*@01*/
1/***/
/* DECLARATION FOR PGM-LOGIC */
/***/
DCL FIRST BIT(1);

```

```

DCL PAGING BIT(1);
DCL OFLOW BIT(1);
DCL EMPLOYEE_NO CHAR (6);
DCL PHONE_NO CHAR (4);
DCL CHAR_SQLCODE CHAR (14);
DCL 1 CHAR_SQLSTR BASED(ADDR(CHAR_SQLCODE)),
 2 CHAR_BLNK CHAR(4),
 2 CHAR_SQLCOD CHAR(10);

1/***/
/* FIELDS SENT TO MESSAGE ROUTINE */
1/***/
DCL MODULE CHAR (7) INIT('DSN8CP3');
DCL OUTMSG CHAR (69);

DCL DSN8MPG EXTERNAL ENTRY;
1/***/
/* DECLARATION FOR SQL */
1/***/
0EXEC SQL INCLUDE SQLCA; /* SQL COMMUNICATION AREA */
 /* SQL DECLARATION FOR VIEW PHONE */

EXEC SQL DECLARE VPHONE TABLE
 (LASTNAME VARCHAR(15) ,
 FIRSTNAME VARCHAR(12) ,
 MIDDLEINITIAL CHAR(1) ,
 PHONENUMBER CHAR(4) ,
 EMPLOYEENUMBER CHAR(6) ,
 DEPTNUMBER CHAR(3) NOT NULL,
 DEPTNAME VARCHAR(36) NOT NULL);
 /* STUCTURE FOR PHONE RECORD */

DCL 1 PPHONE,
 2 LASTNAME CHAR (15) VAR,
 2 FIRSTNAME CHAR (12) VAR,
 2 MIDDLEINITIAL CHAR (1),
 2 PHONENUMBER CHAR (4),
 2 EMPLOYEENUMBER CHAR (6),
 2 DEPTNUMBER CHAR (3),
 2 DEPTNAME CHAR (36) VAR;
 /* SQL DECLARATION FOR VIEW VEMPLP */

EXEC SQL DECLARE VEMPLP TABLE
 (EMPLOYEENUMBER CHAR(6) ,
 PHONENUMBER CHAR(4)); /* STRUCTURE FOR PEMPLP RECORD */

DCL 1 PEMP,
 2 EMPLOYEENUMBER CHAR (6),
 2 PHONENUMBER CHAR (4);

1/***/
/* SQL CURSORS */
1/***/

EXEC SQL DECLARE TELE1 CURSOR FOR
 SELECT *
 FROM VPHONE;

EXEC SQL DECLARE TELE2 CURSOR FOR
 SELECT *
 FROM VPHONE
 WHERE LASTNAME LIKE :DSN8CN2I.LNAMEI
 AND FIRSTNAME LIKE :DSN8CN2I.FNAMEI;

EXEC SQL DECLARE TELE3 CURSOR FOR
 SELECT *
 FROM VPHONE
 WHERE LASTNAME = :DSN8CN2I.LNAMEI
 AND FIRSTNAME LIKE :DSN8CN2I.FNAMEI;

1/***/
/* SQL RETURN CODE HANDLING */
1/***/

EXEC SQL WHENEVER SQLERROR GOTO P3_DBERROR;
EXEC SQL WHENEVER SQLWARNING GOTO P3_DBERROR;
EXEC SQL WHENEVER NOT FOUND CONTINUE;

1/***/
/* MAIN PROGRAM ROUTINE */
1/***/
 /* SET HANDLE CONDITIONS */
 /* CICS HANDLE CONDITION MAPFAIL (P3_MAPFAIL); */
 /* CICS HANDLE AID CLEAR (P3_CLEAR); */

```

```

/***** CLEAR THE BMS OUTPUT AREA *****/
SUBSTR(BMS_IO,1,BMS_LL) = LOW(BMS_LL) ; /*@EDVG*/
P3_START:
FIRST = '1'B; /*INITIALIZE FIRST BIT */
OFLW = '0'B; /*INITIALIZE OVERFLOW BIT*/
SELECT (EIBTRNID); /* SELECT ACTION */
WHEN ('D8PT') DO; /* LIST EMPLOYEES */

 /* GET INPUT FROM SCREEN */
EXEC CICS RECEIVE MAP('DSN8CN2') MAPSET('DSN8CPN');

1/*****
/* LIST ALL EMPLOYEES
*/
*****/

IF DSN8CN2I.LNAMEI = '*' THEN /*LIST ALL EMPLOYEES */
DO;

EXEC SQL OPEN TELE1; /* OPEN CURSOR */
EXEC SQL FETCH TELE1 /* GET FIRST RECORD */
INTO :PPHONE;

I = 0; /* INITIALIZE COUNTER */

IF SQLCODE = 100 THEN /* NO EMPLOYEE FOUND */
DO; /* PRINT ERROR MESSAGE */
CALL DSN8MPG (MODULE, '008I', OUTMSG);
DSN8CN3I.EMSGI = OUTMSG;
EXEC CICS SEND MAP('DSN8CN3') MAPSET('DSN8CPN') ERASE;
EXEC CICS SEND MAP('DSN8CN2') MAPSET('DSN8CPN');
END;

DO WHILE (SQLCODE = 0); /*LIST EMPLOYEES*/
I = I + 1; /* INCREMENT COUNTER*/
PAGING = '1'B;
SUBMAPO.FNAMED(I) = PPHONE.FIRSTNAME;
SUBMAPO.MINITD(I) = PPHONE.MIDDLEINITIAL;
SUBMAPO.LNAMED(I) = PPHONE.LASTNAME;
SUBMAPO.PNOD(I) = PPHONE.PHONENUMBER;
SUBMAPO.ENOD(I) = PPHONE.EMPLOYEENUMBER;
SUBMAPO.WDEPTD(I) = PPHONE.DEPTNUMBER;
SUBMAPO.WNAMED(I) = PPHONE.DEPTNAME;

IF I = 15 THEN /*POSSIBLE OVERFLOW */
DO; /* PRINT ERROR MESSAGE*/
OFLW = '1'B;
CALL DSN8MPG (MODULE, '057I', OUTMSG);
DSN8CL30.EMSGO = OUTMSG;
END;

IF I = 15 THEN LEAVE; /* SCREEN IS FILLED */

EXEC SQL FETCH TELE1 /* GET NEXT RECORD */
INTO :PPHONE;
END; /* END OF WHILE */

EXEC SQL CLOSE TELE1; /* CLOSE CURSOR */
END; /* END OF IF */
1/*****
/* LIST GENERIC EMPLOYEES
*/
*****/

ELSE /* SELECT EMPLOYEES BY NAME*/
DO; /* SEARCH ON PART OF NAME? */
IF DSN8CN2I.LNAMEI = 0 THEN /* IS LAST NAME BLANK? */
DSN8CN2I.LNAMEI = '%%%%%%%%%%%%%%'; /* YES, ANYTHING */
IF INDEX(DSN8CN2I.LNAMEI,'%') > 0 THEN /* IS IT A PATTERN*/
DO; /* YES, SEARCH ON */
 /* PART OF LAST NAME */
DSN8CN2I.LNAMEI = TRANSLATE(DSN8CN2I.LNAMEI,'%', ' ');

 /*AND OPTIONAL FIRST NAME*/
IF DSN8CN2I.FNAMEI = 0 THEN
DSN8CN2I.FNAMEI = '%%%%%%%%%%%%%%';
ELSE
DSN8CN2I.FNAMEI = TRANSLATE(DSN8CN2I.FNAMEI,'%', ' ');

EXEC SQL OPEN TELE2; /* OPEN CURSOR */

```

```

EXEC SQL FETCH TELE2 /* GET FIRST RECORD */
INTO :PPHONE;

I = 0; /* INITIALIZE COUNTER */

IF SQLCODE = 100 THEN /* EMPLOYEE NOT FOUND */
DO; /* PRINT ERROR MESSAGE */
CALL DSN8MPG (MODULE, '008I', OUTMSG);
DSN8CN3I.EMSGI = OUTMSG;
EXEC CICS SEND MAP('DSN8CN3') MAPSET('DSN8CPN') ERASE;
EXEC CICS SEND MAP('DSN8CN2') MAPSET('DSN8CPN');
END;

DO WHILE (SQLCODE = 0); /* LIST EMPLOYEES */
I = I + 1; /* INCREMENT COUNTER */
PAGING = '1'B;
SUBMAPO.FNAMED(I) = PPHONE.FIRSTNAME;
SUBMAPO.MINITD(I) = PPHONE.MIDDLEINITIAL;
SUBMAPO.LNAMED(I) = PPHONE.LASTNAME;
SUBMAPO.PNOD(I) = PPHONE.PHONENUMBER;
SUBMAPO.ENOD(I) = PPHONE.EMPLOYEENUMBER;
SUBMAPO.WDEPTD(I) = PPHONE.DEPTNUMBER;
SUBMAPO.WNAMED(I) = PPHONE.DEPTNAME;

IF I = 15 THEN /* POSSIBLE OVERFLOW */
DO; /* PRINT ERROR MESSAGE */
OFLW = '1'B;
CALL DSN8MPG (MODULE, '057I', OUTMSG);
DSN8CL30.EMSG0 = OUTMSG;
END;

IF I = 15 THEN LEAVE; /* SCREEN IS FILLED */

EXEC SQL FETCH TELE2 /* GET NEXT RECORD */
INTO :PPHONE;
END; /* END OF DO WHILE */

EXEC SQL CLOSE TELE2; /* CLOSE CURSOR */
END; /* END OF IF */

1/***/
/* LIST SPECIFIC EMPLOYEE(S) */
/***/
ELSE /* SEARCH ON LAST NAME */
DO; /* AND OPTIONALLY FIRST NAME*/
IF DSN8CN2I.FNAMEL = 0 THEN
 DSN8CN2I.FNAMEI = '%/%/%/%/%/%/%/%';
ELSE
 DSN8CN2I.FNAMEI = TRANSLATE(DSN8CN2I.FNAMEI, '%', ' ');

EXEC SQL OPEN TELE3; /* OPEN CURSOR */
EXEC SQL FETCH TELE3 /* GET FIRST RECORD */
INTO :PPHONE;

I = 0; /* INITIALIZE COUNTER */

IF SQLCODE = 100 THEN /* EMPLOYEE NOT FOUND */
DO; /* PRINT ERROR MESSAGE */
CALL DSN8MPG (MODULE, '008I', OUTMSG);
DSN8CN3I.EMSGI = OUTMSG;
EXEC CICS SEND MAP('DSN8CN3') MAPSET('DSN8CPN') ERASE;
EXEC CICS SEND MAP('DSN8CN2') MAPSET('DSN8CPN');
END;

DO WHILE (SQLCODE = 0); /* LIST EMPLOYEE(S) */
I = I + 1; /* INCREMENT COUNTER */
PAGING = '1'B;
SUBMAPO.FNAMED(I) = PPHONE.FIRSTNAME;
SUBMAPO.MINITD(I) = PPHONE.MIDDLEINITIAL;
SUBMAPO.LNAMED(I) = PPHONE.LASTNAME;
SUBMAPO.PNOD(I) = PPHONE.PHONENUMBER;
SUBMAPO.ENOD(I) = PPHONE.EMPLOYEENUMBER;
SUBMAPO.WDEPTD(I) = PPHONE.DEPTNUMBER;
SUBMAPO.WNAMED(I) = PPHONE.DEPTNAME;

IF I = 15 THEN /* POSSIBLE OVERFLOW */
DO; /* PRINT ERROR MESSAGE */
OFLW = '1'B;
CALL DSN8MPG (MODULE, '057I', OUTMSG);
DSN8CL30.EMSG0 = OUTMSG;

```

```

 END;

 IF I = 15 THEN LEAVE; /* SCREEN IS FILLED */
 EXEC SQL FETCH TELE3 /* GET NEXT RECORD */
 INTO :PPHONE;
 END; /* END OF DO WHILE */

 EXEC SQL CLOSE TELE3; /* CLOSE CURSOR */
 END; /* END OF ELSE */
 END; /* END OF IF */

IF PAGING THEN
DO;
PAGING = '0'B;
EXEC CICS SEND MAP ('DSN8CL1') MAPSET('DSN8CPL') ERASE
ACCUM PAGING;
EXEC CICS SEND MAP ('DSN8CL2') MAPSET('DSN8CPL')
ACCUM PAGING;

IF OFLOW THEN
DO;
OFLOW = '0'B;
EXEC CICS SEND MAP ('DSN8CL3') MAPSET('DSN8CPL')
ACCUM PAGING;
END;

EXEC CICS SEND PAGE;
EXEC CICS RETURN TRANSID('D8PU');
END; /* END OF IF */

ELSE EXEC CICS RETURN TRANSID ('D8PT');
END; /* END OF WHEN */
 /* CHANGE ERROR HANDLING */
 /* FOR UPDATE */

EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
1/***
/* UPDATES PHONE NUMBERS FOR EMPLOYEES
*****/WHEN ('D8PU') DO; /* TELEPHONE UPDATE */

 /* GET UPDATED DATA */
EXEC CICS RECEIVE MAP('DSN8CU2') MAPSET('DSN8CPU');
 /* FIND WHICH NUMBERS HAVE */
 /* BEEN UPDATED */
DSN8CN3I.EMSGI = ''; /* SET IN CASE NO UPDATES */

DO I = 1 TO 15;
 IF SUBMAPI.NEWNOL(I) = 0 THEN; /* NO UPDATE ON THIS LINE */
 ELSE
 DO;
 EMPLOYEE_NO = SUBMAPI.ENOD(I);
 PHONE_NO = SUBMAPI.NEWNOD(I);

 EXEC SQL UPDATE VEMPLP /* PERFORM UPDATE */
 SET PHONENUMBER = :PHONE_NO
 WHERE EMPLOYEENUMBER = :EMPLOYEE_NO;

 IF SQLCODE ^= 0 THEN
 DO; /* UPDATE FAILED */
 /* PRINT ERROR MESSAGE */
 CALL DSN8MPG (MODULE, '007E', OUTMSG);
 DSN8CU3I.EMSGI = OUTMSG;
 EXEC CICS SEND MAP('DSN8CU3') MAPSET('DSN8CPU');
 GOTO P3_DBERROR2;
 END; /* UPDATE SUCCESSFUL */
 /* PRINT CONFIRMATION */
 /* MESSAGE */
 ELSE
 DO;
 CALL DSN8MPG (MODULE, '004I', OUTMSG);
 DSN8CN3I.EMSGI = OUTMSG;
 END;
 END; /* END ELSE */
 END; /* END FOR */

 EXEC CICS SEND MAP('DSN8CN3') MAPSET('DSN8CPN') ERASE;
 EXEC CICS SEND MAP('DSN8CN2') MAPSET('DSN8CPN') ;
 EXEC CICS RETURN TRANSID('D8PT');

END; /* END WHEN */

```

```

 OTHERWISE GOTO P3_CLEAR; /* WRONG TX CODE */
END; /* END SELECT */
GOTO P3_END;
P3_MAPFAIL: /* D8PT FROM UNFORMATTED */
 /* SCREEN */
 /* MAP ONLY */
EXEC CICS SEND MAP('DSN8CN2') MAPONLY MAPSET('DSN8CPN') ERASE;
EXEC CICS RETURN TRANSID('D8PT');
1/***
/* SQL ERROR HANDLING */
/*****
P3_DBERROR: /* SQL ERROR HANDLING */
CALL DSN8MPG (MODULE, '060E', OUTMSG);
CHAR_SQLCODE = SQLCODE;
DSN8CN3I.EMSGI = OUTMSG||CHAR_SQLCOD;
EXEC CICS SEND MAP('DSN8CN3') MAPSET('DSN8CPN') ;
P3_DBERROR2:
EXEC CICS SEND PAGE ; /* PERFORM ROLLBACK */
EXEC CICS SYNCPOINT ROLLBACK;
EXEC CICS RETURN;
P3_CLEAR: /* CLEAR SCREEN */
EXEC CICS SEND CONTROL FREEKB ;
EXEC CICS RETURN;
/*****
P3_END: /* PROGRAM END */
END DSN8CP3;

```

### Referencia relacionada

[“Ejemplos de aplicaciones en CICS” en la página 1461](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

## DSNTEJ5C

ESTE JCL REALIZA LA CONFIGURACIÓN DE LA FASE 5 PARA LAS APLICACIONES DE MUESTRA EN SITIOS CON COBOL.

```

//*****
//** NAME = DSNTEJ5C
//*
//** DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//** PHASE 5
//** COBOL, CICS
//*
//** Licensed Materials - Property of IBM
//** 5650-DB2
//** (C) COPYRIGHT 1982, 2016 IBM Corp. All Rights Reserved.
//*
//** STATUS = Version 12
//*
//** FUNCTION = THIS JCL PERFORMS THE PHASE 5 SETUP FOR THE SAMPLE
//** APPLICATIONS AT SITES WITH COBOL. IT PREPARES THE
//** COBOL CICS PROGRAM.
//*
//** RUN THIS JOB ANYTIME AFTER PHASE 2.
//*
//** CHANGE ACTIVITY =
//** 08/18/2014 Single-phase migration s21938_inst1 s21938
//*
//*****JOBLIB DD DSN=DSN!!0.SDSNEXIT,DISP=SHR
// DD DSN=DSN!!0.SDSNLOAD,DISP=SHR
// DD DSN=CICSTS.SDFHLOAD,DISP=SHR
//*
//** STEP 1: CREATE CICS LOGICAL MAP
//MAPG EXEC DFHASMVS,PARM='DECK,NOBJECT,SYSPARM(DSECT)',
// OUTC='*'
//SYSPUNCH DD DSN=DSN!!0.SRCLIB.DATA(DSN8MCMG),
// DISP=OLD
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CCG),
// DISP=SHR
//*
//** STEP 2: CREATE CICS LOGICAL MAP
//MAPD EXEC DFHASMVS,PARM='DECK,NOBJECT,SYSPARM(DSECT)',
// COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=DSN!!0.SRCLIB.DATA(DSN8MCMG),
// DISP=OLD
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CCD),
// DISP=SHR
//
```

```

//*
//** STEP 3: PREPARE CICS COBOL PROGRAMS
//DSNH EXEC PGM=IKJEFT01,COND=(4,LT),DYNAMNBR=50
//SYSTSPRT DD SYSOUT=*
//SYSTERM DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSPROC DD DSN=DSN!!0.SDSNCLST,DISP=SHR
//SYSTSIN DD *
%DSNH INPUT(''DSN!!0.SSDNSAMP(DSN8CC0)''') +
PLIB(''DSN!!0.SRCLIB.DATA'') +
P2LIB(''DSN!!0.SSDNSAMP'') +
TERM(LEAVE) PRINT(LEAVE) SOURCE(NO) XREF(NO) +
HOST(IBMCOB) RUN(CICS) BIND(NO) +
DELIM(APOST) SQLDELIM(APOSTSQL) +
DBRMLIB(''DSN!!0.DBRMLIB.DATA'') +
PRELINK(YES) +
LLIB(''DSN!!0.RUNLIB.LOAD'') +
COBICOMP(''IGY.V!R!M!.SIGYCOMP(IGYCRCTL)'') +
COPTION(''NOSEQUENCE,QUOTE,RENT,PGMNAME(LONGUPPER)'') +
LOPTION('LIST,XREF,MAP,RENT') +
STDSQL(NO) +
XLIB(''DSN!!0.SDSNLOAD'') +
LOAD(''DSN!!0.RUNLIB.LOAD'')
%DSNH INPUT(''DSN!!0.SSDNSAMP(DSN8CC1)''') +
PLIB(''DSN!!0.SRCLIB.DATA'') +
P2LIB(''DSN!!0.SSDNSAMP'') +
TERM(LEAVE) PRINT(LEAVE) SOURCE(NO) XREF(NO) +
HOST(IBMCOB) RUN(CICS) BIND(NO) +
DELIM(APOST) SQLDELIM(APOSTSQL) +
DBRMLIB(''DSN!!0.DBRMLIB.DATA'') +
PRELINK(YES) +
LLIB(''DSN!!0.RUNLIB.LOAD'') +
COBICOMP(''IGY.V!R!M!.SIGYCOMP(IGYCRCTL)'') +
COPTION(''NOSEQUENCE,QUOTE,RENT,PGMNAME(LONGUPPER)'') +
LOPTION('LIST,XREF,MAP,RENT') +
STDSQL(NO) +
XLIB(''DSN!!0.SDSNLOAD'') +
LOAD(''DSN!!0.RUNLIB.LOAD'')
%DSNH INPUT(''DSN!!0.SSDNSAMP(DSN8CC2)''') +
PLIB(''DSN!!0.SRCLIB.DATA'') +
P2LIB(''DSN!!0.SSDNSAMP'') +
TERM(LEAVE) PRINT(LEAVE) SOURCE(NO) XREF(NO) +
HOST(IBMCOB) RUN(CICS) BIND(NO) +
DELIM(APOST) SQLDELIM(APOSTSQL) +
DBRMLIB(''DSN!!0.DBRMLIB.DATA'') +
PRELINK(YES) +
LLIB(''DSN!!0.RUNLIB.LOAD'') +
COBICOMP(''IGY.V!R!M!.SIGYCOMP(IGYCRCTL)'') +
COPTION(''NOSEQUENCE,QUOTE,RENT,PGMNAME(LONGUPPER)'') +
LOPTION('LIST,XREF,MAP,RENT') +
STDSQL(NO) +
XLIB(''DSN!!0.SDSNLOAD'') +
LOAD(''DSN!!0.RUNLIB.LOAD'')
*/
//*
//** STEP 4: BIND THE PROGRAM
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 GRANT BIND, EXECUTE ON PLAN DSN8CC0
 TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE (DSN8CC!!) MEMBER(DSN8CC0) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE (DSN8CC!!) MEMBER(DSN8CC1) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE (DSN8CC!!) MEMBER(DSN8CC2) APPLCOMPAT(V!!R1) +
ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8CC0) PKLIST(DSN8CC!!.*)
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END
//*
//** STEP 5: CREATE THE CICS BMS PHYSICAL MAP

```

```

//MAPGP EXEC DFHASMVS,COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=&&TEMP,
// DISP=(,PASS),
// UNIT=SYSDA,SPACE=(1024,(100,10)),
// DCB=(RECFM=F,BLKSIZE=80)
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CCG),
// DISP=SHR
///*
//** STEP 6: LINKEDIT THE CICS BMS PHYSICAL MAP
//MAPGL EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
// DD *
NAME DSN8CCG(R)
///*
//** STEP 7: CREATE THE CICS BMS PHYSICAL MAP
//MAPDP EXEC DFHASMVS,COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=&&TEMP,
// DISP=(,PASS),
// UNIT=SYSDA,SPACE=(1024,(100,10)),
// DCB=(RECFM=F,BLKSIZE=80)
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CCD),
// DISP=SHR
///*
//** STEP 8: LINKEDIT THE CICS BMS PHYSICAL MAP
//MAPDL EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
// DD *
NAME DSN8CCD(R)

```

## Referencia relacionada

[“Ejemplos de aplicaciones en CICS” en la página 1461](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.

## DSNTEJ5P

ESTE JCL REALIZA LA CONFIGURACIÓN DE LA FASE 5 PARA LAS APLICACIONES DE MUESTRA EN SITIOS CON PL/I.

```

//***** ****
//* NAME = DSNTEJ5P
//*
//* DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION
//* PHASE 5
//* PL/I, CICS
//*
//* Licensed Materials - Property of IBM
//* 5650-DB2
//* (C) COPYRIGHT 1982, 2016 IBM Corp. All Rights Reserved.
//*
//* STATUS = Version 12
//*
//* FUNCTION = THIS JCL PERFORMS THE PHASE 5 SETUP FOR THE SAMPLE
//* APPLICATIONS AT SITES WITH PL/I. IT PREPARES THE
//* PL/I CICS PROGRAM.
//*
//* RUN THIS JOB ANYTIME AFTER PHASE 2.
//*
//* CHANGE ACTIVITY =
//* 08/18/2014 Single-phase migration s21938_inst1 s21938
//*
//***** ****
//*
//JOBLIB DD DISP=SHR,DSN=CICSTS.SDFHLOAD
// DD DISP=SHR,DSN=DSN!!0.SDSNLOAD
//*
//* STEP 1: CREATE CICS BMS LOGICAL MAPS
//PH05PS01 EXEC DFHASMVS,PARM='DECK,NOBJECT,SYSPARM(DSECT)',
// OUTC='*'
//SYSPUNCH DD DSN=DSN!!0.SRCLIB.DATA(DSN8MPMG),
// DISP=OLD

```

```

//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CPG),
// DISP=SHR
///*
//** STEP 2: CREATE CICS BMS LOGICAL MAPS
//PH05PS02 EXEC DFHASMVS,PARM='DECK,NOBJECT,SYSPARM(DSECT)',
// COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=DSN!!0.SRCLIB.DATA(DSN8MPMD),
// DISP=OLD
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CPD),
// DISP=SHR
///*
//** STEP 3: CREATE CICS BMS LOGICAL MAPS
//PH05PS03 EXEC DFHASMVS,PARM='DECK,NOBJECT,SYSPARM(DSECT)',
// COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=DSN!!0.SRCLIB.DATA(DSN8MPMN),
// DISP=OLD
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CPN),
// DISP=SHR
///*
//** STEP 4: CREATE CICS BMS LOGICAL MAPS
//PH05PS04 EXEC DFHASMVS,PARM='DECK,NOBJECT,SYSPARM(DSECT)',
// COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=DSN!!0.SRCLIB.DATA(DSN8MPML),
// DISP=OLD
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CPL),
// DISP=SHR
///*
//** STEP 5: CREATE CICS BMS LOGICAL MAPS
//PH05PS05 EXEC DFHASMVS,PARM='DECK,NOBJECT,SYSPARM(DSECT)',
// COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=DSN!!0.SRCLIB.DATA(DSN8MPMU),
// DISP=OLD
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CPU),
// DISP=SHR
///*
//** STEP 6: CICS TRANSLATE FOR SQL 0 PART
//PH05PS06 EXEC PGM=DFHEPP1$,COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPUNCH DD DSN=&&CICSOUTO,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(400,(100,100)),
// DCB=BLKSIZE=400
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CP0),
// DISP=SHR
///*
//** STEP 7: PREPARE SQL 0 PART
//PH05PS07 EXEC DSNHPLI,MEM=DSN8CP0,
// COND=(4,LT),
// PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',
// PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=&&CICSOUTO,DISP=(OLD,DELETE)
//PPLI.SYSLIB DD DSN=CICSTS.SDFHPL1,
// DISP=SHR
//PC.DBRMLIB DD DSN=DSN!!0.DBRMLIB.DATA(DSN8CP0),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT0
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT0
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP0),
// DISP=SHR
//LKED.SYSIN DD DUMMY
///*
//** STEP 8: CICS TRANSLATE FOR SQL 1 PART
//PH05PS08 EXEC PGM=DFHEPP1$,COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPUNCH DD DSN=&&CICSOUT1,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(400,(100,100)),
// DCB=BLKSIZE=400
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CP1),
// DISP=SHR
///*
//** STEP 9: PREPARE SQL 1 PART
//PH05PS09 EXEC DSNHPLI,MEM=DSN8CP1,

```

```

// COND=(4,LT),
// PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',
// PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=&&CICCSOUT1,DISP=(OLD,DELETE)
//PPLI.SYSLIB DD DSN=CICSTS.SDFHPL1,
// DISP=SHR
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8CP1),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT1
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SDSENSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT1
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP1),
// DISP=SHR
//LKED.SYSIN DD DUMMY
///*
//** STEP 10: CICS TRANSLATE FOR SQL 2 PART
//PH05PS10 EXEC PGM=DFHEPP1$,COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPUNCH DD DSN=&&CICCSOUT2,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(400,(100,100)),
// DCB=BLKSIZE=400
//SYSIN DD DSN=DSN!!0.SDSENSAMP(DSN8CP2),
// DISP=SHR
///*
//** STEP 11: PREPARE SQL 2 PART
//PH05PS11 EXEC DSNHPLI,MEM=DSN8CP2,
// COND=(4,LT),
// PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',
// PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=&&CICCSOUT2,DISP=(OLD,DELETE)
//PPLI.SYSLIB DD DSN=CICSTS.SDFHPL1,
// DISP=SHR
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8CP2),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT2
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SDSENSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT2
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP2),
// DISP=SHR
//LKED.SYSIN DD DUMMY
///*
//** STEP 12: CICS TRANSLATE FOR TELEPHONE APPLICATION
//PH05PS12 EXEC PGM=DFHEPP1$,COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPUNCH DD DSN=&&CICCSOUT3,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(400,(100,100)),
// DCB=BLKSIZE=400
//SYSIN DD DSN=DSN!!0.SDSENSAMP(DSN8CP3),
// DISP=SHR
///*
//** STEP 13: PREPARE TELEPHONE APPLICATION
//PH05PS13 EXEC DSNHPLI,MEM=DSN8CP3,
// COND=(4,LT),
// PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',
// PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=&&CICCSOUT3,DISP=(OLD,DELETE)
//PPLI.SYSLIB DD DSN=CICSTS.SDFHPL1,
// DISP=SHR
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8CP3),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT3
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR

```

```

// DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT3
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP3),
// DISP=SHR
//LKED.SYSIN DD DUMMY
///*
//** STEP 14: CREATE CICS BMS LOGICAL MAPS
//PH05PS14 EXEC DFHASMVS,PARM='DECK,NOBJECT,SYSPARM(DSECT) ',
// COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=DSN!!0.SRCLIB.DATA(DSN8MPMF),
// DISP=OLD
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CPF),
// DISP=SHR
///*
//** STEP 15: CREATE CICS BMS LOGICAL MAPS
//PH05PS15 EXEC DFHASMVS,PARM='DECK,NOBJECT,SYSPARM(DSECT) ',
// COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=DSN!!0.SRCLIB.DATA(DSN8MPME),
// DISP=OLD
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CPE),
// DISP=SHR
///*
//** STEP 16: CICS TRANSLATE FOR SQL 0 PART
//PH05PS16 EXEC PGM=DFHEPP1$,COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPUNCH DD DSN=&&CICSOOUT6,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(400,(100,100)),
// DCB=BLKSIZE=400
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CP6),
// DISP=SHR
///*
//** STEP 17: PREPARE SQL 0 PART
//PH05PS17 EXEC DSNHPLI,MEM=DSN8CP6,
// COND=(4,LT),
// PARM.PPLI='MACRO,NOSNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',
// PARM.PLI=('NOPT,SOURCE,OBJECT,Margins(2,72,0)',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=&&CICSOOUT6,DISP=(OLD,DELETE)
//PPLI.SYSLIB DD DSN=CICSTS.SDFHPL1,
// DISP=SHR
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8CP6),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT6
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SDSNSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT6
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP6),
// DISP=SHR
//LKED.SYSIN DD DUMMY
///*
//** STEP 18: CICS TRANSLATE FOR SQL 1 PART
//PH05PS18 EXEC PGM=DFHEPP1$,COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPUNCH DD DSN=&&CICSOOUT7,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(400,(100,100)),
// DCB=BLKSIZE=400
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CP7),
// DISP=SHR
///*
//** STEP 19: PREPARE SQL 1 PART
//PH05PS19 EXEC DSNHPLI,MEM=DSN8CP7,
// COND=(4,LT),
// PARM.PPLI='MACRO,NOSNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',
// PARM.PLI=('NOPT,SOURCE,OBJECT,Margins(2,72,0)',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=&&CICSOOUT7,DISP=(OLD,DELETE)
//PPLI.SYSLIB DD DSN=CICSTS.SDFHPL1,
// DISP=SHR
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8CP7),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT7

```

```

//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SSDNSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT7
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP7),
// DISP=SHR
//LKED.SYSIN DD DUMMY
//*
//* STEP 20: CICS TRANSLATE FOR SQL 2 PART
//PH05PS20 EXEC PGM=DFHEPP1$,COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSPUNCH DD DSN=&&CICCSOUT8,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(400,(100,100)),
// DCB=BLKSIZE=400
//SYSIN DD DSN=DSN!!0.SSDNSAMP(DSN8CP8),
// DISP=SHR
//*
//* STEP 21: PREPARE SQL 2 PART
//PH05PS21 EXEC DSNHPLI,MEM=DSN8CP8,
// COND=(4,LT),
// PARM.PPLI='MACRO,NOSYNTAX,MDECK,NOINSOURCE,NOSOURCE',
// PARM.PC='HOST(PLI),CCSID(37),NOGRAPHIC,STDSQL(NO)',
// PARM.PLI=('NOPT,SOURCE,OBJECT,MARGINS(2,72,0)',
// 'LIMITS(EXTNAME(7)),OPTIONS','SYSTEM(CICS)'),
// PARM.LKED='NCAL'
//PPLI.SYSIN DD DSN=&&CICCSOUT8,DISP=(OLD,DELETE)
//PPLI.SYSLIB DD DSN=CICSTS.SDFHPL1,
// DISP=SHR
//PC.DBRLMLIB DD DSN=DSN!!0.DBRLMLIB.DATA(DSN8CP8),
// DISP=SHR
//PC.SYSCIN DD DSN=&&DSNHOUT8
//PC.SYSLIB DD DSN=DSN!!0.SRCLIB.DATA,
// DISP=SHR
// DD DSN=DSN!!0.SSDNSAMP,
// DISP=SHR
//PLI.SYSIN DD DSN=&&DSNHOUT8
//LKED.SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD(DSN8CP8),
// DISP=SHR
//LKED.SYSIN DD DUMMY
//*
//* STEP 22: LINKEDIT PROGRAMS TOGETHER
//PH05PS22 EXEC PGM=IEWL,PARM='LIST,XREF,LET',COND=(4,LT)
//SYSLIB DD DSN=CEE.V!R!M!.SCEELKED,
// DISP=SHR
// DD DSN=DSN!!0.SSDNLOAD,
// DISP=SHR
// DD DSN=CICSTS.SDFHPL1,
// DISP=SHR
// DD DSN=CICSTS.SDFHLOAD,
// DISP=SHR
//SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD,
// DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(50,50))
//SYSLIN DD *
INCLUDE SYSLIB(CEESTART)
INCLUDE SYSLIB(CEESG010)
INCLUDE SYSLIB(DFHELII)
INCLUDE SYSLIB(DSNCLI)
REPLACE PLISTART
CHANGE PLIMAIN(CEEMAIN)
INCLUDE SYSLMOD(DSN8CP0)
INCLUDE SYSLMOD(DSN8MPG)
ORDER CEESTART
ENTRY CEESTART
NAME DSN8CP0(R)
INCLUDE SYSLIB(CEESTART)
INCLUDE SYSLIB(CEESG010)
INCLUDE SYSLIB(DFHELII)
INCLUDE SYSLIB(DSNCLI)
REPLACE PLISTART
CHANGE PLIMAIN(CEEMAIN)
INCLUDE SYSLMOD(DSN8CP1)
INCLUDE SYSLMOD(DSN8MPG)
ORDER CEESTART
ENTRY CEESTART
NAME DSN8CP1(R)
INCLUDE SYSLIB(CEESTART)

```

```

INCLUDE SYSLIB(CEESG010)
INCLUDE SYSLIB(DFHELII)
INCLUDE SYSLIB(DSNCLI)
REPLACE PLISTART
CHANGE PLIMAIN(CEEMAIN)
INCLUDE SYSLMOD(DSN8CP2)
INCLUDE SYSLMOD(DSN8MPG)
ORDER CEESTART
ENTRY CEESTART
 NAME DSN8CP2(R)
INCLUDE SYSLIB(CEESTART)
INCLUDE SYSLIB(CEESG010)
INCLUDE SYSLIB(DFHELII)
INCLUDE SYSLIB(DSNCLI)
REPLACE PLISTART
CHANGE PLIMAIN(CEEMAIN)
INCLUDE SYSLMOD(DSN8CP3)
INCLUDE SYSLMOD(DSN8MPG)
ORDER CEESTART
ENTRY CEESTART
 NAME DSN8CP3(R)
INCLUDE SYSLIB(CEESTART)
INCLUDE SYSLIB(CEESG010)
INCLUDE SYSLIB(DFHELII)
INCLUDE SYSLIB(DSNCLI)
REPLACE PLISTART
CHANGE PLIMAIN(CEEMAIN)
INCLUDE SYSLMOD(DSN8CP6)
INCLUDE SYSLMOD(DSN8MPG)
ORDER CEESTART
ENTRY CEESTART
 NAME DSN8CP6(R)
INCLUDE SYSLIB(CEESTART)
INCLUDE SYSLIB(CEESG010)
INCLUDE SYSLIB(DFHELII)
INCLUDE SYSLIB(DSNCLI)
REPLACE PLISTART
CHANGE PLIMAIN(CEEMAIN)
INCLUDE SYSLMOD(DSN8CP7)
INCLUDE SYSLMOD(DSN8MPG)
ORDER CEESTART
ENTRY CEESTART
 NAME DSN8CP7(R)
INCLUDE SYSLIB(CEESTART)
INCLUDE SYSLIB(CEESG010)
INCLUDE SYSLIB(DFHELII)
INCLUDE SYSLIB(DSNCLI)
REPLACE PLISTART
CHANGE PLIMAIN(CEEMAIN)
INCLUDE SYSLMOD(DSN8CP8)
INCLUDE SYSLMOD(DSN8MPG)
ORDER CEESTART
ENTRY CEESTART
 NAME DSN8CP8(R)

/*
/* STEP 23: BIND PROGRAMS
//PH05PS23 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB DD DISP=SHR,DSN=DSN!0.DBRMLIB.DATA
//SYSUDUMP DD SYSOUT=*
//SYSTSPT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
 SET CURRENT SQLID = 'SYSADM';
 GRANT BIND, EXECUTE ON PLAN DSN8CP0, DSN8CQ0, DSN8CH0
 TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DSN)
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP0) APPLCOMPAT(V!!R1) +
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP1) APPLCOMPAT(V!!R1) +
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP2) APPLCOMPAT(V!!R1) +
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP3) APPLCOMPAT(V!!R1) +
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP6) APPLCOMPAT(V!!R1) +
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP7) APPLCOMPAT(V!!R1) +
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PACKAGE(DSN8CP!!) MEMBER(DSN8CP8) APPLCOMPAT(V!!R1) +
 ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8CP0) +

```

```

PKLIST(DSN8CP0!.DSN8CP0, +
 DSN8CP0!.DSN8CP1, +
 DSN8CP0!.DSN8CP2) +
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8CQ0) +
PKLIST(DSN8CP6, +
 DSN8CP6!.DSN8CP7, +
 DSN8CP6!.DSN8CP8) +
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
BIND PLAN(DSN8CH0) +
PKLIST(DSN8CP3) +
ACTION(REPLACE) RETAIN +
ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA!!) -
LIB('DSN!!0.RUNLIB.LOAD')
END
//*
//** STEP 24: CREATE CICS BMS PHYSICAL MAPS
//PH05PS24 EXEC DFHASMVS,COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=&&TEMP,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(1024,(100,10)),
// DCB=(RECFM=F,BLKSIZE=80)
//SYSIN DD DISP=SHR,DSN=DSN!!0.SDSNSAMP(DSN8CPG)
//*
//** STEP 25: LINKEDIT CICS BMS PHYSICAL MAPS
//PH05PS25 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSLMOD DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
// DD *
NAME DSN8CPG(R)
//*
//** STEP 26: CREATE CICS BMS PHYSICAL MAPS
//PH05PS26 EXEC DFHASMVS,COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=&&TEMP,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(1024,(100,10)),
// DCB=(RECFM=F,BLKSIZE=80)
//SYSIN DD DISP=SHR,DSN=DSN!!0.SDSNSAMP(DSN8CPD)
//*
//** STEP 27: LINKEDIT CICS BMS PHYSICAL MAPS
//PH05PS27 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSLMOD DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
// DD *
NAME DSN8CPD(R)
//*
//** STEP 28: CREATE CICS BMS PHYSICAL MAPS
//PH05PS28 EXEC DFHASMVS,COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=&&TEMP,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(1024,(100,10)),
// DCB=(RECFM=F,BLKSIZE=80)
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CPN),
// DISP=SHR
//*
//** STEP 29: LINKEDIT CICS BMS PHYSICAL MAPS
//PH05PS29 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSLMOD DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
// DD *
NAME DSN8CPN(R)
//*
//** STEP 30: CREATE CICS BMS PHYSICAL MAPS
//PH05PS30 EXEC DFHASMVS,COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=&&TEMP,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(1024,(100,10)),
// DCB=(RECFM=F,BLKSIZE=80)
//SYSIN DD DSN=DSN!!0.SDSNSAMP(DSN8CPL),
// DISP=SHR

```

```

//*
//** STEP 31: LINKEDIT CICS BMS PHYSICAL MAPS
//PH05PS31 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSLMOD DD DSN=DSN!!0.RUNLIB.LOAD,
// DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
// DD *
NAME DSN8CPL(R)
//*
//** STEP 32: CREATE CICS BMS PHYSICAL MAPS
//PH05PS32 EXEC DFHASMVS,COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=&&TEMP,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(1024,(100,10)),
// DCB=(RECFM=F,BLKSIZE=80)
//SYSIN DD DSN=DSN!!0.SDSENSAMP(DSN8CPU),
// DISP=SHR
//*
//** STEP 33: LINKEDIT CICS BMS PHYSICAL MAPS
//PH05PS33 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSLMOD DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
// DD *
NAME DSN8CPU(R)
//*
//** STEP 34: CREATE CICS BMS PHYSICAL MAPS
//PH05PS34 EXEC DFHASMVS,COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=&&TEMP,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(1024,(100,10)),
// DCB=(RECFM=F,BLKSIZE=80)
//SYSIN DD DSN=DSN!!0.SDSENSAMP(DSN8CPF),
// DISP=SHR
//*
//** STEP 35: LINKEDIT CICS BMS PHYSICAL MAPS
//PH05PS35 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSLMOD DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
// DD *
NAME DSN8CPF(R)
//*
//** STEP 36: CREATE CICS BMS PHYSICAL MAPS
//PH05PS36 EXEC DFHASMVS,COND=(4,LT),OUTC='*'
//SYSPUNCH DD DSN=&&TEMP,
// DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(1024,(100,10)),
// DCB=(RECFM=F,BLKSIZE=80)
//SYSIN DD DSN=DSN!!0.SDSENSAMP(DSN8CPE),
// DISP=SHR
//*
//** STEP 37: LINKEDIT CICS BMS PHYSICAL MAPS
//PH05PS37 EXEC PGM=IEWL,PARM='LIST,LET,XREF',COND=(4,LT)
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSLMOD DD DISP=SHR,DSN=DSN!!0.RUNLIB.LOAD
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSLIN DD DSN=&&TEMP,DISP=(OLD,DELETE)
// DD *
NAME DSN8CPE(R)

```

## Referencia relacionada

[“Ejemplos de aplicaciones en CICS” en la página 1461](#)

Un conjunto de aplicaciones de muestra de Db2 se ejecutan en el CICS entorno.



# Recursos de información para Db2 for z/OS y productos relacionados

---

Puede encontrar la documentación en línea de los productos de Db2 12 for z/OS y productos relacionados en Documentación de IBM.

Para toda la documentación de productos en línea de Db2 12 for z/OS, consulte [Documentación de IBM](https://www.ibm.com/docs/en/db2-for-zos/12) (<https://www.ibm.com/docs/en/db2-for-zos/12>).

Para otros manuales en PDF, consulte [Manuales en formato PDF para Db2 12 para z/OS](https://www.ibm.com/docs/en/SSEPEK_12.0.0/home/src/tpc/db2z_pdfmanuals.html) ([https://www.ibm.com/docs/en/SSEPEK\\_12.0.0/home/src/tpc/db2z\\_pdfmanuals.html](https://www.ibm.com/docs/en/SSEPEK_12.0.0/home/src/tpc/db2z_pdfmanuals.html)).



## Avisos

---

Esta información se ha desarrollado para productos y servicios que se ofrecen en EE.UU. Este material podría estar disponible a través de IBM en otros idiomas. Sin embargo, es posible que deba ser propietario de una copia del producto o de la versión del producto en dicho idioma para acceder a él.

Es posible que IBM no ofrezca los productos, servicios o funciones que se tratan en este documento en otros países. Consulte al representante local de IBM para obtener información sobre los productos y servicios que se pueden adquirir actualmente en su zona geográfica. Cualquier referencia a un producto, programa o servicio de IBM no pretende afirmar ni dar a entender que solo se pueda utilizar ese producto, programa o servicio de IBM. En su lugar podrá utilizarse cualquier producto, programa o servicio equivalente que no infrinja ninguno de los derechos de propiedad intelectual de IBM. Sin embargo, será responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier programa, producto o servicio que no sea de IBM.

IBM puede tener patentes o solicitudes de patentes pendientes que cubran el tema principal descrito en este documento. La posesión de este documento no confiere ninguna licencia sobre dichas patentes. Puede enviar consultas de licencia por escrito a:

*IBM Director of Licensing IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785 EE.UU.*

Para consultas de licencias relativas a información de doble byte (DBCS), póngase en contacto con el departamento de propiedad intelectual de IBM de su país o envíe las consultas, por escrito, a:

*Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, Japón*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL", SIN GARANTÍAS DE NINGUNA CLASE, NI EXPLÍCITAS NI IMPLÍCITAS, INCLUIDAS, PERO SIN LIMITARSE A, LAS GARANTÍAS IMPLÍCITAS DE NO INFRACCIÓN, COMERCIALIZACIÓN O IDONEIDAD PARA UNA FINALIDAD DETERMINADA. Algunas jurisdicciones no permiten la exclusión de garantías expresas o implícitas en determinadas transacciones, por lo que es posible que esta declaración no le concierna.

Esta publicación puede contener inexactitudes técnicas o errores tipográficos. Periódicamente se efectúan cambios en la información aquí contenida; dichos cambios se incorporarán a las nuevas ediciones de la publicación. IBM puede realizar mejoras y/o cambios en los productos y/o programas descritos en esta publicación en cualquier momento sin previo aviso.

Traducción automática (por máquina): La versión original del contenido técnico de IBM, incluida la documentación de los productos, es la versión en inglés de este contenido. Si tiene alguna pregunta o duda sobre el contenido traducido, consulte la versión en inglés. IBM declina cualquier responsabilidad por daños o pérdidas de cualquier tipo causados por el uso de contenido traducido automáticamente (por máquina). Para proporcionar comentarios sobre el contenido traducido, consulte la nota en la parte superior de la página equivalente en la versión HTML en línea traducida del contenido.

Cualquier referencia en este documento a sitios web que no son de IBM se proporciona únicamente para su comodidad y no significa en modo alguno que se recomiende dichos sitios web. La información de esos sitios web no forma parte de la información del presente producto de IBM y la utilización de esos sitios web se realiza bajo la responsabilidad del usuario.

IBM puede utilizar o distribuir cualquier información que se le proporcione en la forma que considere adecuada, sin incurrir por ello en ninguna obligación para con el remitente.

Los tenedores de licencias de este programa que deseen obtener información acerca de éste con el fin de permitir: (i) el intercambio de información entre programas creados independientemente y otros programas (incluido el presente) y (ii) la utilización mutua de la información que se ha intercambiado, deben ponerse en contacto con:

*IBM Director of Licensing IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785 EE. UU.*

Puede que esta información esté disponible, sujeta a los términos y condiciones adecuados, y puede incluir en algunos casos, el pago de una tasa.

IBM proporciona el programa bajo licencia descrito en este documento y todo el material bajo licencia disponible para el mismo según los términos del acuerdo de cliente de IBM, el acuerdo internacional de licencia de programa de IBM o cualquier otro acuerdo equivalente entre las partes.

Esta información contiene ejemplos de datos e informes utilizados en operaciones empresariales cotidianas. Para ilustrarlos de la forma más completa posible, los ejemplos incluyen nombres de personas, empresas, marcas y productos. Todos estos nombres son ficticios y cualquier similitud con personas y empresas reales es mera coincidencia.

#### LICENCIA DE COPYRIGHT:

Este manual contiene programas de aplicaciones de ejemplo escritos en lenguaje fuente, que muestran técnicas de programación en diversas plataformas operativas. Puede copiar, modificar y distribuir estos programas de ejemplo de cualquier forma sin pagar nada a IBM, bajo el propósito de desarrollo, uso, marketing o distribución de programas de aplicación de acuerdo con la interfaz de programación de la aplicación para la plataforma operativa para la cual se han escrito los programas de ejemplo. Estos ejemplos no se han probado exhaustivamente bajo todas las condiciones. Por consiguiente, IBM, no puede garantizar ni presuponer la fiabilidad, capacidad de servicio o función de dichos programas. Los programas de muestra se proporcionan "TAL Y COMO ESTÁN", sin garantía de ningún tipo. IBM no será responsable de los daños derivados del uso de los programas de ejemplo por parte del usuario.

Cada copia de cualquier parte de estos programas de ejemplo o de cualquier trabajo que derive de estos debe incluir un aviso de copyright como se indica a continuación:

© (nombre de su empresa) (año).

Las partes de este código se derivan de IBM Corp. Programas de ejemplo.

© Copyright IBM Corp. (*introduzca el año o los años*).

Si está viendo esta información en copia software, es posible que las fotografías y las ilustraciones en color no aparezcan.

## Información de la interfaz de programación

Esta información está destinada a ayudarle a escribir programas que contengan instrucciones SQL. Esta información documenta principalmente la interfaz de programación de uso general y la información de orientación asociada proporcionada por Db2 12 for z/OS. Sin embargo, esta información también documenta la interfaz de programación sensible al producto y la información de orientación asociada proporcionada por Db2 12 for z/OS.

### General-use Interfaz de programación e información de guía asociada

Las interfaces de programación de uso general permiten al cliente escribir programas que obtienen los servicios de Db2 12 for z/OS.

### Interfaz de programación sensible al producto e información de guía asociada

Las interfaces de programación sensibles al producto permiten que la instalación de cliente realice tareas como el diagnóstico, la modificación, la supervisión, la reparación, la personalización o el ajuste de este producto de software de IBM. La utilización de interfaces de este tipo crea dependencias en el diseño o implementación detallados del producto de software de IBM. Las interfaces de programación sensibles al producto se deben utilizar sólo para estos fines especializados. Debido a sus dependencias de diseño e implementación detallados, se espera que sea necesario cambiar los programas escritos en estas interfaces para que se ejecuten con nuevos releases o versiones del producto, o como resultado del servicio.

La interfaz de programación sensible al producto y la información de ayuda asociada se identifica donde se produce mediante los marcadores siguientes:

 Interfaz de programación para el producto e información de orientación asociada... 

## Marcas comerciales

---

IBM, el IBM logotipo y ibm.com son marcas comerciales o marcas registradas de International Business Machines Corp., registradas en muchas jurisdicciones en todo el mundo. Otros nombres de productos y servicios pueden ser marcas registradas de IBM o de otras empresas. Existe una lista actual de marcas registradas de IBM en la web en "Copyright and trademark information" en: <http://www.ibm.com/legal/copytrade.shtml>.

Linux es una marca comercial de Linus Torvalds en Estados Unidos y otros países, o ambos.

Microsoft, Windows, Windows NT y el logotipo de Windows son marcas registradas de Microsoft Corporation en Estados Unidos o en otros países.

UNIX es una marca registrada de The Open Group en Estados Unidos y en otros países.

Java y todas las marcas y logotipos basados en Java son marcas registradas de Oracle y/o de sus filiales.

## Términos y condiciones de la documentación de producto

---

El permiso para usar estas publicaciones se otorga conforme a los siguientes términos y condiciones:

**Aplicabilidad:** Estos términos y condiciones completan los términos y condiciones de uso del sitio web de IBM.

**Uso personal:** Puede reproducir estas publicaciones para su uso personal, no comercial, siempre y cuando se mantengan los avisos sobre la propiedad. No podrá distribuir, visualizar ni crear trabajo derivado de estas publicaciones, o cualquier parte de éstas, sin el consentimiento expreso de IBM.

**Uso comercial:** Puede reproducir, distribuir y visualizar estas publicaciones únicamente dentro de su empresa, siempre y cuando se mantengan todos los avisos sobre la propiedad. No podrá crear trabajo derivado de estas publicaciones, ni reproducir, distribuir ni visualizar estas publicaciones o cualquier parte de éstas sin el consentimiento expreso de IBM.

**Derechos:** Excepto lo expresamente concedido en este permiso, no se conceden otros permisos, licencias ni derechos, explícitos o implícitos, sobre las publicaciones ni sobre ninguna información, datos, software u otra propiedad intelectual allí contenida.

IBM se reserva el derecho de retirar los permisos otorgados aquí siempre que, a su discreción, considere que la utilización de las publicaciones actúa en detrimento de sus intereses o, según determine IBM, no se cumplan adecuadamente las instrucciones anteriores.

No puede descargar, exportar ni volver a exportar esta información excepto en el caso de cumplimiento total con todas las leyes y regulaciones vigentes, incluyendo todas las leyes y regulaciones sobre exportación de los Estados Unidos.

IBM NO PROPORCIONA NINGUNA GARANTÍA RELACIONADA CON EL CONTENIDO DE ESTAS PUBLICACIONES. LAS PUBLICACIONES SE PROPORCIONAN "TAL CUAL" Y SIN GARANTÍA DE NINGUNA CLASE, NI EXPLÍCITA NI IMPLÍCITA, INCLUYENDO PERO SIN LIMITARSE A LAS GARANTÍAS IMPLÍCITAS DE COMERCIALIZACIÓN, NO VULNERACIÓN E IDONEIDAD PARA UN FIN DETERMINADO.

## Consideraciones sobre la política de privacidad

---

Los productos de software de producto de IBM, incluido el software como soluciones de servicios, ("Software Offerings") pueden utilizar cookies u otras tecnologías para recopilar información del uso de productos, para ayudar a mejorar la experiencia del usuario final, para adaptar las interacciones con el usuario final o para otros fines. En muchos casos, las ofertas de software no recopilan información de identificación personal. Algunas de nuestras Ofertas de Software pueden ayudarle a recopilar información personal identifiable. Si esta Oferta de software usa cookies para recopilar información que lo identifique

de forma personal, la información específica sobre el uso de las cookies por parte de esta oferta se indica a continuación.

Esta oferta de software no utiliza cookies ni otras tecnologías para recopilar información identificable personalmente.

Si la configuración desplegada para esta Oferta de software le proporciona a usted como cliente la habilidad de recopilar información que lo identifique personalmente de los usuarios finales a través de las cookies y otras tecnologías, debe buscar su propia asesoría legal acerca de cualquier ley aplicable a dicha colección de datos, incluyendo cualquier requisito de aviso y consentimiento.

Para obtener más información sobre el uso de diversas tecnologías, incluidas las cookies, para estos fines, consulte IBM la Declaración de privacidad de <http://www.ibm.com/privacy>.

# Glosario

---

El glosario está disponible en la documentación de IBM

Para ver definiciones de términos de Db2 for z/OS, consulte [Glosario de Db2 \(Db2 Glossary\)](#).



# Índice

## Caracteres Especiales

\_ (subrayado)  
  variable de host ensamblador [582](#)  
' (apóstrofo)  
  opción de precompilador de delimitador de cadena [913](#)  
Área de comunicaciones de SQL (SQLCA)  
  descripción [557](#)  
  usando DSNTIAR para formatear [558](#)  
área de diagnóstico  
  RENUNCIA afectar en [262](#)  
  SEÑAL afectar en [262](#)  
índice  
  tipos  
    clave foránea [143](#)  
    exclusivo [150](#)  
    primario [150](#)  
    único en la clave principal [142](#)

## A

ABIERTO (función de conexión de CAF)  
  descripción [52](#)  
  ejemplo de programa [66](#)  
  ejemplos de idiomas [57](#)  
  sintaxis [57](#)  
  uso de sintaxis [57](#)  
accesibilidad  
  teclado [xiii](#)  
  teclas de acceso directo [xiii](#)  
acceso a datos  
  desde un programa de aplicación [373](#)  
Acceso a DRDA con declaraciones CONNECT  
  programa de ejemplo [668](#)  
acceso directo de fila [455](#)  
acceso DRDA  
  acceder a la mesa temporal remota [819](#)  
  codificación de una aplicación [817](#)  
  conectando con el servidor remoto [820](#)  
  consejos de programación [822](#)  
  liberar conexiones [821](#)  
  Limitaciones de SQL en diferentes servidores [822](#)  
  opciones de enlace [938](#)  
  opciones de precompilador [925](#)  
  preparación de programas [938](#)  
  programa de ejemplo [668](#)  
actualización  
  durante la recuperación [411](#)  
  grandes volúmenes [370](#)  
actualizar datos  
  mediante el uso de variables de host [518](#)  
adicción  
  datos [352](#)  
ADJUNTE la opción de precompilador [913](#)  
almacenamiento  
  adquisición  
    fila recuperada [532](#)

almacenamiento (*continuación*)  
  adquisición (*continuación*)  
    SQLDA [532](#)  
    direcciones en SQLDA [532](#)  
  almacenamiento de datos no tabulares [367](#)  
Anidación de sentencias SQL  
  funciones definidas por el usuario (UDF) [422](#)  
  procedimientos almacenados [422](#)  
  restricciones [422](#)  
Aplicación  
  volver a enlazar  
    Aplicación [947](#)  
aplicación de ejemplo  
  Acceso a DRDA con declaraciones CONNECT [668](#)  
  acceso DRDA [668](#)  
  DRDA con nombres de tres partes [674](#)  
  entornos [1094](#)  
  función definida por el usuario [1092](#)  
  idiomas [1094](#)  
  LOB [1092](#)  
  organización [1092](#)  
  procedimiento almacenado [1092](#)  
  programas [1074, 1091](#)  
  proyecto [1092](#)  
  SQL dinámico [606](#)  
  SQL estático [606](#)  
  teléfono [1092](#)  
  uso [1091](#)  
  aplicación para teléfono, descripción [1092](#)  
aplicaciones  
  diseño [1](#)  
  escribir para Db2 [487](#)  
  identificar incompatibilidades [888](#)  
  migración [1](#)  
  muestras suministradas con Db2 [1053](#)  
  planificación [1](#)  
  verificar cambios para detectar incompatibilidades [888](#)  
aplicaciones de ejemplo  
  almacenamiento [1070](#)  
  bases de datos [1071](#)  
  estructura [1070](#)  
  grupos de almacenamiento [1071](#)  
Aplicaciones de ejemplo  
  TSO [1095](#)  
aplicaciones de muestra suministradas [1053](#)  
Aplicaciones DSN, ejecutándose con CAF [40](#)  
aplicaciones incompatibles  
  identificar [888](#)  
aplicaciones SQL incorporadas  
  datos XML [571](#)  
  variables de host, datos XML [572](#)  
APLCOMPAT [878, 884](#)  
AS, cláusula  
  denominación de columnas de resultados [382](#)  
  denominación de columnas derivadas [382](#)  
  nombrar columnas en unión [382](#)  
  nombrar columnas para ver [382](#)

AS, cláusula (*continuación*)  
    ORDER BY NAME 380  
asignación de búfer fija  
    RECUPERAR CON CONTINUAR 446  
asignación dinámica de búfer  
    RECUPERAR CON CONTINUAR 445  
asignación, reglas de compatibilidad 129  
AUTH SIGNON (función de conexión de RRSAF)  
    ejemplos de idiomas 94  
    sintaxis 94  
autónomas  
    procedimientos  
        autónomas 240  
    procedimientos de SQL nativos  
        autónomas 240  
autorización  
    crear tablas de prueba 1017  
    ID de autorización 1012  
    Tabla SYSIBM.SYSTABAUTH 373  
Autorización de paquete  
    para procedimientos almacenados externos 290  
autovinculación 957

**B**

bibliotecas  
    para declaraciones de tablas y estructuras de variables de host 498

BIND  
    Db2 línea de comandos procesador comando 930

bloquear  
    escalamiento  
        al recuperar un gran número de filas 471

BTS (simulador de terminal de lotes) 1046

bucle infinito 410

**C**

C/C++  
    creación de un procedimiento almacenado 270

CAF (recurso de conexión de llamada)  
    descripción 42

CAMBIAR A (función de conexión de RRSAF)  
    ejemplos de idiomas 87  
    sintaxis 87

Campo AUTOCOMMIT del panel SPUFI 1023

campo de rastreo QW0366FN 884

campo de rastreo QW0376FN 878, 884

Campo SQLN de SQLDA 532

Campo SQLVAR de SQLDA 532

captación en bloque  
    con estabilidad del cursor 424  
    impedir 424

Carácter NUL en C 603

caracteres terminadores SQL no válidos 1081

CARGAR macro z/OS utilizada por CAF 47

CARGAR macro z/OS utilizada por RRSAF 79

CCSID (identificador de juego de caracteres codificados)  
    configuración de variables de host 508  
    control en programas COBOL 712  
    opción de precompilador 913  
    SQLDA 532

CEEDUMP

CEEDUMP (*continuación*)  
    utilizar para depurar procedimientos almacenados 1036

CERRAR (función de conexión de CAF)  
    descripción 52  
    ejemplo de programa 66  
    ejemplos de idiomas 59  
    sintaxis 59

CICS  
    gestión de almacenamiento  
        C 603  
        COBOL 654  
        ensamblador 582  
        PL/I 734  
    módulo de interfaz de lenguaje (DSNCLI)  
        uso en la edición de enlaces de una aplicación 926

operación  
    ejecutar un programa 1001  
    planificación medioambiental 1013  
    preparación con procedimientos JCL 964

programación  
    aplicaciones de ejemplo 1094, 1461  
    Macro DFHEIENT 582  
    Mandato SYNCPOINT 25

punto de sincronización 25

recursos  
    áreas de control 1001  
    conversor de lenguaje de mandatos 911  
    EDF (sistema de diagnóstico de ejecución) 1047

Subrutina DSNTIAC  
    C 603  
    COBOL 654  
    ensamblador 582  
    PL/I 734  
    unidad de trabajo 25

CICS, aplicaciones  
    reutilización de hebras 125

cifrado de datos 144

Cláusula CON RETENCIÓN  
    DECLARE CURSOR, sentencia 427  
    restricciones 427  
    y CICS 427  
    y IMS 427

Cláusula CONTINUE de la sentencia WHENEVER 562

Cláusula EXCEPT  
    columnas de la tabla de resultados 389

cláusula FOREIGN KEY  
    descripción 143  
    uso 143

Cláusula FULL OUTER JOIN 407

Cláusula INNER JOIN 401

Cláusula INTERSECT  
    columnas de la tabla de resultados 389

Cláusula LEFT OUTER JOIN 404

Cláusula NOT FOUND de la sentencia WHENEVER 562

Cláusula ON, unión de tablas 396

cláusula PRIMARY KEY  
    Sentencia ALTER TABLE 150  
    Sentencia CREATE TABLE 141

Cláusula RIGHT OUTER JOIN 405

Cláusula SET de la instrucción UPDATE 367

cláusula SQLERROR de la sentencia WHENEVER 562

cláusula SQLWARNING de sentencia WHENEVER 562

Cláusula ÚNICA 141

cláusula UNION

cláusula UNION (*continuación*)  
    columnas de la tabla de resultados 389  
    combinación de sentencias SELECT 389

Cláusula USING DESCRIPTOR  
    EXECUTE, sentencia 532  
    OPEN, sentencia 532  
    Sentencia FETCH 532

Cláusula VALUES, sentencia INSERT 352

Cláusula WHERE  
    Sentencia SELECT  
        descripción 376  
        unir tablas 396  
        unir una mesa consigo misma 401

cláusula WITH  
    expresiones de tabla comunes 156

clave  
    compuesto 143  
    exclusivo 150  
    foráneo 143  
    nivel superior 142  
    primario  
        definición 141  
        recomendaciones para definir 141  
        seleccionar 142  
        usando marca de tiempo 142

clave compuesta 143

Clave de clasificación  
    encargar 384  
    ORDER BY, cláusula 384

clave padre 142

cliente 37

CLOSE  
    sentencia  
        Cláusula SIEMPRE QUE NO SE ENCUENTRE 530, 532  
        descripción 433  
        recomendación 438

COALESCE, función 407

COBOL  
    creación de un procedimiento almacenado 270

codificación de instrucciones SQL  
    dinámica 524

código de razón  
    CAF 64  
    RRSAF 117  
    X"00D44057" 474

código de retorno  
    CAF 64  
    Comando DSN 1001  
    RRSAF 117

código reentrant  
    en procedimientos almacenados 295

columna  
    ancho de los resultados 1026, 1032  
    encabezado creado por SPUFI 1033  
    especificado en CREAR TABLA 127  
    etiquetas, uso 532  
    mostrando, lista de 373  
    nombre, con la instrucción UPDATE 367  
    recuperar, con SELECT 374  
    tipos de datos 129  
    valor predeterminado  
        definido por el sistema 127  
        definido por el usuario 127

columna de identidad  
    definición 133, 356  
    desencadenante 160  
    IDENTITY\_VAL\_LOCAL, función 133  
    insertar en la tabla 150  
    insertar valores en 356  
    usar como clave principal 133

Columna de resultados  
    nombrar con cláusula AS 382

Columna LOB, definición 131

Columna ROWID  
    definición 355  
    definición de LOB 131  
    insertar valores en 355  
    utilizar para acceso directo a la fila 455

Comando DSNH de TSO 1042

Comando SYNCPOINT de CICS 25

Comando TEST de TSO 1045

comparación  
    Cláusula WHERE  
        subconsulta 415  
        HAVING, cláusula  
            subconsulta 415  
        normas de compatibilidad 129  
        operador, subconsulta 415

compatibilidad  
    Reglas 129  
    tipos de datos 129

compatibilidad de aplicaciones  
    parámetro de subsistema 889  
    verificación 888

comprobación de seguridad multinivel (MLS)  
    desencadenantes 176  
    restricciones referenciales 140

CON RETENCIÓN cursor  
    efecto en SQL dinámico 549

condición de búsqueda  
    Cláusula WHERE 376  
    operadores de comparación 376  
    palabra clave NOT 376  
    Sentencia SELECT 412

condición de fin de datos 431, 434

condiciones  
    ignorar 258

conectar  
    Db2 39

CONNEXIÓN UBICACIÓN campo del panel SPUFI 1023

conexión a Db2  
    Requisitos de entorno 39

conexión CAF implícita 48

conexiones RRSAF implícitas 79

configuración del terminador SQL  
    DSNTIAD 1081  
    SPUFI 1030

conjunto de datos de entrada DDITV02 960

Conjunto de datos DSNTRACE 63

conjunto de filas  
    actualizar actual 435  
    eliminar actual 435

Conjuntos de datos SYSLIB 963

conjuntos de resultados  
    recibir de un procedimiento almacenado 811

conjuntos de resultados de procedimiento almacenado  
    recibir en un programa 811

CONNECT  
  sentencia  
    SPUFI 1023

CONNECT (función de conexión de CAF)  
  descripción 52  
  ejemplo de programa 66  
  ejemplos de idiomas 53  
  sintaxis 53

connection  
  Db2  
    conectando desde tareas 1004  
  función de CAF  
    CLOSE 59  
    CONNECT 53  
    DISCONNECT 61  
    OPEN 57  
    TRANSLATE 62

función de RRSAF  
  Cambiar a 87  
  CONTEXTO INICIAR SESIÓN 100  
  Crear hebra 109  
  FIND\_DB2\_SYSTEMS 116  
  IDENTIFY 83  
  INICIO DE SESIÓN AUTÉNTICO 94  
  SET\_CLIENT\_ID 105  
  SET\_ID 104  
  SET\_REPLICATION 108  
  SIGNON 89  
  Terminar hebra 112  
  TERMINAR IDENTIFICAR 113  
  TRANSLATE 114

constantes, sintaxis  
  C/C++ 617  
  Fortran 727

consultas  
  para las que la referencia de tabla no importa 1073  
  sintonización de programas de aplicación 473

consumidor de servicios web  
  SQLSTATE 863

CONTEXTO SIGNON (función de conexión de RRSAF)  
  ejemplos de idiomas 100  
  sintaxis 100

Contextos XPath  
  XMLEXISTS 472

Controlador CONTINUE (procedimiento SQL)  
  descripción 250  
  ejemplo 250

Controlador EXIT (procedimiento SQL) 250

CONVENCIÓN DE ENLACE GENERAL CON NULOS 274, 278

Convención de vinculación SQL 274, 282

CONVENCIÓN GENERAL DE ENLACES 274, 276

convenciones de enlaces  
  GENERAL 274, 276  
  GENERAL WITH NULLS 274, 278  
  procedimientos almacenados 274  
  SQL 274, 282

convenciones de registro  
  RRSAF 79

convenio de denominación  
  C 603  
  COBOL 654  
  ensamblador 582  
  Fortran 722  
  PL/I 734

convenio de denominación (*continuación*)  
  REXX 766  
  tablas que cree 144

coordinación de actualizaciones  
  datos distribuidos 37

coprocesador de Db2 897

creación de procedimientos almacenados  
  procedimientos de SQL externos 306

crear  
  procedimiento almacenado externo 270  
  procedimiento SQL externo mediante DSNTPSMP 309  
  procedimiento SQL externo mediante JCL 320

CREAR HILO (función de conexión de RRSAF)  
  ejemplo de programa 119  
  ejemplos de idiomas 109  
  sintaxis 109

crear objetos  
  en un programa de aplicación 127

CREATE PROCEDURE, sentencia  
  para procedimientos SQL externos 320  
  procedimiento almacenado externo 270

CREATE TRIGGER  
  descripción 160  
  ejemplo 160  
  indicación de fecha y hora 173  
  nombre de activación 160  
  orden de activación 173

CREATE TYPE, sentencia  
  ejemplo 181

CUALQUIER predicado cuantificado 415

CURRENT SERVER 37

CURRENT SERVER, registro especial  
  descripción 935  
  guardar valor en el programa de aplicación 822

CURRENT SQLID, registro especial  
  uso en prueba 1015  
  valor en la instrucción INSERT 127

cursor  
  atributos  
    usando OBTENER DIAGNÓSTICOS 448  
    usando SQLCA 447  
  cerrando  
    CLOSE, sentencia 438  
  descripción 423  
  desplazables  
    actualizable 424  
    descripción 424  
    dinámica 424  
    estática 424  
    ESTÁTICA SENSIBLE 424  
    INSENSITIVE 424  
    orientación de captación 439  
    recuperar filas 439  
    sensibilidad 424  
    SENSIBLE DINÁMICO 424  
  desplazables dinámicos 424  
  efecto de la noche sobre la posición 427  
  ejemplo  
    actualizar con cursor no desplazable 452  
    actualizar con cursor posicionado en fila 453  
    actualizar fila específica con cursor posicionado en fila 454  
    recuperar hacia atrás con cursor desplazable 452  
    eliminar una fila actual 435

cursor (*continuación*)  
estado abierto 427  
estático desplazable 424  
insensible desplazable 424  
mantener la posición 427  
no desplazables 424  
OPEN, sentencia 431  
rowset-positioned  
actualizar un conjunto de filas actual 435  
apertura 434  
condición de fin de datos 434  
declarar 434  
descripción 423  
número de filas 435  
número de filas en el conjunto de filas 438  
pasos para usar 433  
recuperar un conjunto de datos 435  
tabla de resultados 423  
tipos 424  
ubicación de fila  
actualizar una fila actual 431  
condición de fin de datos 431  
declarar 429  
descripción 423  
eliminar una fila actual 431  
pasos para usar 428  
recuperar una fila de datos 431  
WITH HOLD  
descripción 427  
cursor actualizable 429  
cursor de conjunto de filas  
apertura 434  
captación de varias filas 435  
cerrando 438  
condición de fin de datos 434  
Db2 for z/OS solicitante de nivel inferior 824  
declarar 434  
ejemplo 453  
utilización 433  
cursor desplazable  
actualizable 424  
comparación de tipos 440  
Db2 for z/OS solicitante de nivel inferior 824  
dinámica  
Modelo dinámico 424  
obtener la fila actual 444  
estática  
agujeros en la tabla de resultados 443  
creación de un agujero de actualización 443  
crear eliminar agujero 443  
eliminación de agujeros 442  
modelo estático 424  
número de filas 441  
estática sensible 424  
orientación de captación 439  
recuperar filas 439  
sensibilidad 440  
sensible dinámico 424  
cursos  
declarar en procedimientos SQL 249

## D

datos  
acceder desde un programa de aplicación 373  
actualización durante la recuperación 411  
actualizar datos recuperados anteriormente 450  
adición 352  
añadir al final de una tabla 367  
asociado con la cláusula WHERE 376  
desplazarse hacia atrás a través de 448  
distribuidas 37  
modificación 352  
moneda 424  
no en una tabla 473  
recuperación de grandes volúmenes 471  
recuperación mediante SELECT \* 412  
recuperar un conjunto de filas 431, 435  
seguridad e integridad 23  
Datos ASCII, recuperación 532  
datos de ejemplo 1053  
datos de entrada de caracteres  
Programa REXX 789  
datos del mensaje  
IBM MQ 825  
datos distribuidos  
confirmación en dos fases 37  
coordinación de actualizaciones 37  
copiar una mesa remota 817  
diseño de aplicaciones para 36  
ejecución de sentencias SQL largas 823  
ejemplo  
acceder a la mesa temporal remota 819  
conectando con el servidor remoto 820  
enlace en el servidor remoto 937  
usando la instrucción RELEASE 821  
usando nombres de tabla de tres partes 817  
uso de alias para varios sitios 821  
esquema de codificación de los datos recuperados 823  
identificar el servidor en tiempo de ejecución 822  
mantenimiento de la vigencia de los datos 424  
nombres de tabla de tres partes 817  
Opción de enlace DBPROTOCOL 817  
preparación de programas 941  
Proceso BIND 937  
programación  
codificación con acceso al protocolo privado de Db2 817  
codificación con acceso DRDA 817  
recuperación de tablas ASCII o Unicode 823  
transmisión de datos mixtos 822  
usar alias para la ubicación 821  
datos mixtos  
convertir 822  
descripción 129  
transmisión a una ubicación remota 822  
datos numéricos  
ancho de la columna en los resultados 1026  
descripción 129  
datos XML  
actualización, aplicaciones SQL integradas 577  
aplicaciones SQL incorporadas 571  
recuperación de tablas, aplicaciones SQL incrustadas 579  
seleccionar 381

Db2  
    conexión desde un programa 39  
Db2 acceso de protocolo privado  
    codificación de una aplicación 817  
Db2 aplicación de ejemplo  
    DSN8BC3 1098  
    DSN8BD3 1106  
    DSN8BE3 1113  
    DSN8BF3 1122  
    DSN8BP3 1116  
    DSN8CC0 1461  
    DSN8CC1 1469  
    DSN8CC2 1471  
    DSN8CP0 1476  
    DSN8CP1 1482  
    DSN8CP2 1484  
    DSN8CP3 1503  
    DSN8CP6 1489  
    DSN8CP7 1495  
    DSN8CP8 1498  
    DSN8DLPL 1321  
    DSN8DLPV 1347  
    DSN8DLRV 1334  
    DSN8DUAD 1248  
    DSN8DUAT 1256  
    DSN8DUCD 1261  
    DSN8DUCT 1278  
    DSN8DUCY 1290  
    DSN8DUTI 1296  
    DSN8DUWC 1303  
    DSN8DUWF 1306  
    DSN8EC1 1207  
    DSN8EC2 1215  
    DSN8ED1 1190  
    DSN8ED2 1200  
    DSN8ED3 1222  
    DSN8ED4 325  
    DSN8ED5 341  
    DSN8ED6 1228  
    DSN8ED7 1231  
    DSN8ED9 1235  
    DSN8EP1 1169  
    DSN8EP2 1182  
    DSN8EPU 1187  
    DSN8ES1 1220  
    DSN8ES2 1226  
    DSN8ES3 1241  
    DSN8EUDN 1314  
    DSN8EUMN 1318  
    DSN8HC3 1129  
    DSN8IC0 1415  
    DSN8IC1 1419  
    DSN8IC2 1422  
    DSN8IP0 1427  
    DSN8IP1 1430  
    DSN8IP2 1432  
    DSN8IP3 1448  
    DSN8IP6 1437  
    DSN8IP7 1440  
    DSN8IP8 1442  
    DSN8SC3 1156  
    DSN8SP3 1163  
    DSN8WLMP 339  
    DSNTEJ1L 1372

Db2 aplicación de ejemplo (*continuación*)  
    DSNTEJ1P 1370  
    DSNTEJ2A 1367  
    DSNTEJ2C 1353  
    DSNTEJ2D 1356  
    DSNTEJ2E 1357  
    DSNTEJ2F 1362  
    DSNTEJ2P 1359  
    DSNTEJ2U 1399  
    DSNTEJ3C 1363  
    DSNTEJ3P 1366  
    DSNTEJ4C 1455  
    DSNTEJ4P 1457  
    DSNTEJ5C 1510  
    DSNTEJ5P 1380, 1512  
    DSNTEJ6 1365  
    DSNTEJ61 1380, 1512  
    DSNTEJ62 1382  
    DSNTEJ63 1384  
    DSNTEJ64 1385  
    DSNTEJ65 1387  
    DSNTEJ66 1394  
    DSNTEJ67 345  
    DSNTEJ6D 1376  
    DSNTEJ6P 1374, 1410  
    DSNTEJ6S 1375  
    DSNTEJ6T 1378, 1413  
    DSNTEJ6W 1390  
    DSNTEJ6Z 1393  
    DSNTEJ71 1374, 1410  
    DSNTEJ73 1412  
    DSNTEJ75 1378, 1413  
    DSNTIJLC 872  
    DSNTIJLR 875

Db2 coprocessor  
    DSNXDBRM 901  
        módulo de solicitud de base de datos (DBRM) 901  
        Procesamiento de sentencias SQL 896  
        salida 901

Db2 precompiler  
    descripción 896  
    programas de precompilación 896

Db2 terminación anómala  
    Lote DL/I 474

DB2\_RETURN\_STATUS  
    usando para obtener el estado del procedimiento 810

DB2I  
    invocando DCLGEN 491  
    paneles predeterminados 894

DB2I (DB2 Interactive)  
    ejemplo de preparación del programa 967  
    interrupción 1019  
    menú 1019  
    paneles  
        BIND PACKAGE 976  
        BIND PLAN 979  
        Compilar, vincular y ejecutar 989  
        DB2IMenú de opciones principal 1019  
        Precompilar 974  
        preparación de programas 967  
        Tipos de conexión del sistema 986  
        Valores predeterminados actuales de SPUFI 1025  
        Valores predeterminados para BIND PLAN 984

**DB2I ( DB2 Interactive) (continuación)**  
 paneles (*continuación*)
 

- Valores predeterminados para el PAQUETE REBIND 982
- Valores predeterminados para el PLAN REBIND 984
- Valores predeterminados para ENVÍO GRATUITO 982
- preparación de programas 891
- Procesamiento EDITJCL
  - ejecutar bibliotecas de tiempo 971
- Proceso en segundo plano
  - ejecutar bibliotecas de tiempo 971
- seleccionar
  - SPUFI 1019
  - SPUFI 1019

- DB2I valores por omisión
- valor 894
- DBCS (juego de caracteres de doble byte)
- traducción en CICS 911
- DBINFO
- función definida por el usuario 205
- pasar a un procedimiento almacenado externo 270
- DBRM (módulo de solicitud de base de datos)
- Db2 coprocessor salida 901
- descripción 901, 908
- DBRM en archivos HFS
- vincular 930
- DCLGEN
- declaraciones de variables 496
- declarar matrices de variables de indicadores 491
- Ejemplo de COBOL 498
- generación de declaraciones de tabla y vista 490
- generar declaraciones de tabla y vista de DB2I 491
- INCLUDE, sentencia 498
- incluidas las declaraciones en un programa 498
- invocar 490
- tipos de datos 496
- usando desde DB2I 491
- DCLGEN (generador de declaraciones)
- descripción 490
- DDITV02 conjunto de datos de entrada 960
- DDOTV02 conjunto de datos de salida 960
- DEC15
- opción de precompilador 913
- Reglas 411
- DEC31
- evitar el desbordamiento 412
- opción de precompilador 913
- Reglas 411
- decimal
- aritméticos 411
- precisión de 15 dígitos 411
- precisión de 31 dígitos 411
- DECIMAL, tipo de datos
- C/C++ 617
- Declaración CONNECT, con acceso DRDA 820
- Declaración de EXENCIÓN, con acceso DRDA 821
- declaración FETCH de varias filas
- CÓDIGO SQL +100 557
- comprobación DB2\_LAST\_ROW 565
- declaración INSERT de varias filas
- ejecución dinámica 552
- NO ATÓMICO CONTINUAR EN SQLEXCEPTION 563
- usando GET DIAGNOSTICS 563
- declaraciones de origen modificadas 908
- declaraciones de tabla
- añadir a bibliotecas 498
- Declaraciones de tablas y vistas generadas con DCLGEN 490
- declarar tablas y vistas
- ventajas 489
- DECLARE CURSOR, sentencia
- Cláusula CON RETENCIÓN 427
- CON cláusula ROWSET POSITIONING 434
- CON opción DE DEVOLUCIÓN 293
- cursor desplazable 424
- declaración preparada 530, 532
- descripción, posicionado en fila 429, 434
- FOR UPDATE, cláusula 429
- seguridad de varios niveles 429
- DECLARE GLOBAL TEMPORARY TABLE, sentencia 148
- depuración
- grabación de mensajes para procedimientos almacenados 1039
- procedimientos almacenados 1036
- depuración de programas de aplicación 1040
- DESCONECTAR (función de conexión de CAF)
- descripción 52
- ejemplo de programa 66
- ejemplos de idiomas 61
- sintaxis 61
- DESCRIBE INPUT, sentencia 555
- DESCRIBE, sentencia
- Cláusulas INTO 532
- etiquetas de columna 532
- desencadenante
- actualización 160
- codificar 160
- con seguridad a nivel de fila 176
- denominación 160
- descripción 160
- ejemplo de piezas 160
- FOR EACH ROW 160
- FOR EACH STATEMENT 160
- granularidad 160
- hora de activación 160
- insertar 160
- integridad de los datos 176
- interacción con las columnas de etiquetas de seguridad 176
- interacción con restricciones 174
- invocación de función definida por el usuario 168
- invocar procedimiento almacenado 168
- orden de activación 173
- partes de 160
- pasar tablas de transición 168
- suceso desencadenante 160
- suprimir 160
- tabla de transición 160
- tabla sujeto 160
- transmitir en cascada 173
- utilizando columnas de identidad 160
- variable de transición 160
- desplazamiento
- en cualquier dirección 440
- hacia atrás a través de los datos 448
- hacia atrás utilizando columnas de identidad 448
- hacia atrás utilizando ROWIDs 448
- ISPF (Interactive System Productivity Facility) 1033

determinación del problema, directrices	<a href="#">1041</a>
DFSLI000 (módulo de interfaz de lenguaje IMS)	<a href="#">926</a>
diagrama de sintaxis	
cómo leer	<a href="#">xiv</a>
direcccionamiento de 31 bits	<a href="#">989</a>
discapacidad	<a href="#">xiii</a>
diseño	
aplicaciones	<a href="#">1</a>
Diseño de aplicaciones	
datos distribuidos	<a href="#">36</a>
DISTINCT	
cláusula de la sentencia SELECT	<a href="#">382</a>
valores exclusivos	<a href="#">382</a>
dos puntos	
precediendo a una matriz de variables de host	<a href="#">520</a>
precediendo a una variable de host	<a href="#">514</a>
DRDA con nombres de tres partes	
programa de ejemplo	<a href="#">674</a>
DSN8BC3	<a href="#">1098</a>
DSN8BC3 programa de muestra	<a href="#">654</a>
DSN8BD3	<a href="#">1106</a>
DSN8BD3 programa de muestra	<a href="#">603</a>
DSN8BE3	<a href="#">1113</a>
DSN8BE3 programa de muestra	<a href="#">603</a>
DSN8BF3	<a href="#">1122</a>
DSN8BF3 programa de muestra	<a href="#">722</a>
DSN8BP3	<a href="#">1116</a>
DSN8BP3 programa de muestra	<a href="#">734</a>
DSN8CC0	<a href="#">1461</a>
DSN8CC1	<a href="#">1469</a>
DSN8CC2	<a href="#">1471</a>
DSN8CP0	<a href="#">1476</a>
DSN8CP1	<a href="#">1482</a>
DSN8CP2	<a href="#">1484</a>
DSN8CP3	<a href="#">1503</a>
DSN8CP6	<a href="#">1489</a>
DSN8CP7	<a href="#">1495</a>
DSN8CP8	<a href="#">1498</a>
DSN8DLPL	<a href="#">1321</a>
DSN8DLPV	<a href="#">1347</a>
DSN8DLRV	<a href="#">1334</a>
DSN8DUAD	<a href="#">1248</a>
DSN8DUAT	<a href="#">1256</a>
DSN8DUCD	<a href="#">1261</a>
DSN8DUCT	<a href="#">1278</a>
DSN8DUCY	<a href="#">1290</a>
DSN8DUTI	<a href="#">1296</a>
DSN8DUWC	<a href="#">1303</a>
DSN8DUWF	<a href="#">1306</a>
DSN8EC1	<a href="#">1207</a>
DSN8EC2	<a href="#">1215</a>
DSN8ED1	<a href="#">1190</a>
DSN8ED2	<a href="#">1200</a>
DSN8ED3	<a href="#">1222</a>
DSN8ED4	<a href="#">325</a>
DSN8ED5	<a href="#">341</a>
DSN8ED6	<a href="#">1228</a>
DSN8ED7	<a href="#">1231</a>
DSN8ED9	<a href="#">1235</a>
DSN8EP1	<a href="#">1169</a>
DSN8EP2	<a href="#">1182</a>
DSN8EPU	<a href="#">1187</a>
DSN8ES1	<a href="#">1220</a>
DSN8ES2	<a href="#">1226</a>
DSN8ES3	<a href="#">1241</a>
DSN8EUDN	<a href="#">1314</a>
DSN8EUMN	<a href="#">1318</a>
DSN8HC3	<a href="#">1129</a>
DSN8ICO	<a href="#">1415</a>
DSN8IC1	<a href="#">1419</a>
DSN8IC2	<a href="#">1422</a>
DSN8IP0	<a href="#">1427</a>
DSN8IP1	<a href="#">1430</a>
DSN8IP2	<a href="#">1432</a>
DSN8IP3	<a href="#">1448</a>
DSN8IP6	<a href="#">1437</a>
DSN8IP7	<a href="#">1440</a>
DSN8IP8	<a href="#">1442</a>
DSN8SC3	<a href="#">1156</a>
DSN8SP3	<a href="#">1163</a>
DSN8WLMP	<a href="#">339</a>
DSNALI	
cargando	<a href="#">45</a>
disponibles	<a href="#">45</a>
DSNALI (módulo de interfaz de lenguaje CAF)	
ejemplo de carga	<a href="#">66</a>
ejemplo de eliminación	<a href="#">66</a>
DSNCLI (módulo de interfaz de lenguaje CICS)	<a href="#">926</a>
DSNEBP10	<a href="#">982</a>
DSNEBP11	<a href="#">982</a>
DSNHCOB2	procedimiento <a href="#">962</a>
DSNHCPP2	procedimiento <a href="#">962</a>
DSNHICB2	procedimiento <a href="#">962</a>
DSNHLI2	punto de entrada a DSNALI
ejemplo de programa	<a href="#">66</a>
DSNMTV01	módulo <a href="#">1007</a>
DSNRLI	
cargando	<a href="#">77</a>
disponibles	<a href="#">77</a>
DSNTEDIT CLIST	<a href="#">953</a>
DSNTEJ1L	<a href="#">1372</a>
DSNTEJ1P	<a href="#">1370</a>
DSNTEJ2A	<a href="#">1367</a>
DSNTEJ2C	<a href="#">1353</a>
DSNTEJ2D	<a href="#">1356</a>
DSNTEJ2E	<a href="#">1357</a>
DSNTEJ2F	<a href="#">1362</a>
DSNTEJ2P	<a href="#">1359</a>
DSNTEJ2U	<a href="#">1399</a>
DSNTEJ3C	<a href="#">1363</a>
DSNTEJ3P	<a href="#">1366</a>
DSNTEJ4C	<a href="#">1455</a>
DSNTEJ4P	<a href="#">1457</a>
DSNTEJ5C	<a href="#">1510</a>
DSNTEJ5P	<a href="#">1380</a> , <a href="#">1512</a>
DSNTEJ6	<a href="#">1365</a>
DSNTEJ61	<a href="#">1380</a> , <a href="#">1512</a>
DSNTEJ62	<a href="#">1382</a>
DSNTEJ63	<a href="#">1384</a>
DSNTEJ64	<a href="#">1385</a>
DSNTEJ65	<a href="#">1387</a>
DSNTEJ66	<a href="#">1394</a>
DSNTEJ67	<a href="#">345</a>
DSNTEJ6D	<a href="#">1376</a>
DSNTEJ6P	<a href="#">1374</a> , <a href="#">1410</a>
DSNTEJ6S	<a href="#">1375</a>
DSNTEJ6T	<a href="#">1378</a> , <a href="#">1413</a>
DSNTEJ6W	<a href="#">1390</a>

DSNTEJ6Z [1393](#)  
 DSNTEJ71 [1374](#), [1410](#)  
 DSNTEJ73 [1412](#)  
 DSNTEJ75 [1378](#), [1413](#)  
 DSNTEP2 programa de muestra  
     cómo ejecutar [1074](#)  
     parameters [1074](#)  
     preparación de programas [1074](#)  
 DSNTEP2 y programa de muestra de DSNTEP4  
     especificar terminador SQL [1076](#), [1084](#)  
 DSNTEP4 programa de muestra  
     cómo ejecutar [1074](#)  
     parameters [1074](#)  
     preparación de programas [1074](#)  
 DSNTIJLC [872](#)  
 DSNTIJLR [875](#)  
 DSNTPSMP  
     Autorizaciones necesarias [309](#)  
     creación de procedimientos SQL externos [309](#)  
     sintaxis para invocar [313](#)  
 DSNULI [123](#)  
 DSNXDBRM  
     Db2 coprocessor salida [901](#)  
 DSNXNBRM [908](#)

**E**

ECB (bloque de control de suceso, event control block)  
     Función CONNECT de CAF [53](#)  
     Función IDENTIFICAR de RRSAF [83](#)  
     Función SET\_REPLICATION de RRSAF [108](#)  
 edición de enlaces  
     Opción AMODE [989](#)  
     Opción RMODE [989](#)  
 editar enlace [123](#)  
 efecto desagüe pluvial [124](#)  
 ejecución del programa de aplicación  
     CICS [1013](#)  
     errores [1041](#)  
     IMS [1013](#)  
 ejecutar bibliotecas de tiempo, DB2I  
     Procesamiento EDITJCL [971](#)  
     Proceso en segundo plano [971](#)  
 ejemplos  
     MQListener [857](#)  
 El diseño de programas de aplicación  
     planificación de cambios [16](#)  
 ENLACAR COPIAR REEMPLAZAR  
     para procedimientos SQL nativos [263](#)  
 ENLACE A LA COPIA  
     para procedimientos SQL nativos [262](#)  
 enlaces  
     sitios web no de IBM [1525](#)  
 entorno de producción  
     implementación de procedimientos SQL nativos [267](#)  
 entorno de prueba, diseño [1001](#)  
 error  
     códigos de retorno [557](#)  
     desbordamiento [570](#)  
     división por cero [570](#)  
     ejecución [1041](#)  
     expresión aritmética [570](#)  
     gestión [562](#)  
     mensajes generados por el precompilador [1042](#)  
     errores al recuperar datos en una variable de host  
         causa determinante [509](#)  
     escasez de almacenamiento  
         al llamar a procedimientos almacenados [807](#)  
     espacio de tablas  
         no registrado cronológicamente  
         recuperar [35](#)  
     espacios de tablas  
         para aplicaciones de ejemplo [1071](#)  
 Especificación de RANK  
     ejemplo [386](#)  
 Especificación DENSE\_RANK  
     ejemplo [386](#)  
 estado del procedimiento  
     recuperación [810](#)  
     valor [810](#)  
 estructura de host  
     C/C++ [636](#)  
     COBOL [705](#)  
     descripción [506](#)  
     estructura del indicador [506](#)  
     PL/I [757](#)  
     recuperación de una fila de datos [523](#)  
     usando SELECT INTO [523](#)  
 estructura del indicador  
     descripción [506](#)  
 estructura matriz host variable  
     declarar [642](#)  
     referencias en sentencias SQL [641](#)  
 etiqueta, columna [532](#)  
 EXCEPT  
     mantener filas duplicadas con TODOS [389](#)  
 EXECUTE IMMEDIATE, sentencia [548](#)  
 EXECUTE, sentencia  
     Cláusula USING DESCRIPTOR [532](#)  
     ejecución dinámica [549](#)  
     tipos de parámetro [532](#)  
 EXISTS predicado, subconsulta [415](#)  
 EXPLAIN  
     revinculación automática [957](#)  
 expresión de tabla anidada  
     nombre de correlación [396](#)  
     operación de unión [396](#)  
     referencia correlacionada [396](#)  
 expresiones aritméticas en la instrucción UPDATE [367](#)  
 expresiones de tabla comunes  
     bucles infinitos [410](#)  
     descripción [156](#)  
     ejemplos [156](#)  
     en una declaración CREATE VIEW [154](#)  
     en una instrucción SELECT [154](#)  
     en una sentencia INSERT [154](#)  
     recurrencia [156](#)  
 expresiones XPath [381](#)

**F**

fila  
     actualización [367](#)  
     actualización de grandes volúmenes [370](#)  
     actualizar actual [431](#)  
     seleccionar con la cláusula WHERE [374](#)  
 FIND\_DB2\_SYSTEMS (función de conexión de RRSAF)  
     ejemplos de idiomas [116](#)

FIND\_DB2\_SYSTEMS (función de conexión de RRSAF) (*continuación*)  
    sintaxis 116  
FOR UPDATE, cláusula 429  
formatear  
    resultados, tabla 381  
formato  
    Resultados de la instrucción SELECT 1032  
    SQL en conjunto de datos de entrada 1019  
Fortran programa de aplicación  
    compatibilidad de tipos de datos 731  
    convenio de denominación 722  
    declaración @PROCESO 722  
    declaración de variable 727  
    declaración de variable indicadora 729  
    declarar mesas 722  
    declarar opiniones 722  
    definición de la SQLDA 502, 725  
    etiquetas de extracto 722  
    INCLUDE, sentencia 722  
    incluido SQLCA 724  
    opción de precompilador predeterminada 924  
    opción paralela 722  
    sintaxis constante 727  
    tipo de datos byte 722  
    Variable de host SQLCODE 724  
    Variable de host SQLSTATE 724  
    variable de host, declarando 726  
    WHENEVER, sentencia 722  
FROM, cláusula  
    Sentencia SELECT 374  
    unir tablas 396  
FRR (rutina de recuperación funcional)  
    en CAF 45  
función definida por el usuario  
    z/OS Debugger 1033  
función definida por el usuario (UDF)  
    acceder a tablas de transición 219  
    ALTER FUNCTION, sentencia 189  
    anidar sentencias SQL 422  
    argumentos de reparto 484  
    consideraciones de paralelismo 197  
    convenciones de parámetros  
        C 209  
        COBOL 212  
        ensamblador 208  
        PL/I 213  
    CREATE FUNCTION, sentencia 189  
    cursor desplazable 476  
Definidor 196  
denominación 203  
descripción 196  
directrices de codificación 197  
DSN\_FUNCTION\_TABLE 482  
ejemplo  
    derivadas 189  
    escalares externas 189, 224  
    resolución de función 479  
    sobrecarga del operador 189  
    SQL 189  
    tabla externa 189  
establecer valores de resultados 202  
Estructura DBINFO 205  
heredar registros especiales 215  
ID de autorización 477  
    función definida por el usuario (UDF) (*continuación*)  
        implementación 192  
        Implementador 196  
        indicadores  
            entrada 202  
            resultado 202  
        invocación desde el predicado 476  
        invocación desde un activador 168  
        invocador 196  
        invocar 476  
        localizadores de mesas  
            C 221  
            COBOL 221  
            ensamblador 220  
            PL/I 222  
        Mensaje de diagnóstico 203  
        muestras 225  
        múltiples programas 214  
        Nombre específico 203  
        pasos para crear y utilizar 196  
        preparar 222  
        probar 1033  
        programa principal 197  
        Promoción de tipo de datos 479  
        reentrant 214  
        resolución de función 479  
        restricciones 197  
        scratchpad 203, 223  
        simplificar la resolución de funciones 478  
        Subprograma 197  
        terminación anormal 223  
        Tipo de llamada 203  
        tipos 196  
        tipos de datos de host  
            C 200  
            COBOL 200  
            ensamblador 200  
            PL/I 200  
    Función ENCRYPT\_TDES 144  
    Función incorporada POWER 8  
    función RID 455  
    Funciones de CAF  
        resumen de comportamiento 51  
    Funciones de Db2  
        IBM MQ  
            MQREADALL 827  
            MQREADALLCLOB 827  
            MQREADCLOB 827  
            MQRECEIVE 827  
            MQRECEIVEALL 827  
            MQRECEIVEALLCLOB 827  
            MQRECEIVECLOB 827  
            MQSEND 827  
        funciones de tabla de SQL 196  
        funciones definidas por el usuario (UDF)  
            SOAPHTTPNC 863  
            SOAPHTTPNV 863  
    funciones RRSAF  
        resumen de comportamiento 81  
fusión  
    datos 358

## G

generación de declaraciones de tabla y vista  
    con DCLGEN de DB2I 491  
    usando DCLGEN 490  
generación de documentos XML para la cola de mensajes de MQ 829  
generador de declaraciones (DCLGEN)  
    descripción 490  
gestión de condiciones de excepción 562  
GET DIAGNOSTICS  
    usando para obtener el estado del procedimiento 810  
GROUP BY, cláusula  
    uso con funciones agregadas 393  
grupos de almacenamiento  
    para aplicaciones de ejemplo 1071

## H

HAVING, cláusula  
    seleccionar grupos sujetos a condiciones 394  
hora en que se modificó esa fila  
    determinación 471  
HOST  
    opción de precompilador 913  
    Valor FOLD para C y CPP 913

## I

IBM Data Studio Developer  
    creación de procedimientos SQL externos 306  
    Creación de procedimientos SQL nativos 244  
IBM MQ  
    descripción 824  
    Funciones de Db2  
        conexión de aplicaciones 845  
        consideraciones sobre programación 827  
        envío de mensajes 843  
        recuperación de mensajes 844  
    funciones de tabla  
        Función escalar MQREADALL 827  
        MQREADALLCLOB 827  
        MQRECEIVEALL 827  
        MQRECEIVEALLCLOB 827  
    funciones escalares  
        MQREADCLOB 827  
        MQRECEIVE 827  
        MQRECEIVECLOB 827  
        MQSEND 827  
interacción con Db2 824  
Interfaz de colas de mensajes (MQI) 825  
Las API 824  
manejo de mensajes 825  
mensajes 824  
IDENTIFICAR (función de conexión de RRSAF)  
    ejemplo de programa 119  
    ejemplos de idiomas 83  
    syntax 83  
IFCID 0366 884  
IFCID 0376 878, 884  
IKJEFT01 programa de monitorización de terminales en TSO 1012  
IMS

IMS (*continuación*)  
    edición de enlaces 926  
    Llamada CHKP 29  
    Llamada de ASIENTOS 26, 29  
    Llamada ROLB 26, 29  
    Llamada SYNC 29  
    llamadas en los puntos de control 29  
    módulo de interfaz de lenguaje (DFSLI000) 926  
    planificación medioambiental 1013  
    punto de confirmación 29  
    puntos de comprobación 31  
    recuperación 26  
    unidad de trabajo 29  
IN predicado, subconsulta 415  
Incompatibilidades de release de SQL 1  
incompatibilidades de releases  
    aplicaciones y SQL 1  
incompatibilidades del release de aplicación 1  
indicador variable arraysC/C++ syntax 639  
indicador variable arraysCOBOL syntax 711  
indicador variable arraysPL/I syntax 759  
indicador variables Syntax C/C++ 639  
indicador variablesPL/I syntax 759  
información de interfaz de programación, descripción 1524  
información de programación de uso general, descripción 1524  
información de programación sensible al producto, descripción 1524  
información del marcador de parámetros  
    obtención mediante el uso de un SQLDA 555  
insertar  
    valores de matrices de variables de host 521  
insertar datos  
    mediante el uso de variables de host 519  
Instrucción REPLACE (COBOL) 654  
Instrucción SELECT FROM DELETE  
    con cláusula INCLUDE 372  
    descripción 372  
    recuperación  
        Varias filas 372  
Instrucción SELECT FROM UPDATE  
    con la cláusula INCLUDE 361, 369  
    descripción 369  
    recuperación  
        Varias filas 369  
integridad de los datos  
    tablas 136  
integridad referencial  
    consideraciones sobre programación 1050  
    efecto en subconsultas 419  
Interactive System Productivity Facility (ISPF) 1019  
Interfaz de colas de mensajes (MQI)  
    políticas 826  
    services 826  
    Tablas MQ de Db2 830  
    WebSphere MQ 825  
Interfaz de lenguaje universal 121  
INTERSECT  
    mantener filas duplicadas con TODOS 389  
introducción progresiva  
    rebind 950  
Invocación de procedimientos almacenados  
    syntax para el procesador de línea de comandos Db2 1011  
invocar

## invocar (*continuación*)

- recurso de conexión de llamada (CAF) [40](#)
- Recurso de conexión de Resource Recovery Services (RRSAF) [71](#)
- IR A cláusula de SIEMPRE declaración [562](#)
- ISPF (Interactive System Productivity Facility)
  - comando de desplazamiento [1033](#)
  - DB2 utiliza gestión de diálogos [1019](#)
  - examinar [1023, 1032](#)
  - Panel de preparación del programa [967](#)
  - programación [1004](#)

## J

### JCL (lenguaje de control de trabajos)

- DDNAME Formato de lista [907](#)
- ejemplo de eliminación de lotes [1009](#)
- Formato de lista de opciones del precompilador [907](#)
- formato de número de página [908](#)
- iniciar una aplicación por lotes TSO [1012](#)
- preparar un programa de e CICS [964](#)
- preparar un programa orientado a objetos [901](#)
- procedimientos de precompilación [962](#)

## L

### lenguaje principal

- SQL dinámico [528](#)
- lista de comprobación de problemas del programa
  - documentar situaciones de error [1040](#)
  - mensajes de error [1045, 1046](#)
- lista de materiales aplicaciones [156](#)

### Lista de parámetros

- procedimientos almacenados [230](#)

### Llamada CHKP, IMS

- 29

### Llamada de lista, IMS

- 26, 29

### Llamada de sincronización cancelada

- [474](#)

### Llamada ROLB, IMS

- 26, 29

### Llamada SYNC, IMS

- 29

### LLAMAR A DSNALI

- Lista de parámetros [49](#)

- parámetros necesarios [49](#)

### LLAMAR A DSNRLI

- Lista de parámetros [80](#)

- parámetros necesarios [80](#)

### llamar adjunto idioma interfaz

- cargando [45](#)

- disponibles [45](#)

### LOB matriz de variables de host

- C/C++ [628](#)

- COBOL [696](#)

- PL/I [751](#)

### Localizador de conjunto de resultados

- C/C++ [617](#)

- COBOL [686](#)

- ensamblador [588](#)

- Fortran [727](#)

- PL/I [745](#)

### localizador de tablas

- C/C++ [617](#)

- COBOL [686](#)

- ensamblador [588](#)

- PL/I [745](#)

## localizador LOB

- C/C++ [617, 628](#)
- COBOL [696](#)
- ensamblador [588](#)
- Fortran [727](#)
- PL/I [745, 751](#)

### Lote DL/I

- características [474](#)
- DDITV02 conjunto de datos de entrada [960](#)
- DSNMTV01 módulo [1007](#)
- iD del punto de control [1010](#)
- Parámetro SSM= [1007](#)
- presentar una solicitud [1007](#)
- programación de aplicaciones [474](#)
- Requisitos de Db2 [474](#)

## M

### macro de correlación

- aplicaciones de ensamblador [601](#)
- DSNXDBRM [908](#)
- DSNXNBRM [908](#)

### Macro DFHEIENT

#### mandato de DSN de TSO

- procesamiento del código de devolución [1001](#)
- Subcomandos RUN [1001](#)

#### mandatos

- MQListener [853](#)

#### manejador, utilizando en el procedimiento SQL

#### manejadores de condiciones

- vacía [258](#)

#### manejo de mensajes

- IBM MQ [825](#)

#### materialización

- LOB [464](#)

#### matrices

##### ejemplo

- utilizar matrices en un procedimiento SQL nativo [187](#)

##### ejemplo de procedimiento SQL nativo

#### matrices de variables indicadoras

- declarar con DCLGEN [491](#)

#### matriz binaria de variables de host

- C/C++ [628](#)

- PL/I [751](#)

#### matriz de variables de host

- C/C++ [628](#)

- COBOL [686, 696](#)

- descripción [504, 520](#)

- Inserción de varias filas [521](#)

- matriz de variables indicadoras [506](#)

- PL/I [745, 751](#)

- recuperar varias filas [520](#)

#### matriz de variables de host de caracteres

- C/C++ [628](#)

- COBOL [696](#)

- PL/I [751](#)

#### matriz de variables de host numéricas

- C/C++ [628](#)

- PL/I [751](#)

#### Matriz de variables de host ROWID

- C/C++ [628](#)

- COBOL [696](#)

- PL/I [751](#)

Matriz de variables de host XML  
C/C++ 628  
COBOL 696  
PL/I 751  
matriz de variables indicadoras  
descripción 506  
insertar valores nulos 522  
matriz gráfica de variables de host  
C/C++ 628  
COBOL 696  
PL/I 751  
matriz numérica de variables de host  
COBOL 696  
matriz variable  
C/C++ 628  
COBOL 696  
PL/I 751  
mensaje  
análisis 1042  
obtención de texto  
C 603  
COBOL 654  
ensamblador 569  
Fortran 722  
PL/I 734  
mensajes  
IBM MQ 824  
migración  
aplicaciones 1  
MLS (seguridad multinivel)  
desencadenantes 176  
restricciones referenciales 140  
modificación  
datos 352  
modificar  
definición de procedimiento almacenado externo 305  
módulo de solicitud de base de datos (DBRM)  
Db2 coprocessor salida 901  
módulos de interfaz de lenguaje  
DSNCLI 926  
MQ cola de mensajes  
envío de datos de la tabla 829  
trocear documentos XML 830  
MQ Procedimientos almacenados de composición XML  
método alternativo 829  
MQ Procedimientos almacenados de descomposición XML  
método alternativo 830  
MQListener  
ejemplos 857  
mandatos 853  
MQREADALL, función de tabla 827  
MQREADALLCLOB, función de tabla 827  
MQREADCLOB, función escalar 827  
MQRECEIVE, función escalar 827  
MQRECEIVEALL, función de tabla 827  
MQRECEIVEALLCLOB, función de tabla 827  
MQRECEIVECLOB, función escalar 827  
MQSEND, función escalar 827

no registrado cronológicamente (*continuación*)  
espacios de tablas  
recuperar 35  
nombre de correlación 418  
nombre de ubicación 37  
NORMAS ACTUALES registro especial  
efecto en las restricciones de verificación 139  
uso 959  
nulo  
determinar el valor de la variable host de salida 517  
Nulo  
puntero en C 603  
Nulo, en REXX 766  
numéricos  
datos  
ancho de la columna en los resultados 1032  
números de secuencia  
Fortran 722  
PL/I 734  
Programa de aplicación COBOL 654

**O**

Objeto de secuencia  
creación 179  
referencia 471  
utilizar en varias tablas 179  
objeto grande (LOB)  
aplicaciones de ejemplo 457  
conversión de caracteres 466  
declaración de variables de host  
para precompilador 459  
declarar localizadores LOB 459  
declarar variables de referencia de archivos LOB 459  
definir y mover datos a Db2 131  
expresión 467  
locator 465  
materialización 464  
variable de indicador 466  
variable de referencia de archivo 468  
objetos  
crear en un programa de aplicación 127  
Opción de edición de enlaces AMODE 926, 989  
Opción de edición de enlaces RMODE 989  
Opción de enlace DEPLOY  
para procedimientos SQL nativos 267  
Opción de enlace DYNAMICRULES 944  
Opción de enlace KEEPDYNAMIC  
RETROTRAER 5  
Opción de precompilador APOST 913  
Opción de precompilador COMMA 913  
Opción de precompilador CONNECT 913  
Opción de precompilador DATE 913  
Opción de precompilador de COTIZACIÓN 913  
Opción de precompilador FLAG 913  
Opción de precompilador FLOAT 913  
Opción de precompilador GRAPHIC 913  
Opción de precompilador LEVEL 913  
Opción de precompilador LINECOUNT 913  
Opción de precompilador MARGINS 913  
Opción de precompilador NEWFUN 913  
Opción de precompilador NOFOR 913  
Opción de precompilador NOGRAPHIC 913  
Opción de precompilador NOOPTIONS 913

## N

nivel de aislamiento  
REXX 790  
no registrado cronológicamente

Opción de precompilador NOPADNTSTR 913  
 Opción de precompilador NOSOURCE 913  
 Opción de precompilador NOXREF 913  
 Opción de precompilador ONEPASS 913  
 Opción de precompilador PADNTSTR 913  
 Opción de precompilador PERIOD 913  
 Opción de precompilador QUOTESQL 913  
 Opción de precompilador SOURCE 913  
 Opción de precompilador SQL 913  
 Opción de precompilador STDSQL 913  
 Opción de precompilador TIME 913  
 Opción de precompilador TWOPASS 913  
 Opción de precompilador VERSION 913, 929  
 Opción de precompilador XREF 913  
 Opción de procesamiento CONNECT  
     hacer cumplir las reglas de uso restringido del sistema 38  
 Opción de tiempo de ejecución MSGFILE  
     utilizar para depurar procedimientos almacenados 1039  
 Opción del precompilador SQLLEVEL 913  
 Opción DYNAM de COBOL 654  
 Opción NODYNAM de COBOL 654  
 Opción PARMS 1004  
 Opción ROLLBACK  
     CICS Comando SYNCPOINT 25  
 opciones de enlace  
     planificar 22  
 OPCIONES opción de precompilador 913  
 OPEN  
     sentencia  
         abrir un cursor 431  
         abrir un cursor de fila 434  
         Cláusula USING DESCRIPTOR 532  
         preparado SELECCIONAR 530  
         sin marcadores de parámetros 532  
 operación de unión  
     FULL OUTER JOIN 407  
     LEFT OUTER JOIN 404  
     más de una suscripción 399  
 operando  
     expresión de tabla anidada 396  
     función de tabla definida por el usuario 396  
 Reglas SQL 408  
 RIGHT OUTER JOIN 405  
 UNIÓN INTERNA 401  
 unir tablas 396  
 unir una mesa consigo misma 401  
 ORDEN DE cláusula 366  
 ORDER BY, cláusula  
     con cláusula de PEDIDO 366  
 Sentencia SELECT 384

**P**

panel  
 DB2IMenú de opciones principal 1019  
 DSNEPRI 1019  
 DSNESP01 1019  
 DSNESP02 1025  
 DSNESP07 1029  
 EDITAR (para el conjunto de datos de entrada SPUFI) 1019  
 SPUFI 1019  
 Valores predeterminados actuales de SPUFI 1025, 1029

Panel DEFAULTS DE SPUFI 1026  
 Panel EDITAR, SPUFI  
     sentencias SQL 1019  
 paneles  
     DB2I (DB2 Interactive) 498  
     DB2I DEFAULTS 498  
     DCLGEN 498  
     DSNEDP01 498  
     DSNEOP01 498  
     DSNEOP02 498  
     REBIND PACKAGE 992  
     REBIND TRIGGER PACKAGE 994  
 paquete  
     desencadenante 172  
     ejemplos de reimpresión 948  
     identificación en tiempo de ejecución 935  
     Invalidadada 957  
     listado 933  
     no válidos 16, 20  
     reencuadernación con caracteres que coincidan con el patrón 948  
     seleccionar 935  
     ubicación 935  
     versión, identificando 929  
     vincular  
         a planes 933  
         DBRM a un paquete 927  
         remota 940  
 paquetes  
     ID de colección para paquetes de procedimientos almacenados 294  
     para procedimientos externos 290  
     para procedimientos SQL nativos 262  
     para rutinas anidadas 294  
     reagrupación gradual 950  
     reenlaces automáticos 5  
 PAQUETES ACTUALESRegistro especial  
     cambio dinámico de plan 946  
     identificar la recogida de paquetes 935  
 paquetes anulados  
     identificar 20  
 paquetes copias  
     introducción progresiva en el momento de la renovación 950  
 Parámetro del subsistema APPLCOMPAT  
     cambiar a nueva versión 889  
 PARÁMETRO ESTILO Opción SQL  
     utilizar para depurar procedimientos almacenados 1036  
 Parámetro NUMTCB 809  
 parámetros de entrada  
     procedimientos almacenados 230  
 parámetros de salida  
     procedimientos almacenados 230, 800  
 parámetros de subsistema 809  
 peticonario 37  
 PL/I  
     creación de un procedimiento almacenado 270  
 plan de aplicación  
     creación 927  
     listado de paquetes 933  
     selección dinámica de planes para aplicaciones de Internet (CICS) 946  
     vincular 933  
     volver a enlazar 951

planes  
   reenlaces automáticos 5  
 planificación  
   aplicaciones 1  
   opciones de enlace 22  
 Planificación de opciones de procesamiento de SQL para 15  
 planificación de programas de aplicación  
   Opciones de procesamiento SQL 15  
 planificación de solicitudes  
   recuperación 23  
 Plegar  
   valor de la opción de precompilador HOST 913  
   valor para C y CPP 913  
 políticas  
   IBM MQ 825  
   Interfaz de colas de mensajes (MQI) 826  
 precompilador  
   conjuntos de datos utilizados por 904  
   declaraciones de origen modificadas 908  
   descripciones de opciones 913  
   diagnósticos 908  
   ejecutar 901  
   entrada 906  
   envío de trabajos  
     Paneles de ISPF 967  
   envío de trabajos con paneles de ISPF 891  
   funciones 901  
   inicio  
     dinámicamente 907  
     JCL para procedimientos 962  
   opciones  
     acceso DRDA 925  
     CONNECT 925  
     SQL 925  
     valores por omisión 924  
   salida 908  
   tamaño máximo de entrada 906  
   vinculante en otro sistema 901  
 predicado  
   reglas generales 376  
 preparación de programas 891  
 privilegio de amortización 176  
 procedimiento almacenado  
   acceder a recursos no pertenecientes aDb2 291  
   acceder a tablas de transición 219  
   acceder CICS 291  
   acceder IMS 291  
   autorización para ejecutar 795  
   convenciones de enlaces 274  
   creación 226  
   creación de un procedimiento almacenado externo 270  
   cursos 243  
   definición de listas de parámetros 274  
   devolver datos no relacionales 293  
   ejecución de varias instancias 807  
   ejemplo 231  
   escribir 240  
   escribir en REXX 302  
   invocación desde un activador 168  
   lenguajes soportados 240  
   llamada desde un procedimiento REXX 801  
   llamada desde una aplicación 795  
   preparación 226  
   reentrante 295  
 procedimiento almacenado (*continuación*)  
   resultado de retorno establecido 293  
   Sentencia CALL 795  
   Sentencia COMMIT 243  
   sentencia ROLLBACK 243  
   terminación anómala 795  
   tipos 226  
   Tipos de datos 298  
   Tipos de datos compatibles 801  
   usando tablas temporales en 293  
   usar variables de host con 231  
   uso de registros especiales 244  
 procedimiento almacenado externo  
   Autorizaciones de paquete 290  
   creación 270  
   del servidor 290  
   ejecutándose como programa autorizado 270  
   modificar la definición 305  
   paquete 290  
   preparar 270  
   reentrante 295  
 procedimiento almacenado remoto  
   preparación del programa del cliente 805  
 Procedimiento de DSNHC 962  
 procedimiento de SQL externo  
   creación 306  
 Procedimiento DSNFOR 962  
 Procedimiento DSNHASM 962  
 Procedimiento DSNHCOB 962  
 Procedimiento DSNHCPP 962  
 Procedimiento DSNHICOB 962  
 Procedimiento DSNHPLI 962  
 procedimiento SQL  
   condiciones, manipulación 250  
   cuerpo 236  
   declaraciones permitidas 236  
   ignorar las condiciones 258  
   modificación 269  
   parameters 236  
   preparación mediante el procedimiento DSNTPSMP 311  
   variable de SQL 236  
 procedimientos  
   Creación de versiones 264  
   heredar registros especiales 215  
 procedimientos almacenados  
   Creación de procedimientos SQL nativos 244  
   depuración 1036  
   depuración con el Unified Debugger 1037  
   depuración con z/OS Debugger 1038  
   descripción 228  
   desde el procesador de línea de comandos de Db2 1011  
   ejecución simultánea 809  
   grabación de mensajes de depuración 1039  
   heredar registros especiales 215  
   ID de colección del paquete 294  
   Lista de parámetros 230  
   llamando a otros programas 294  
   migración de SQL externo a SQL nativo 307  
   paquetes para rutinas anidadas 294  
   sintaxis para invocar desde el procesador de línea de comandos Db2 1011  
   transmitir parámetros de salida de gran tamaño 800  
 Procedimientos almacenados Java  
   depuración con el Unified Debugger 1037

Procedimientos de SQL  
     Creación de versiones [264](#)  
     declaración de cursos [249](#)  
     sentencias compuestas anidadas [247](#)  
     variables [237](#)  
 procedimientos de SQL externos  
     creado usando DSNTPSMP [309](#)  
     crear usando JCL [320](#)  
     depuración con el Unified Debugger [1037](#)  
     migrar a procedimientos SQL nativos [307](#)  
 procedimientos de SQL nativos  
     creación [244](#)  
     depuración con el Unified Debugger [1037](#)  
     despliegue a otro servidor [267](#)  
     despliegue a producción [267](#)  
     ENLACAR COPIAR REEMPLAZAR [263](#)  
     ENLACE A LA COPIA [262](#)  
     migración desde procedimientos SQL externos [307](#)  
     paquetes para [262](#)  
     reemplazar paquetes para [263](#)  
 Procesador de la línea de mandatos de DB2  
     procedimientos almacenados [1011](#)  
     Sentencia CALL [1011](#)  
     vincular [930](#)  
 Procesador de procedimientos SQL (DSNTPSMP)  
     conjunto de resultados [320](#)  
 procesamiento de variables de host  
     errores [509](#)  
 procesamiento de variables de host de salida  
     errores [509](#)  
 procesamiento por lotes  
     acceso a Db2 y DL/I juntos  
     confirmaciones [474](#)  
     encuadernación de un plan [941](#)  
     llamadas en los puntos de control [474](#)  
     precompilación [901](#)  
     Db2 encia de lotes de la aplicación  
         ejecutar [1012](#)  
         empezando por un CLIST [1013](#)  
 Proceso BIND  
     datos distribuidos [937](#)  
 Programa BMP (procesamiento de mensajes por lotes)  
     puntos de comprobación [31](#)  
 programa cliente  
     preparándose para llamar a un procedimiento  
     almacenado remoto [805](#)  
 programa de aplicación  
     codificación de instrucciones SQL  
         entrada de datos [352](#)  
         seleccionar filas con un cursor [423](#)  
         SQL dinámico [524, 528](#)  
     consideraciones sobre el diseño  
         estructura [1004](#)  
         IMS llamadas [474](#)  
         llamada de sincronización cancelada [474](#)  
         Llamada XRST [474](#)  
         programación para DL/I batch [474](#)  
         punto de comprobación [474](#)  
         sentencias SQL [474](#)  
         utilizando el sistema interactivo de productividad  
             ( ISPF ) [891](#)  
     declaraciones CALL duplicadas [807](#)  
     ejecutar  
         CICS [1013](#)  
     programa de aplicación (*continuación*)  
         ejecutar (*continuación*)  
             IMS [1013](#)  
             sincronización de programas en lote DL/I [474](#)  
             TSO [1001](#)  
             TSO CLIST [1013](#)  
             entorno de prueba [1001](#)  
             extensiones de objeto [181](#)  
             Lista de materiales (BOM) [156](#)  
         preparación  
             acceso DRDA [938](#)  
             compilación [926](#)  
             DB2 opción de precompilador predeterminada [924](#)  
             Db2 precompiler Valores predeterminados de las  
             opciones [896](#)  
             definir a CICS [926](#)  
             edición de enlaces [926](#)  
             ejemplo [967](#)  
             ensamblaje [926](#)  
             mediante DB2I ( DB2 Interactive ) [891](#)  
             panel de preparación del programa [891](#)  
             preparación para correr [891](#)  
             vincular [927](#)  
             probar [1001](#)  
             procedimientos almacenados externos [240](#)  
             tabla y ver declaraciones [490](#)  
             verificar el éxito de las sentencias SQL [501](#)  
 Programa de aplicación C  
     aplicación de ejemplo [1092](#)  
     declarar mesas [603](#)  
 Programa de aplicación C/C++  
     compatibilidad de tipos de datos [644](#)  
     con clases, preparando [901](#)  
     convenio de denominación [603](#)  
     declaración de matriz de variables de indicador [639](#)  
     declaración de matriz variable [628](#)  
     declaración de variable [617](#)  
     declaración de variable indicadora [639](#)  
     declarar opiniones [603](#)  
     definición de la SQLDA [502, 615](#)  
     estructura de host [636](#)  
     INCLUDE, sentencia [603](#)  
     incluido SQLCA [614](#)  
     opción de precompilador predeterminada [924](#)  
     Soporte DCLGEN [496](#)  
     Variable de host SQLCODE [614](#)  
     Variable de host SQLSTATE [614](#)  
 Programa de aplicación COBOL  
     compatibilidad de tipos de datos [714](#)  
     compilación [926](#)  
     con clases, preparando [901](#)  
     controlar CCSID [712](#)  
     convenio de denominación [654](#)  
     Db2 opción de precompilador predeterminada [924](#)  
     declaración de matriz de variables de indicador [711](#)  
     declaración de matriz variable [696](#)  
     declaración de variable [686](#)  
     declaración de variable indicadora [711](#)  
     declarar mesas [654](#)  
     declarar opiniones [654](#)  
     definición de la SQLDA [502, 685](#)  
     estructura de host [705](#)  
     extensiones orientadas a objetos [720](#)  
     INCLUDE, sentencia [654](#)

**Programa de aplicación COBOL (*continuación*)**  
 incluido SQLCA 684  
 matriz de variables de host, declarando 686  
 opciones 654  
 preparación 926  
 programa de ejemplo 659  
 restablecer SQL-INIT-FLAG 654  
 Soporte DCLGEN 496  
 SQL dinámico 528  
 Variable de host SQLCODE 684  
 Variable de host SQLSTATE 684  
 variable de host, declarando 686  
 variable del lenguaje principal  
     uso de guiones 654  
 WHENEVER, sentencia 654  
**programa de aplicación ensamblador**  
     compatibilidad de tipos de datos 594  
     declaración de variable 588  
     declaración de variable indicadora 593  
     declarar mesas 582  
     declarar opiniones 582  
     definición de la SQLDA 502, 586  
     ensamblaje 926  
 INCLUDE, sentencia 582  
 incluido SQLCA 585  
 reentrant 582  
 Variable de host SQLCODE 585  
 Variable de host SQLSTATE 585  
 variable de host, declarando 587  
 variable del lenguaje principal  
     convenio de denominación 582  
**Programa de aplicación PL/I**  
     compatibilidad de tipos de datos 760  
     consideraciones de codificación 734  
 Constantes DBCS 734  
     convenio de denominación 734  
     declaración de matriz de variables de indicador 759  
     declaración de matriz variable 751  
     declaración de variable 745  
     declaración de variable indicadora 759  
     declarar mesas 734  
     declarar opiniones 734  
     definición de la SQLDA 502, 744  
     estructura de host 757  
     etiquetas de extracto 734  
 INCLUDE, sentencia 734  
 incluido SQLCA 743  
     matriz de variables de host, declarando 745  
 Soporte DCLGEN 496  
     Variable de host SQLCODE 743  
     Variable de host SQLSTATE 743  
     variable de host, declarando 745  
 WHENEVER, sentencia 734  
**Programa de aplicación REXX**  
 incluido SQLCA 784  
     Variable de host SQLCODE 784  
     Variable de host SQLSTATE 784  
**programa de ejemplo**  
     DSN8BC3 654  
     DSN8BD3 603  
     DSN8BE3 603  
     DSN8BF3 722  
     DSN8BP3 734  
**Programa de ejemplo DSNTIAD**  
     cómo ejecutar 1074  
     especificar terminador SQL 1081  
     parameters 1074  
     preparación de programas 1074  
**Programa de ejemplo DSNTIAUL**  
     cómo ejecutar 1074  
     parameters 1074  
     preparación de programas 1074  
 programa de monitorización de terminales (TMP) 1001  
**Programa de muestra DSNTIJS**  
     usando para configurar el Unified Debugger 1037  
**Programa MPP**  
     puntos de comprobación 31  
**programa orientado a objetos, preparación** 901  
**Programa REXX**  
     convenio de denominación 766  
     conversión de tipo de datos 785  
     cursores de nombres 792  
     datos de entrada de caracteres 789  
     DSNREXX 787  
     ejecutar 1004  
     Etiqueta de sentencia 766  
     interfaz de programación de aplicaciones  
         CONNECT 787  
         DISCONNECT 787  
         EXECSQL 787  
     manejo de errores 766  
     nivel de aislamiento 790  
     nombrar declaraciones preparadas 792  
     SQLDA 502, 785  
     Tipo de datos de entrada 785  
**programación de aplicaciones**  
     Declaraciones de variables DCLGEN 496  
     Ejemplo DCLGEN 498  
     rendimiento 22  
**programas**  
     muestras suministradas con Db2 1053  
**programas de aplicación**  
     compatibilidad 867, 878, 884  
     estructuras de lenguaje principal 506  
     matrices de variables de lenguaje principal 504  
     rendimiento 473  
     Tipos de datos compatibles 511  
     variables del lenguaje principal 503  
**Programas de aplicación C/C++**  
     variables de host de puntero 642  
**programas de aplicación suministrados** 1053  
**Programas IMS**  
     recuperación 33  
**propiedades de la conexión**  
     recurso de conexión de llamada (CAF) 43  
     Recurso de conexión de Resource Recovery Services (RRSAF) 75  
**punto de comprobación**  
     especificar frecuencia 31  
     llamadas 29, 31  
**punto de confirmación**  
     descripción 24  
     IMS unidad de trabajo 29  
**Punto de entrada DSNHLI a DSNALI**  
     ejemplo de programa 66  
**punto y coma**  
     incorporado 1081

punto y coma (*continuación*)  
terminador de la instrucción SPUFI predeterminado 1025  
punto y coma incrustado incorporado 1081  
puntos de salvaguarda 33

## R

recogida, paquete identificar 935  
SET CURRENT PACKAGESET, sentencia 935  
recuperación datos en ASCII de Db2 for z/OS 532  
datos en Unicode de Db2 for z/OS 532  
datos utilizando SELECT \* 412  
datos, cambiando el CCSID 532  
grandes volúmenes de datos 471  
planificación para su solicitud 23  
Programas IMS 26, 33  
varias filas en matrices de variables de host 520  
recuperar espacios de tabla que no están registrados 35  
RECUPERAR ACTUALIZAR CONTINUAR 445  
RECUPERAR CON CONTINUAR 445  
recuperar una sola fila en variables de host 514  
recurso de conexión opciones en un entorno de z/OS 39  
Recurso de conexión CICS control desde aplicaciones 124  
detectar si está operativo 124  
detención 124  
inicio 124  
recurso de conexión de llamada (CAF) ámbito 43  
atención rutinas de salida 44  
casos de ejemplo 65  
códigos de devolución y códigos de motivo 64  
códigos de retorno ejemplo de comprobación 66  
conexiones implícitas con 48  
descripción 42  
funciones de conexión 52  
ID de autorización 43  
invocar 40  
mensajes de error 63  
Nombre de conexión 43  
parámetros para CALL DSNALI 49  
programa de aplicación ejemplos 66  
preparación 47  
propiedades de la conexión 43  
registrar cambios 48  
requisitos del programa 47  
resumen de comportamiento 51  
rutinas de recuperación 45  
tarea finalizada 43  
Terminaciones anómalas de Db2 43  
tipo de conexión 43  
trace 63  
Recurso de conexión de Resource Recovery Services (RRSAF)

(RRSAF) (*continuación*)  
ámbito 75  
cargando 77  
casos de ejemplo 117  
códigos de devolución y códigos de motivo 117  
conexiones implícitas 79  
convenciones de registro 79  
descripción 73  
disponibles 77  
ejemplos de programas 119  
funciones de conexión 83  
ID de autorización 75  
invocar 71  
JCL de muestra 119  
Nombre de conexión 75  
parámetros para CALL DSNRLI 80  
programa de aplicación preparación 79  
propiedades de la conexión 75  
requisitos del programa 79  
resumen de comportamiento 81  
tarea finalizada 75  
Terminaciones anómalas de Db2 75  
tipo de conexión 75  
recursos no-Db2 es acceso desde procedimiento almacenado 291  
reenlaces automatic 5  
reenlaces automáticos planes y paquetes antiguos 5  
reescritura automática de consulta 140  
referencia correlacionada nombre de correlación 418  
Reglas SQL 396  
usar en subconsulta 418  
uso 396  
Registro especial comportamiento en funciones definidas por el usuario y procedimientos almacenados 215  
comportamiento en procedimientos almacenados 244  
CURRENT PACKAGE PATH 934  
CURRENT PACKAGESET 934  
CURRENT RULES 959  
Registros cambiado por CAF (call attachment facility) 48  
reiniciar, programas por lotes DL/I utilizando JCL 1009  
RELEASE SAVEPOINT, sentencia 33  
rendimiento afectado por Estructura de aplicación 1004  
programación de aplicaciones 22  
programas de aplicación 473  
resolución de función 479  
resolución de problemas errores para las variables de host de salida 509  
restablecimiento de bloques de control CAF 61  
restricción de comprobación consideraciones 138  
consideraciones sobre programación 1050  
definición 137  
descripción 137  
determinación de infracciones 1050  
enforcement 137

restricción de comprobación (*continuación*)  
     NORMAS ACTUALES efecto de registro especial [139](#)  
 restricción referencial  
     definición [139](#)  
     descripción [139](#)  
     determinación de infracciones [1050](#)  
     en mesas con seguridad multinivel [140](#)  
     en tablas con cifrado de datos [144](#)  
     informativo [140](#)  
     nombre [143](#)  
 restricción referencial informativa  
     descripción [140](#)  
     reescritura automática de consulta [140](#)  
**Restricted System**  
     definición [38](#)  
     normas de actualización [38](#)  
     reglas de forzado [38](#)  
 resultados, tabla  
     formatear [381](#)  
 Resumen de valores de grupo [393](#)  
 revinculación automática  
     condiciones de [957](#)  
     paquete no válido [957](#)  
     SQLCA no disponible [957](#)  
**REXX**  
     creación de un procedimiento almacenado [270](#)  
**RIB (bloque de información de liberación)**  
     Función CONNECT de CAF [53](#)  
     Función IDENTIFICAR de RRSAF [83](#)  
     Función SET\_REPLICATION de RRSAF [108](#)  
**RID**  
     para acceso directo a la fila [455](#)  
**rollback**  
     cambios dentro de una unidad de trabajo [33](#)  
**ROW CHANGE TIMESTAMP** [395](#)  
**ROW\_NUMBER** [385](#)  
**ROWID**  
     insertar en la tabla [150](#)  
     tipo de datos [129](#)  
**rutina de recuperación funcional (FRR)**  
     en CAF [45](#)  
**rutina de recuperación nocturna**  
     en CAF [45](#)  
**rutina de salida**  
     atención al cliente con CAF [44](#)  
     recuperación nocturna con CAF [45](#)  
**rutinas**  
     heredar registros especiales [215](#)

**S**

**Salida del precompilador SYSPRINT**  
     sección de declaraciones de origen, ejemplo [1042](#)  
     sección de referencias cruzadas de símbolos [1042](#)  
     sección de resumen, ejemplo [1042](#)  
     Sección Opciones [1042](#)  
     utilizado para analizar errores [1042](#)  
**Salida SYSTEM para analizar errores** [1042](#)  
**SAVEPOINT, sentencia** [33](#)  
**seguridad a nivel de fila** [140](#)  
**selección dinámica de planes**  
     restricciones con PAQUETES ACTUALESregistro especial  
         [946](#)  
     usando paquetes con [946](#)

seleccionar  
     algunas columnas [374](#)  
     columnas nombradas [374](#)  
     columnas sin nombre [380](#)  
     filas [374](#)  
     todas las columnas [374](#)  
**SELECCIONAR DE INSERTAR declaración**  
     ANTES de los valores desencadenantes [361](#)  
     descripción [361](#)  
     insertar a la vista [361](#)  
     recuperación  
         ANTES de los valores desencadenantes [361](#)  
         registros especiales [361](#)  
         valores generados [361](#)  
         valores predeterminados [361](#)  
         Varias filas [361](#)  
     tabla de resultados [361](#)  
     usando SELECT INTO [361](#)  
     valores predeterminados [361](#)  
     Varias filas  
         efecto de CON RETENCIÓN [361](#)  
         efecto de los cambios [361](#)  
         efecto de SAVEPOINT y ROLLBACK [361](#)  
         errores de procesamiento [361](#)  
         sensibilidad del cursor [361](#)  
         tabla de resultados del cursor [361](#)  
         usando el cursor [361](#)  
         usando FETCH FIRST [361](#)  
         usando SECUENCIA DE ENTRADA [361](#)  
**SELECT INTO**  
     usar con variables de host [514](#)  
**sentencia ALTER PROCEDURE**  
     procedimiento almacenado externo [305](#)  
**Sentencia CALL**  
     ejemplos [795](#)  
     múltiple [807](#)  
     Procesador de la línea de mandatos de DB2 [1011](#)  
     sintaxis para invocar DSNTPSMP [313](#)  
**Sentencia COMMIT**  
     con RRSAF [73](#)  
     cuándo emitir [24](#)  
     descripción [1023](#)  
     en un procedimiento almacenado [243](#)  
**sentencia compuesta**  
     ejemplo  
         instrucciones IF y WHILE anidadas [238](#)  
         SQL dinámico [239](#)  
     etiquetas [236](#)  
     Manejador EXIT [250](#)  
**sentencia CREATE GLOBAL TEMPORARY TABLE** [146](#)  
**Sentencia CREATE TABLE**  
     cláusula DEFAULT [127](#)  
     cláusula PRIMARY KEY [141](#)  
     Cláusula ÚNICA [127, 141](#)  
     nombres de relaciones [143](#)  
     NOT NULL, cláusula [127](#)  
     uso [127](#)  
**Sentencia CREATE VIEW** [152](#)  
**Sentencia de procedimiento SQL**  
     CONTINUAR gestor [250](#)  
     manejador [250](#)  
     Manejador EXIT [250](#)  
     manejo de errores [250](#)  
**sentencia DECLARE TABLE**

sentencia DECLARE TABLE (*continuación*)  
     C 603  
     COBOL 654  
     en programas de aplicación 489  
     ensamblador 582  
     Fortran 722  
     PL/I 734  
 sentencia DELETE  
     de posición  
         DÓNDE cláusula CURRENT 431, 435  
         PARA LA FILA n DE LA cláusula ROWSET 435  
             restricciones 431  
     descripción 358, 370  
     subconsulta correlacionada 418  
 Sentencia Drop table 152  
 Sentencia FETCH  
     Cláusula USING DESCRIPTOR 532  
     descripción, una sola fila 431  
     descripción, varias filas 435  
     desplazarse por los datos 448  
     orientación de captación 439  
     posicionamiento de filas y conjuntos de filas 451  
     usando el cursor posicionado en fila 431  
     variables del lenguaje principal 530  
     varias filas  
         Cláusula FOR n ROWS 438  
         descripción 435  
         ensamblador 582  
         número de filas en el conjunto de filas 438  
         usar con descriptor 435  
         uso con matrices de variables de host 435  
 Sentencia GET DIAGNOSTICS  
     artículos de conexión 563  
     descripción 563  
     Elemento RETURN\_STATUS 259  
     Elemento ROW\_COUNT 435  
     Elementos de condición 563  
     iINSERTAR varias filas 563  
     partidas de estado de cuenta 563  
     tipos de datos para artículos 563, 565  
     usar en controlador 258  
 Sentencia GRANT 1017  
 Sentencia INSERT  
     con columna de identidad 356  
     con la columna ROWID 355  
     descripción 352  
     Fila única 352  
     VALUES, cláusula 352  
     Varias filas 354  
 Sentencia PREPARE  
     Cláusula INTO 532  
     ejecución dinámica 549  
     variable del lenguaje principal 530  
 Sentencia RESIGNAL  
     establecer valor SQLSTATE 260  
     plantear una condición 259  
 Sentencia RETURN  
     devolver el estado del procedimiento SQL 810  
 sentencia ROLLBACK  
     Cláusula PARA GUARDAR PUNTO 33  
     con RRSAF 73  
     cuándo emitir 24  
     descripción 1023  
     en un procedimiento almacenado 243  
     sentencia ROLLBACK (*continuación*)  
         error en IMS 474  
 Sentencia SELECT  
     AS, cláusula  
         con cláusula ORDER BY 384  
     cambiar el formato de los resultados 1032  
     cláusulas  
         DISTINCT 382  
         EXCEPT 389  
         FROM 374  
         GROUP BY 393  
         HAVING 394  
         INTERSECT 389  
         ORDER BY 384  
         UNION 389  
         WHERE 376  
     columna derivada con cláusula AS 380  
     columnas nombradas 374  
     columnas sin nombre 380  
     condición de búsqueda 412  
     filtrado por hora cambiado 395  
     lista fija 530  
     lista variable 532  
     marcadores de parámetro 532  
     ORDER BY, cláusula  
         columnas derivadas 384  
         con cláusula AS 384  
     seleccionar un conjunto de filas 423  
     subconsultas 412  
     utilizar con  
         \* (para seleccionar todas las columnas) 374  
     DECLARE CURSOR, sentencia 429, 434  
     lista de nombres de columnas 374  
 sentencia SELECT FROM MERGE  
     con la cláusula INCLUDE 359  
     descripción 359  
 sentencia SET ENCRYPTION PASSWORD 144  
 Sentencia SIGNAL  
     configuración del mensaje de estado 260  
     plantear una condición 259  
 Sentencia UPDATE  
     Cláusula SET 367  
     de posición  
         DÓNDE cláusula CURRENT 431, 435  
         PARA LA FILA n DEL CONJUNTO DE FILAS 435  
             restricciones 431  
         descripción 367  
         subconsultas correlacionadas 418  
 sentencias compuestas  
     anidado 247  
     dentro de la declaración de un gestor de condiciones 251  
 sentencias compuestas anidadas  
     ámbito de variables 245  
     Declaraciones de cursor 249  
     definición 247  
     etiquetas de extracto 247  
     para controlar el alcance de las condiciones 251  
 sentencias SQL  
     ALTER FUNCTION 189  
     CLOSE 433, 438, 530  
     códigos de error de retorno 558  
     comentarios  
         C 603

sentencias SQL (*continuación*)  
comentarios (*continuación*)  
COBOL 654  
ensamblador 582  
Fortran 722  
PL/I 734  
REXX 766  
CONECTARSE, con acceso DRDA 820  
continuación  
C 603  
COBOL 654  
ensamblador 582  
Fortran 722  
PL/I 734  
REXX 766  
CREATE FUNCTION 189  
DECLARE CURSOR  
descripción 429, 434  
ejemplo 530, 532  
DELETE  
descripción 431  
ejemplo 370  
DESCRIBE 532  
establecer símbolos 582  
etiquetas  
C 603  
COBOL 654  
ensamblador 582  
Fortran 722  
PL/I 734  
REXX 766  
EXECUTE 549  
EXECUTE IMMEDIATE 548  
FETCH  
descripción 431, 435  
ejemplo 530  
Fortran secciones del programa 722  
incorporado 906  
INSERT 352  
LIBERACIÓN, con acceso DRDA 821  
márgenes  
C 603  
COBOL 654  
ensamblador 582  
Fortran 722  
PL/I 734  
REXX 766  
MERGE  
ejemplo 358  
OPEN  
descripción 431, 434  
ejemplo 530  
PREPARE 549  
Secciones del programa COBOL 654  
Secciones del programa PL/I 734  
Secciones del programa REXX 766  
SELECCIONAR DE ACTUALIZAR 369  
SELECCIONAR DE BORRAR 372  
SELECCIONAR DE FUSIONAR 359  
SELECCIONAR DE INSERTAR 361  
SELECT  
descripción 376  
unir tablas 396  
unir una mesa consigo misma 401

sentencias SQL (*continuación*)  
UPDATE  
descripción 431, 435  
ejemplo 367  
verificar la ejecución correcta 501  
WHENEVER 562  
serie  
tipo de datos 129  
serie de caracteres  
ancho de la columna en los resultados 1026, 1032  
datos mixtos 129  
services  
IBM MQ 825  
Interfaz de colas de mensajes (MQI) 826  
Servicios ISPLINK SELECT 1004  
servidor 37  
SET CURRENT PACKAGESET, sentencia 935  
SET\_CLIENT\_ID (función de conexión de RRSAF)  
ejemplos de idiomas 105  
sintaxis 105  
SET\_ID (función de conexión de RRSAF)  
ejemplos de idiomas 104  
sintaxis 104  
SET\_REPLICATION (función de conexión de RRSAF)  
ejemplos de idiomas 108  
sintaxis 108  
SIGNON (función de conexión de RRSAF)  
ejemplo de programa 119  
ejemplos de idiomas 89  
sintaxis 89  
símbolos de PSPI 1525  
sistemas restringidos 36  
solicitud de organización  
ejemplos 1092  
solicitud del proyecto, descripción 1092  
SOME, predicado cuantificado 415  
SPUFI  
cambiar el ancho de las columnas 1032  
Código SQL devuelto 1032  
CONECTAR SITUACIÓN campo 1023  
configuración del terminador SQL 1030  
Db2 regulador 1019  
encabezado de columna creado 1033  
especificación del terminador de la instrucción SQL 1025  
introducir comentarios 1023  
paneles  
asigna el conjunto de datos RESULT 1023  
formato y salida de visualización 1032  
rellenar 1019  
seleccionando en el menú de DB2I 1019  
valores anteriores mostrados en el panel 1019  
Procesamiento de sentencias SQL 1019  
resultados de la navegación 1032  
valores predeterminados 1025  
SQL (Lenguaje de consulta estructurado)  
codificar  
dinámica 528  
extensiones de objeto 181  
códigos de retorno  
comprobar 557  
gestión 558  
comprobación de sintaxis 822  
comprobar ejecución 556

SQL (Lenguaje de consulta estructurado) (*continuación*)  
cursos 423  
delimitador de serie 972  
dinámica  
codificar 524  
programa de muestra C 606  
lista variable 532  
Terminador de sentencias: 1081

SQL dinámico  
declaraciones no SELECT 529, 549  
declaraciones SELECT de lista variable 532  
descripción 524  
efecto de la opción de enlace REOPT(ALWAYS) 532  
efecto del cursor CON RETENCIÓN 549  
EXECUTE IMMEDIATE, sentencia 548  
Fortran programa 722  
idiomas de acogida 528  
instrucciones SELECT de lista fija 530  
PL/I 532  
Preparar y ejecutar 549  
Programa C 532  
Programa COBOL 528  
Programa de aplicación COBOL 654  
programa de muestra C 606  
programa ensamblador 532  
programación 524  
requisitos 525  
restrictiones 525  
Ventajas e inconvenientes 525

SQL estático  
descripción 524  
Programa de aplicación C/C+  
+  
ejemplos 606  
programa de muestra C 606  
variables del lenguaje principal 525

sQL recursivo  
bucles infinitos 410  
descripción 410  
ejemplos 156  
explosión de un solo nivel 156  
explosión resumida 156  
profundidad de control 156  
Reglas 410

SQL-INIT-FLAG, restablecimiento 654

SQLCA (área de comunicación de SQL)  
comprobación SQLWARN0 557  
descripción 557  
programa de muestra C 606

Subrutina DSNTIAC  
C 603  
COBOL 654  
ensamblador 582  
PL/I 734

Subrutina DSNTIAR  
C 603  
COBOL 654  
ensamblador 569  
Fortran 722  
PL/I 734  
verificación de SQLCODE 561  
verificación de SQLERRD(3) 557  
verificación de SQLSTATE 561

SQLCA (área de comunicaciones SQL) (*continuación*)  
C/C++ 614  
COBOL 684  
decidir si incluir 501  
ensamblador 585  
Fortran 724  
PL/I 743  
REXX 784

SQLCODE  
-923 960  
-925 474  
-926 474  
+100 562  
+802 570  
valores 561

SQLDA  
columna XML 532  
establecer una variable de host XML 532

SQLDA (área de descriptores de SQL)  
almacenar información de marcadores de parámetros 555  
asignación de almacenamiento 435, 532  
C 532  
C/C++ 502, 615  
COBOL 502, 685  
configuración de campos de salida 435  
declaración FETCH de varias filas 435  
declarar 435  
ejemplo SELECT dinámico 532  
ensamblador 502, 586  
Fortran 502, 725  
instrucción SELECT de lista variable 532  
marcadores de parámetro 532  
OPEN, sentencia 530  
para LOB y tipos distintos 532  
PL/I 502, 532, 744  
programa ensamblador 532  
requiere direcciones de almacenamiento 532  
REXX 502, 785  
sin apariciones de SQLVAR 532

SQLRULES, opción del subcomando BIND PLAN 959

SQLSTATE  
"2D521" 474  
«01519» 570  
«57015» 960  
consumidor de servicios web 863  
valores 561

SSID (identificador de subsistema), especificando 971

Subcomando RUN de DSN  
ejecutar un programa en primer plano en TSO 1001  
procesamiento del código de devolución 1001

subconsulta  
correlacionadas  
descripción 417  
ejemplo 417  
sentencia DELETE 418  
Sentencia UPDATE 418  
descripción 412  
predicado básico 415  
predicado cuantificado 415  
Predicado EXISTS 415  
Predicado IN 415  
restrictiones con BORRAR 419  
restrictiones referenciales 419

subconsulta (*continuación*)  
sentencia DELETE 418  
Sentencia UPDATE 418  
visión general de los conceptos 412  
submandato BIND PACKAGE de DSN  
opciones  
    CURRENTDATA 939  
    ENCODING 940  
    nombre-ubicación 939  
    OPTIONS 940  
    SQLERROR 939  
opciones asociadas con el acceso DRDA 938, 941  
remota 940  
submandato BIND PLAN de DSN  
opciones  
    CURRENTDATA 939  
    DISCONNECT 938  
    ENCODING 939  
    SQLRULES 939, 959  
opciones asociadas con el acceso DRDA 938  
submandato REBIND PACKAGE de DSN  
generar lista de 953  
reencuadernación con caracteres comodín 948  
remota 940  
submandato REBIND PLAN de DSN  
generar lista de 953  
opciones  
    NOPKLIST 951  
    PKLIST 951  
submandato REBIND TRIGGER PACKAGE de DSN 172  
Subrutina DSNTIAC  
C 603  
COBOL 654  
ensamblador 582  
PL/I 734  
Subrutina DSNTIAR  
C 603  
COBOL 654  
códigos de retorno 560  
descripción 558  
ensamblador 569  
Fortran 722  
PL/I 734  
utilización 558  
Subrutina DSNTIR 722  
subsistema  
    identificador (SSID), especificando 971  
suprimir  
    cada fila de una tabla  
        con TRUNCATE 370  
    datos 370  
    filas actuales 431  
    filas de una tabla 370  
SYSDUMMY1 1073  
SYSDUMMYA 1073  
SYSDUMMYE 1073  
SYSDUMMYU 1073  
SYSIBM.MQPOLICY\_TABLE  
    descripciones de columnas 830  
SYSIBM.MQSERVICE\_TABLE  
    descripciones de columnas 830

## T

tabla  
actualización de filas 367  
copiar desde ubicaciones remotas 817  
declarar en un programa 489  
definición incompleta de 150  
dependiente, restricciones de ciclo 139  
DROP, sentencia 152  
estructura referencial 139  
Fusionar fila 358  
Inserción de varias filas 354  
insertar una sola fila 352  
modificación  
    definiciones cambiantes 144  
    usando CREATE y ALTER 570  
mostrando, lista de 373  
recuperación 423  
rellenar 1018  
rellenar con datos de prueba 1018  
seleccionar valores a medida que actualiza filas 369  
seleccionar valores a medida que inserta filas 361  
seleccionar valores al eliminar filas 372  
seleccionar valores al fusionar filas 359  
supresión de filas 370  
temporal 147  
    usando nombres de tabla de tres partes 817  
tabla de actividades de ejemplo 1053  
tabla de actividades de proyectos de ejemplo 1062  
tabla de catálogo  
    SYSIBM.LOCATIONS 821  
    SYSIBM.SYSCOLUMNS 373  
    SYSIBM.SYSTABAUTH 373  
tabla de departamentos de ejemplo  
    creación 144  
tabla de empleados de actividades de proyectos de ejemplo 1063  
tabla de empleados de ejemplo 1056  
tabla de fotografías y currículums de empleados 1060  
tabla de proyectos de ejemplo 1061  
tabla de resultados  
    de la sentencia SELECT 381  
    descripción 381  
    ejemplo 381  
    numeral filas 385  
    sólo lectura 429  
tabla de transición, desencadenante 160  
Tabla DSN\_FUNCTION\_TABLE 482  
tabla temporal  
    trabajar con 147  
    ventajas de 148  
tabla temporal creada  
    instancias 146  
    trabajar con 147  
tabla temporal declarada  
    acceso remoto mediante un nombre de tres partes 817  
    calificador para 148  
    Cláusula ON COMMIT 150  
    INCLUDING COLUMN DEFAULTS 148  
    incluidas las columnas de identidad 148  
    instancias 148  
    requisitos 148  
    trabajar con 147  
tabla y ver declaraciones

tabla y ver declaraciones (*continuación*)  
     incluido en un programa de aplicación 498

tablas  
     creación para la integridad de los datos 136

tablas de ejemplo  
     almacenamiento 1070  
     DSN8C10.ACT (actividad) 1053  
     DSN8C10.DEMO\_UNICODE (ejemplo Unicode) 1064  
     DSN8C10.DEPT (departamento) 1054  
     DSN8C10.EMP (empleado) 1056  
     DSN8C10.EMP\_PHOTO\_RESUME (fotografía de y currículum de empleado) 1060  
     DSN8C10.EMPPROJECT (empleado de actividad para proyecto) 1063  
     DSN8C10.PROJ (proyecto) 1061  
     PROJACT (actividad de proyecto) 1062  
     relaciones 1065  
     vistas 1066

tablas de prueba 1015

Tablas MQ de Db2  
     descripciones 830

TCB (bloque de control de tareas)  
     capacidades con CAF 42  
     capacidades con RRSAF 73

teclas de acceso directo  
     teclado xiii

terminación anómala  
     efecto en la posición del cursor 427

IMS  
     U0102 1009  
     para llamadas de sincronización 474

sistema  
     X"04E" 474

Terminador de instrucción SQL  
     especificando en SPUFI 1025  
     modificación en DSNTIAD 1081  
     modificar en DSNTEP2 y DSNTEP4 1076, 1084  
     modificar en SPUFI 1025

Terminador SQL, especificando en DSNTEP2 y DSNTEP4 1076, 1084

Terminador SQL, especificando en DSNTIAD 1081

TERMINAR HILO (función de conexión de RRSAF)  
     ejemplo de programa 119  
     ejemplos de idiomas 112  
     sintaxis 112

TERMINAR IDENTIFICAR (función de conexión de RRSAF)  
     ejemplo de programa 119

TERMINATE IDENTIFY (función de conexión de RRSAF)  
     ejemplos de idiomas 113  
     sintaxis 113

tipo de datos  
     comparaciones 514  
     compatibilidad  
         COBOL y SQL 714  
         Fortran y SQL 731  
         Programa de aplicación C 644  
         programa de aplicación ensamblador 594  
         Programa de aplicación PL/I 760  
         REXX y SQL 785  
         incorporadas 129

tipo de datos dateTime 129

tipo diferenciado  
     Argumentos de la función 482  
     asignación de valores 357

tipo diferenciado (*continuación*)  
     comparar tipos 421  
     con EXCEPTO 420  
     con UNIÓN 420  
     descripción 182  
     ejemplo  
         argumento de función definida por el usuario (UDF) 183  
         argumentos de la función de casting 483  
         argumentos del operador infijo 483  
         constantes de fundición 483  
         Tipo de datos LOB 183  
         variables de host de lanzamiento 483  
     tipificación firme 182  
     UNIÓN con INTERSECCIÓN 420

tipos de datos  
     compatibilidad 511  
     utilizado por DCLGEN 496

tipos de datos del idioma de destino  
     compatibilidad con tipos de datos SQL 511

Tipos de datos SQL  
     compatibilidad con los tipos de datos del idioma de destino 511

tipos diferenciados  
     creación 181

TMP (programa de monitorización de terminales)  
     en funcionamiento bajo TSO 1012  
     Procesador de mandatos de DSN 1001

TODOS los predicados cuantificados 415

TRADUCIR (función de conexión de CAF)  
     descripción 52  
     ejemplo de idioma 62  
     ejemplo de programa 66  
     sintaxis 62

TRADUCIR (función de conexión de RRSAF)  
     ejemplos de idiomas 114  
     sintaxis 114

traducir solicitudes a SQL 571

trituración de documentos XML de mensajes de correo electrónico (MQ) 830

Truncado  
     determinar el valor de la variable host de salida 517

TRUNCATE  
     ejemplo 370

TSO  
     CLISTS  
         en primer plano 1013  
         llamada a programas de aplicación 1013  
     Comando TEST 1045

## U

un marcador de parámetro  
     con declaraciones arbitrarias 532  
     lanzamiento en la invocación de funciones 484  
     Más de una 549  
     SQL dinámico 549  
     valores proporcionados por OPEN 530

Unicode  
     datos, recuperando de Db2 for z/OS 532  
     tabla de ejemplo 1064

unidad de trabajo  
     CICS 25

unidad de trabajo (*continuación*)  
descripción 23  
deshacer cambios dentro de 33  
finalización  
abrir cursos 427  
IMS 29  
TSO 24  
Unified Debugger  
configuración 1037  
Depuración de procedimientos almacenados 1037  
UNION  
mantener filas duplicadas con TODOS 389  
unión externa  
FULL OUTER JOIN 407  
LEFT OUTER JOIN 404  
RIGHT OUTER JOIN 405  
USER, registro especial  
valor en la declaración UPDATE 367  
valor en la instrucción INSERT 127  
Utilidad PRELINK 967

## V

valor nulo  
determinación del valor de la columna 379  
insertar en columnas 522  
valor de columna de la instrucción UPDATE 367  
valores calculados  
grupos con condiciones 394  
Resumen de valores de grupo 393  
Valores LOB  
captación 445  
Valores XML  
captación 445  
varbinary matriz de variables de host  
C/C++ 628  
PL/I 751  
Variable  
C/C++ 617  
COBOL 686  
declarar en el procedimiento SQL 236  
ensamblador 588  
Fortran 727  
PL/I 745  
variable binaria de host  
C/C++ 617  
variable de host binaria  
COBOL 686  
ensamblador 588  
PL/I 745  
variable de host de caracteres  
C/C++ 617  
COBOL 686  
ensamblador 588  
Fortran 727  
PL/I 745  
variable de host de puntero de carácter delimitado  
declarar 642  
descripción 642  
referencias en sentencias SQL 641  
variable de host de puntero escalar  
declarar 642  
referencias en sentencias SQL 641  
variable de host de salida  
variable de host de salida (*continuación*)  
determinar si está truncado 517  
determinar si nulo 517  
variable de host del puntero de matriz  
declarar 642  
referencias en sentencias SQL 641  
variable de host numérica  
C/C++ 617  
COBOL 686  
ensamblador 588  
Fortran 727  
PL/I 745  
Variable de host SQLCODE  
decidir si declarar 501  
Variable de host SQLSTATE  
decidir si declarar 501  
variable de host varbinary  
C/C++ 617  
COBOL 686  
ensamblador 588  
PL/I 745  
Variable de host XML  
SQLDA 532  
variable de indicador  
descripción 506  
insertar valores nulos 522  
variable de referencia de archivo  
Db2-constructo generado 469  
Variable de referencia de archivo LOB  
C/C++ 617, 628  
COBOL 686, 696  
ensamblador 588  
PL/I 745, 751  
Variable de referencia de archivo XML  
C/C++ 617, 628  
COBOL 686, 696  
ensamblador 588  
PL/I 745, 751  
variable de SQL 236  
variable de transición, desencadenante 160  
variable del lenguaje principal  
actualización de valores en tablas 518  
C/C++ 617  
COBOL 686  
descripción 503  
ensamblador 587, 588  
establecer CCSID 508  
Fortran 726, 727  
insertar valores en tablas 519  
LOB  
C 460  
COBOL 461  
ensamblador 459  
Fortran 462  
PL/I 462  
PL/I 745  
recuperar una sola fila 514  
Sentencia FETCH 530  
Sentencia PREPARE 530  
sQL estático flexibilidad 525  
utilización 514  
variable de indicador 506  
variable gráfica del host  
C/C++ 617

variable gráfica del host (*continuación*)  
 COBOL 686  
 ensamblador 588  
 PL/I 745

Variable LOB  
 C/C++ 617  
 COBOL 686  
 ensamblador 588  
 Fortran 727  
 PL/I 745

Variable ROWID  
 C/C++ 617  
 COBOL 686  
 ensamblador 588  
 Fortran 727  
 PL/I 745

Variable XML  
 C/C++ 617  
 COBOL 686  
 ensamblador 588  
 PL/I 745

variables  
 en procedimientos SQL 237

variables de host de puntero  
 declarar 642  
 referencias en sentencias SQL 641

variables de indicador  
 utilizado para pasar parámetros de salida grandes 800

variables del lenguaje principal  
 Tipos de datos compatibles 511  
 XML en aplicaciones SQL incrustadas 572  
 XML en COBOL 574, 575  
 XML en ensamblador 572, 573  
 XML en lenguaje C 573, 574  
 XML en PL/I 575–577

variables indicadoras Sintaxis COBOL 711

variables indicadoras sintaxis de Fortran 729

variables indicadoras. Sintaxis del ensamblador 593

versión  
 cambio para el procedimiento SQL 269

versión de un paquete 929

versiones  
 procedimientos 264

vía de acceso  
 acceso directo de fila 455

vincular  
 cambios que requieren 16  
 comprobar las opciones de ENVIAR PAQUETE 941  
 DBRM precompilados en otro lugar 901  
 especificar reglas SQL 959  
 opciones asociadas con el acceso DRDA 938

paquetes  
 remota 940  
 planes 933  
 planes de aplicaciones 927  
 requisitos del paquete remoto 940

vista  
 columnas de identidad 153  
 contenido 151, 153  
 datos de resumen 153  
 declarar en un programa 489  
 descartar 154  
 descripción 152  
 recuperación 423

vista (*continuación*)  
 referencias a registros especiales 153  
 unión de dos o más tablas 153  
 utilización  
 actualización de filas 367  
 insertar filas 352  
 supresión de filas 370

vistas de prueba de tablas existentes 1015

visualización  
 columnas de tabla 373  
 privilegios de tabla 373

volver a enlazar  
 automáticamente  
 condiciones de 957  
 cambios que requieren 16  
 lista de planes y paquetes 952  
 listas de planes o paquetes 952, 953  
 paquetes con caracteres que coincidan con el patrón 948  
 planes 951  
 planificar 957

## W

WHENEVER, sentencia  
 C 603  
 Cláusula CONTINUAR 562  
 Cláusula SQLERROR 562  
 Cláusula SQLWARNING 562  
 COBOL 654  
 Códigos de error de SQL 562  
 ensamblador 582  
 especificar 562  
 Fortran 722  
 IR A la cláusula 562  
 NOT FOUND, cláusula 431, 562  
 PL/I 734

## X

XMLEXISTS  
 descripción 472  
 ejemplo 472

XMLQUERY  
 descripción 381  
 ejemplo 381

XPath  
 Contextos XPath 381

## Z

z/OS Debugger  
 función definida por el usuario 1033





Número de Programa: 5650-DB2  
5770-AF3

SC27-8845-02

