

Base de Datos Oracle 10g: Desarrollo de Unidades de Programa PL/SQL

Volumen 1: Guía del Alumno

D17169CS11

Edición 1.1

Agosto de 2004

D21976

ORACLE®

Autores

Glenn Stokol
Aniket Raut

Colaboradores y Revisores Técnicos

Andrew Brannigan
Dr. Christoph Burandt
Kathryn Cunningham
Marjolein Dekkers
Janis Fleishman
Nancy Greenberg
Stefan Grenstad
Elizabeth Hall
Rosita Hanoman
Craig Hollister
Taj-ul Islam
Eric Lee
Bryn Llewellyn
Werner Nowatzky
Nagavalli Pataballa
Sunitha Patel
Denis Raphaely
Helen Robertson
Grant Spencer
Tone Thomas
Priya Vennapusa
Ken Woolfe
Cesljas Zarco

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Esta documentación contiene información propiedad de Oracle Corporation; se suministra bajo los términos de un contrato de licencia que contiene restricciones de uso y de revelación y está también protegida por la legislación de derechos de autor. Queda prohibida la ingeniería reversa. Si esta documentación se entrega a una agencia del Ministerio de Defensa del Gobierno de EE.UU., se aplicará la siguiente advertencia de "Restricted Rights":

Restricted Rights Legend

Use, duplication or disclosure by the Government is subject to restrictions for commercial computer software and shall be deemed to be Restricted Rights software under Federal law, as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

Este material ni ninguna parte del mismo podrá ser reproducido en cualquier forma o a través de cualquier medio sin el expreso consentimiento por escrito de Oracle Corporation. La reproducción es una violación de la ley de derechos de autor y puede tener consecuencias penales o civiles.

Si esta documentación se entrega a una agencia del Gobierno de EE.UU. no perteneciente al Ministerio de Defensa, se aplicará la advertencia de "Restricted Rights" definida en FAR 52.227-14, Rights in Data-General, incluido Alternate III (junio de 1987).

La información contenida en este documento está sujeta a cambio sin previo aviso. Si detecta cualquier problema en la documentación, le agradeceremos lo comunique por escrito a Education Products, Oracle Corporation, 500 Oracle Parkway, Box SB-6, Redwood Shores, CA 94065. Oracle Corporation no garantiza que este documento esté exento de errores.

Todas las referencias a Oracle y a productos Oracle son marcas comerciales o marcas comerciales registradas de Oracle Corporation.

Todos los demás nombres de compañías y productos mencionados se utilizan a los exclusivos fines de su identificación y pueden ser marcas comerciales de sus respectivos propietarios.

Editor

Joseph Fernandez

Contenido

Prefacio

I Introducción

- Objetivos I-2
- Objetivos del Curso I-3
- Agenda I-4
- Esquema Human Resources (HR) I-7
- Creación de un Diseño de Subprograma Basado en Módulos y Capas I-9
- Desarrollo Basado en Módulos con Bloques PL/SQL I-10
- Revisión de Bloques Anónimos I-11
- Introducción a Procedimientos PL/SQL I-12
- Introducción a Funciones PL/SQL I-13
- Introducción a Paquetes PL/SQL I-14
- Introducción a Disparadores PL/SQL I-15
- Entorno de Ejecución de PL/SQL I-16
- Entornos de Desarrollo de PL/SQL I-17
- Codificación de PL/SQL en *iSQL*Plus* I-18
- Codificación de PL/SQL en SQL*Plus I-19
- Codificación de PL/SQL en Oracle JDeveloper I-20
- Resumen I-21
- Práctica I: Visión General I-22

1 Creación de Procedimientos Almacenados

- Objetivos 1-2
- ¿Qué es un Procedimiento? 1-3
- Sintaxis para Crear Procedimientos 1-4
- Desarrollo de Procedimientos 1-5
- ¿Qué son los Parámetros? 1-6
- Parámetros Formales y Reales 1-7
- Modos de Parámetros de Procedimiento 1-8
- Uso de Parámetros **IN**: Ejemplo 1-9
- Uso de Parámetros **OUT**: Ejemplo 1-10
- Visualización de Parámetros **OUT** con *iSQL*Plus* 1-11
- Llamada a PL/SQL con Variables de Host 1-12
- Uso de Parámetros **IN OUT**: Ejemplo 1-13
- Sintaxis de Transferencia de Parámetros 1-14
- Transferencia de Parámetros: Ejemplos 1-15
- Uso de la Opción **DEFAULT** para Parámetros 1-16
- Resumen de los Modos de Parámetros 1-18
- Llamada a los Procedimientos 1-19
- Excepciones Manejadas 1-20

Excepciones Manejadas: Ejemplo 1-21
Excepciones No Manejadas 1-22
Excepciones No Manejadas: Ejemplo 1-23
Eliminación de Procedimientos 1-24
Visualización de Procedimientos en el Diccionario de Datos 1-25
Ventajas de los Subprogramas 1-27
Resumen 1-28
Práctica 1: Visión General 1-30

2 Creación de Funciones Almacenadas

Objetivos 2-2
Visión General de las Funciones Almacenadas 2-3
Sintaxis para Crear Funciones 2-4
Desarrollo de Funciones 2-5
Función Almacenada: Ejemplo 2-6
Modos de Ejecutar Funciones 2-7
Ventajas del Uso de Funciones Definidas por el Usuario en Sentencias SQL 2-8
Funciones en Expresiones SQL: Ejemplo 2-9
Ubicaciones de Funciones Definidas por el Usuario...2-10
Restricciones para Llamar a Funciones desde Expresiones SQL 2-11
Control de Efectos Secundarios al Llamar a Funciones desde Expresiones SQL 2-12
Restricciones para Llamar a Funciones desde SQL: Ejemplo 2-13
Eliminación de Funciones 2-14
Visualización de Funciones en el Diccionario de Datos 2-15
Procedimientos frente a Funciones 2-16
Resumen 2-17
Práctica 2: Visión General 2-18

3 Creación de Paquetes

Objetivos 3-2
Paquetes PL/SQL: Visión General 3-3
Componentes de un Paquete PL/SQL 3-4
Visibilidad de Componentes de Paquete 3-6
Desarrollo de Paquetes PL/SQL 3-7
Creación de la Especificación del Paquete 3-8
Ejemplo de Especificación de un Paquete: `comm_pkg` 3-9
Creación del Cuerpo del Paquete 3-10
Ejemplo del Cuerpo del Paquete: `comm_pkg` 3-11
Llamada a Subprogramas de Paquete 3-12
Creación y Uso de Paquetes sin Cuerpo 3-13
Eliminación de Paquetes 3-14
Visualización de Paquetes en el Diccionario de Datos 3-15
Instrucciones para la Escritura de Paquetes 3-16
Ventajas del Uso de Paquetes 3-17
Resumen 3-19
Práctica 3: Visión General 3-21

4 Uso de Más Conceptos de Paquete

- Objetivos 4-2
- Sobrecarga de Subprogramas 4-3
- Sobrecarga: Ejemplo 4-5
- Sobrecarga y el Paquete STANDARD 4-7
- Uso de Declaraciones Anticipadas 4-8
- Bloque de Inicialización de Paquetes 4-10
- Uso de Funciones de Paquete en SQL y Restricciones 4-11
- Función de Paquete en SQL: Ejemplo 4-12
- Estado Persistente de Paquetes 4-13
- Estado Persistente de las Variables de Paquetes: Ejemplo 4-14
- Estado Persistente de un Cursor de Paquete 4-15
- Ejecución de CURS_PKG 4-16
- Uso de Tablas PL/SQL de Registros en Paquetes 4-17
- Wrapper PL/SQL 4-18
- Ejecución de Wrapper 4-19
- Resultados del Ajuste 4-20
- Instrucciones para el Ajuste 4-21
- Resumen 4-22
- Práctica 4: Visión General 4-23

5 Uso de Paquetes Proporcionados por Oracle en el Desarrollo de Aplicaciones

- Objetivos 5-2
- Uso de Paquetes Proporcionados por Oracle 5-3
- Lista de Algunos Paquetes Proporcionados por Oracle 5-4
- Funcionamiento del Paquete DBMS_OUTPUT 5-5
- Interacción con los Archivos del Sistema Operativo 5-7
- Procesamiento de Archivos con el Paquete UTL_FILE 5-8
- Excepciones en el Paquete UTL_FILE 5-10
- Parámetros de Función FOPEN e IS_OPEN 5-11
- Uso de UTL_FILE: Ejemplo 5-12
- Generación de Páginas Web con el Paquete HTP 5-14
- Uso de los Procedimientos de Paquete HTP 5-15
- Creación de un Archivo HTML con iSQL*Plus 5-16
- Uso de UTL_MAIL 5-17
- Instalación y Uso de UTL_MAIL 5-18
- Envío de Correo Electrónico con Anexos Binarios 5-19
- Envío de Correo Electrónico con Anexos de Texto 5-20
- Paquete de DBMS_SCHEDULER 5-23
- Creación de un Trabajo 5-25
- Creación de un Trabajo con Parámetros en Línea 5-26
- Creación de un Trabajo Utilizando un Programa 5-27
- Creación de un Trabajo para un Programa con Argumentos 5-28
- Creación de un Trabajo Utilizando una Planificación 5-29
- Definición del Intervalo de Repetición para un Trabajo 5-30

Creación de un Trabajo Utilizando un Programa y una Planificación con Nombre 5-31
Gestión de Trabajos 5-32
Vistas de Diccionario de Datos 5-33
Resumen 5-34
Práctica 5: Visión General 5-35

6 SQL Dinámico y Metadatos

Objetivos 6-2
Flujo de Ejecución de SQL 6-3
SQL Dinámico 6-4
SQL Dinámico Nativo 6-5
Uso de la Sentencia EXECUTE IMMEDIATE 6-6
SQL Dinámico con una Sentencia DDL 6-7
SQL Dinámico con Sentencias DML 6-8
SQL Dinámico con una Consulta de una Sola Fila 6-9
SQL Dinámico con una Consulta de Varias Filas 6-10
Declaración de Variables de Cursor 6-11
Ejecución Dinámica de un Bloque PL/SQL 6-12
Uso de SQL Dinámico Nativo para Compilar Código PL/SQL 6-13
Uso del Paquete DBMS_SQL 6-14
Uso de DBMS_SQL con una Sentencia DML 6-15
Uso de DBMS_SQL con una Sentencia DML con Parámetros 6-16
Comparación de SQL Dinámico Nativo y el Paquete DBMS_SQL 6-17
Paquete DBMS_METADATA 6-18
API de Metadatos 6-19
Subprogramas en DBMS_METADATA 6-20
Subprogramas FETCH_xxx 6-21
Procedimiento SET_FILTER 6-22
Filtros 6-23
Ejemplos de Definición de Filtros 6-24
Uso Programático: Ejemplo 1 6-25
Uso Programático: Ejemplo 2 6-27
API de Exploración 6-29
API de Exploración: Ejemplos 6-30
Resumen 6-32
Práctica 6: Visión General 6-33

7 Consideraciones de Diseño para Código PL/SQL

Objetivos 7-2
Estandarización de Constantes y Excepciones 7-3
Estandarización de Excepciones 7-4
Estandarización del Manejo de Excepciones 7-5
Estandarización de Constantes 7-6
Subprogramas Locales 7-7
Derechos del Responsable de la Definición frente a Derechos del Invocador 7-8
Especificación de Derechos del Invocador 7-9

Transacciones Autónomas	7-10
Funciones de las Transacciones Autónomas	7-11
Uso de Transacciones Autónomas	7-12
Cláusula RETURNING	7-13
Enlace en Bloque	7-14
Uso del Enlace en Bloque	7-15
Enlace en Bloque FORALL: Ejemplo	7-16
Uso de BULK COLLECT INTO con Consultas	7-18
Uso de BULK COLLECT INTO con Cursos	7-19
Uso de BULK COLLECT INTO con una Cláusula RETURNING	7-20
Uso de la Indicación NOCOPY	7-21
Efectos de la Indicación NOCOPY	7-22
La Indicación NOCOPY Se Puede Ignorar	7-23
Indicación PARALLEL_ENABLE	7-24
Resumen	7-25
Práctica 7: Visión General	7-26

8 Gestión de Dependencias

Objetivos	8-2
Descripción de las Dependencias	8-3
Dependencias	8-4
Dependencias Locales	8-5
Supuesto de Dependencias Locales	8-7
Visualización de Dependencias Directas mediante USER_DEPENDENCIES	8-8
Visualización de Dependencias Directas e Indirectas	8-9
Visualización de Dependencias	8-10
Otro Supuesto de Dependencias Locales	8-11
Supuesto de Dependencias Locales de Nomenclatura	8-12
Descripción de las Dependencias Remotas	8-13
Conceptos de Dependencias Remotas	8-15
Parámetro REMOTE_DEPENDENCIES_MODE	8-16
Dependencias Remotas y Modo de Registro de Hora	8-17
El Procedimiento Remoto B se Compila a las 8:00 a.m.	8-19
El Procedimiento Local A se Compila a las 9:00 a.m.	8-20
Ejecución del Procedimiento A	8-21
Procedimiento Remoto B Recompilado a las 11:00 a.m.	8-22
Ejecución del Procedimiento A	8-23
Modo de Firma	8-24
Recompilación de una Unidad de Programa PL/SQL	8-25
Recompilación Incorrecta	8-26
Recompilación Correcta	8-27
Recompilación de Procedimientos	8-28
Paquetes y Dependencias	8-29
Resumen	8-31
Práctica 8: Visión General	8-32

9 Manipulación de Objetos Grandes

- Objetivos 9-2
- ¿Qué es un LOB? 9-3
- Comparación de los Tipos de Dato LONG y LOB 9-5
- Anatomía de un LOB 9-6
- LOB Internos 9-7
- Gestión de los LOB Internos 9-8
- ¿Qué son los BFILE? 9-9
- Protección de BFILE 9-10
- Nuevo Objeto de Base de Datos: DIRECTORY 9-11
- Instrucciones para la Creación de Objetos DIRECTORY 9-12
- Gestión de los BFILE 9-13
- Preparación para Utilizar BFILE 9-15
- Relleno de Columnas BFILE con SQL 9-16
- Relleno de Columnas BFILE con PL/SQL 9-17
- Uso de Rutinas DBMS_LOB con BFILES 9-18
- Migración de LONG a LOB 9-19
- Paquete DBMS_LOB 9-21
- DBMS_LOB.READ y DBMS_LOB.WRITE 9-24
- Inicialización de Columnas LOB Agregadas a una Tabla 9-25
- Relleno de Columnas LOB 9-26
- Actualización de LOB con DBMS_LOB en PL/SQL 9-27
- Selección de Valores CLOB con SQL 9-28
- Selección de Valores CLOB con DBMS_LOB 9-29
- Selección de Valores CLOB en PL/SQL 9-30
- Eliminación de LOB 9-31
- LOB Temporales 9-32
- Creación de un LOB Temporal 9-33
- Resumen 9-34
- Práctica 9: Visión General 9-35

10 Creación de Disparadores

- Objetivos 10-2
- Tipos de Disparadores 10-3
- Instrucciones para el Diseño de Disparadores 10-5
- Creación de Disparadores DML 10-7
- Tipos de Disparadores DML 10-8
- Temporización de Disparadores 10-9
- Secuencia de Arranque de Disparadores 10-10
- Tipos de Evento y Cuerpo del Disparador 10-12
- Creación de un Disparador de Sentencia DML 10-13
- Prueba de SECURE_EMP 10-14
- Uso de Predicados Condicionales 10-15
- Creación de un Disparador de Fila DML 10-16
- Uso de los Cualificadores OLD y NEW 10-17

Uso de los Cualificadores OLD y NEW: Ejemplo con audit_emp	10-16
Restricción de un Disparador de Fila: Ejemplo	10-17
Resumen del Modelo de Ejecución de Disparadores	10-18
Implementación de una Restricción de Integridad con un Disparador	10-19
Disparadores INSTEAD OF	10-20
Creación de un Disparador INSTEAD OF	10-21
Comparación de Disparadores de Base de Datos y Procedimientos Almacenados	10-24
Comparación de Disparadores de Base de Datos y Disparadores de Oracle Forms	10-25
Gestión de Disparadores	10-26
Eliminación de Disparadores	10-27
Prueba de Disparadores	10-28
Resumen	10-29
Práctica 10: Visión General	10-30

11 Aplicaciones para Disparadores

Objetivos	11-2
Creación de Disparadores de Base de Datos	11-3
Creación de Disparadores en Sentencias DDL	11-4
Creación de Disparadores en Eventos de Sistema	11-5
Disparadores LOGON y LOGOFF: Ejemplo	11-6
Sentencias CALL	11-7
Lectura de Datos en una Tabla Mutante	11-8
Tabla Mutante: Ejemplo	11-9
Ventajas de los Disparadores de Base de Datos	11-11
Gestión de Disparadores	11-12
Supuestos de Aplicación de Negocio para la Implementación de Disparadores	11-13
Visualización de Información de Disparador	11-14
Uso de USER_TRIGGERS	11-15
Lista de Códigos de Disparadores	11-16
Resumen	11-17
Práctica 11: Visión General	11-18

12 Descripción e Influencia del Compilador PL/SQL

Objetivos	12-2
Compilación Nativa e Interpretada	12-3
Funciones y Ventajas de la Compilación Nativa	12-4
Consideraciones Cuando se Utiliza la Compilación Nativa	12-5
Parámetros que Influyen en la Compilación	12-6
Cambio entre Compilación Nativa e Interpretada	12-7
Visualización de Información de Compilación en el Diccionario de Datos	12-8
Uso de la Compilación Nativa	12-9
Infraestructura de Advertencias del Compilador	12-10
Definición de Niveles de Advertencia del Compilador	12-11
Instrucciones para el Uso de PLSQL_WARNINGS	12-12
Paquete DBMS_WARNING	12-13

Uso de Procedimientos DBMS_WARNING 12-14
Uso de Funciones DBMS_WARNING 12-15
Uso de DBMS_WARNING: Ejemplo 12-16
Resumen 12-18
Práctica 12: Visión General 12-19

Apéndice A: Soluciones a la Práctica

Apéndice B: Descripciones de las Tablas y Datos

Apéndice C: Estudios para Implementación de Disparadores

Objetivos C-2
Control de la Seguridad en el Servidor C-3
Control de la Seguridad con un Disparador de Base de Datos C-4
Uso de la Utilidad del Servidor para Auditoría de Operaciones de Datos C-5
Auditoría con un Disparador C-6
Auditoría de Disparadores con Construcciones de Paquetes C-7
Paquete AUDIT_PKG C-9
Tabla AUDIT_TABLE y Procedimiento AUDIT_EMP C-10
Forzado de Integridad de Datos en el Servidor C-11
Protección de la Integridad de los Datos con un Disparador C-12
Forzado de la Integridad Referencial en el Servidor C-13
Protección de la Integridad Referencial con un Disparador C-14
Replicación de Tablas en el Servidor C-15
Replicación de Tablas con un Disparador C-16
Cálculo de Datos Derivados en el Servidor C-17
Cálculo de Valores Derivados con un Disparador C-18
Registro de Eventos con un Disparador C-19
Resumen C-21

Apéndice D: Revisión de PL/SQL

Estructura en Bloque para Bloques PL/SQL Anónimos D-2
Declaración de Variables PL/SQL D-3
Declaración de Variables con el Atributo %TYPE D-4
Creación de un Registro PL/SQL D-5
Atributo %ROWTYPE D-6
Creación de una Tabla PL/SQL D-7
Sentencias SELECT en PL/SQL D-8
Inserción de Datos D-9
Actualización de Datos D-10
Supresión de Datos D-11
Sentencias COMMIT y ROLLBACK D-12
Atributos de Cursor SQL D-13
Sentencias IF, THEN y ELSIF D-14
Bucle Básico D-18
Bucle FOR D-16
Bucle WHILE D-17

Control de Cursores Explícitos con Cuatro Comandos	D-18
Declaración del Cursor	D-19
Apertura del Cursor	D-20
Recuperación de Datos del Cursor	D-21
Cierre del Cursor	D-22
Atributos de Cursor Explícito	D-23
Bucles FOR de Cursor	D-24
Cláusula FOR UPDATE	D-25
Cláusula WHERE CURRENT OF	D-26
Detección de Errores Predefinidos del Servidor de Oracle	D-27
Detección de Errores Predefinidos del Servidor de Oracle: Ejemplo	D-28
Error No Predefinido	D-29
Excepciones Definidas por el Usuario	D-30
Procedimiento RAISE_APPLICATION_ERROR	D-31

Apéndice E: JDeveloper

JDeveloper	E-2
Navegador de Conexiones	E-3
Navegador de Aplicaciones	E-4
Ventana Structure	E-5
Ventana Editor	E-6
Despliegue de Procedimientos Almacenados en Java	E-7
Publicación de Java en PL/SQL	E-8
Creación de Unidades de Programa	E-9
Compilación	E-10
Ejecución de una Unidad de Programa	E-11
Borrado de una Unidad de Programa	E-12
Depuración de Programas PL/SQL	E-13
Definición de Puntos de Ruptura	E-16
Análisis de Código	E-17
Examen y Modificación de Variables	E-18

Índice

Prácticas Adicionales

Prácticas Adicionales: Soluciones

Prácticas Adicionales: Descripciones de las Tablas y Datos

Oracle Internal & OAI Use Only

Prefacio

Oracle Internal & OAI Use Only

Oracle Internal & OAI Use Only

Perfil

Cualificación Necesaria para el Curso

Antes de empezar este curso, debe tener un conocimiento exhaustivo de SQL y *iSQL*Plus*, así como experiencia laboral en el desarrollo de aplicaciones. Los requisitos son cualquiera de los siguientes cursos o combinaciones de cursos de Oracle University:

- *Base de Datos Oracle 10g: Introducción a SQL*
- *Base de Datos Oracle 10g: Conceptos Fundamentales de SQL I* y *Base de Datos Oracle 10g: Conceptos Fundamentales de SQL II*
- *Base de Datos Oracle 10g: Conceptos Básicos de SQL y PL/SQL*
- *Base de Datos Oracle 10g: Conceptos Básicos de PL/SQL*

Organización del Curso

Base de Datos Oracle 10g: Desarrollo de Unidades de Programa PL/SQL es un curso dirigido por un instructor que incluye teoría y ejercicios prácticos. Las demostraciones en línea y las sesiones de prácticas sirven para reforzar los conceptos y las habilidades presentados.

Publicaciones Relacionadas

Publicaciones de Oracle

Título	Número de Artículo
<i>Oracle Database Application Developer's Guide – Fundamentals (10g Release 1)</i>	B10795-01
<i>Oracle Database Application Developer's Guide – Large Objects (10g Release 1)</i>	B10796-01
<i>PL/SQL Packages and Types Reference (10g Release 1)</i>	B10802-01
<i>PL/SQL User's Guide and Reference (10g Release 1)</i>	B10807-01

Publicaciones Adicionales

- Boletines de las versiones del sistema
- Guías de instalación y del usuario
- Archivos Léame
- Artículos del grupo internacional de usuarios de Oracle (International Oracle User's Group, IOUG)
- *Oracle Magazine*

Convenciones Tipográficas

Convenciones Tipográficas en el Texto

Convención	Elemento	Ejemplo
Negrita	Palabras y frases resaltadas en contenido Web sólo	Para navegar dentro de esta aplicación, no haga clic en los botones Atrás y Adelante.
Negrita y cursiva	Términos del glosario (si existe uno)	El algoritmo inserta la nueva clave.
Corchetes	Nombres de teclas	Pulse [Intro].
Mayúsculas y minúsculas	Botones, casillas de control, disparadores, ventanas	Haga clic en el botón Executable. Active la casilla de control Registration Required. Asigne un disparador When-Validate-Item. Abra la ventana Master Schedule.
Signo mayor que	Rutas de acceso de menús	Seleccione File > Save.
Comas	Secuencias de teclas	Pulse y suelte estas teclas de una en una: [Alt], [F], [D]

Convenciones Tipográficas (continuación)

Convenciones Tipográficas en el Texto (continuación)

Convención	Objeto o Término	Ejemplo
Courier New, sin distinción entre mayúsculas y minúsculas	Resultado de código, elementos de código SQL y PL/SQL, elementos de código Java, nombres de directorios, nombres de archivos, contraseñas, nombres de rutas de acceso, direcciones URL, entradas del usuario, nombres de usuario	Resultado de código: <code>debug.seti('I', 300);</code> Elementos de código SQL: Utilice el comando <code>SELECT</code> para ver la información almacenada en la columna <code>last_name</code> de la tabla <code>emp</code> . Elementos de código Java: La programación de Java implica las clases <code>String</code> y <code>StringBuffer</code> . Nombres de directorios: <code>bin</code> (DOS), <code>\$FMHOME</code> (UNIX) Nombres de archivos: Localice el archivo <code>init.ora</code> . Contraseñas: Utilice <code>tiger</code> como contraseña. Nombres de rutas de acceso: Abra <code>c:\my_docs\projects</code> . Direcciones URL: Vaya a <code>http://www.oracle.com</code> . Entradas del usuario: Escriba <code>300</code> . Nombres de usuario: Conéctese como <code>scott</code> .
Mayúscula inicial	Etiquetas de los gráficos (a menos que el término sea un nombre propio)	Dirección del cliente (<i>salvo Oracle Payables</i>)
Cursiva	Palabras y frases resaltadas en publicaciones impresas, títulos de libros y cursos, variables	<i>No</i> guarde los cambios en la base de datos. Para obtener más información, consulte <i>Oracle7 Server SQL Language Reference Manual</i> . Escriba <code>user_id@us.oracle.com</code> , donde <code>user_id</code> es el nombre del usuario.
Signo más	Combinaciones de teclas	Pulse estas teclas de forma simultánea: <code>[Ctrl] + [Alt] + [Supr]</code>
Comillas	Títulos de lecciones y capítulos en referencias cruzadas, elementos de la interfaz con nombres extensos que sólo llevan mayúscula inicial	Este tema se trata en la Unidad II, Lección 3, “Trabajar con Objetos”. Haga clic en “Include a reusable module component” y, luego, en <code>Finish</code> . Utilice la propiedad “WHERE clause of query”.

Convenciones Tipográficas (continuación)

Convenciones Tipográficas en las Rutas de Acceso de Navegación

En este curso se utilizan rutas de acceso de navegación simplificadas para guiarle a través de Aplicaciones Oracle, como en el ejemplo siguiente.

Invoice Batch Summary

(N) Invoice > Entry > Invoice Batches Summary (M) Query > Find (B) Approve

Esta ruta de acceso simplificada se traduce en la siguiente secuencia de pasos:

1. (N) En la ventana del navegador, seleccione Invoice > Entry > Invoice Batches Summary.
2. (M) En el menú, seleccione Query > Find.
3. (B) Haga clic en el botón Approve.

Notación:

(N) = Navegador	(I) = Icono
(M) = Menú	(H) = Enlace de hipertexto
(T) = Separador	(B) = Botón

Oracle Internal & OAI Use Only

I

Introducción

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- **Describir los objetivos del curso**
- **Identificar los componentes modulares de PL/SQL:**
 - **Bloques anónimos**
 - **Procedimientos y funciones**
 - **Paquetes**
- **Describir el entorno de ejecución de PL/SQL**
- **Describir las tablas y el esquema de base de datos que se utilizan en el curso**
- **Enumerar los entornos de desarrollo PL/SQL que están disponibles en el curso**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetivos

PL/SQL soporta muchas construcciones de programa. En esta lección, se revisan las unidades de programa como bloques anónimos y se presentan los bloques PL/SQL con nombre. Los bloques PL/SQL con nombre también se denominan subprogramas. Los bloques PL/SQL con nombre incluyen procedimientos y funciones.

Las tablas del esquema Human Resources (HR) (que se utiliza para las prácticas de este curso) se describen brevemente. Además, se enumeran las herramientas de desarrollo para escribir, probar y depurar PL/SQL.

Objetivos del Curso

Al finalizar este curso, debería estar capacitado para lo siguiente:

- **Crear, ejecutar y mantener:**
 - Procedimientos y funciones con parámetros OUT
 - Construcciones de paquetes
 - Disparadores de base de datos
- **Gestionar disparadores y subprogramas PL/SQL**
- **Utilizar un subjuego de paquetes proporcionados por Oracle para:**
 - Generar salidas de pantalla, archivo y Web
 - Planificar trabajos PL/SQL para que se ejecuten independientemente
- **Crear y ejecutar sentencias SQL dinámicas**
- **Manipular grandes objetos (LOB)**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetivos del Curso

Puede desarrollar aplicaciones basadas en módulos con procedimientos de base de datos utilizando objetos de base de datos como los siguientes:

- Procedimientos y funciones
- Paquetes
- Disparadores de base de datos

Las aplicaciones modulares mejoran los siguientes aspectos:

- Funcionalidad
- Seguridad
- Rendimiento global

Agenda

Lecciones que se deben tratar el primer día:

Introducción

- 1. Creación de Procedimientos Almacenados**
- 2. Creación de Funciones Almacenadas**
- 3. Creación de Paquetes**
- 4. Uso de Más Conceptos de Paquete**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Agenda

Lecciones que se deben tratar el segundo día:

- 5. Uso de Paquetes Proporcionados por Oracle en el Desarrollo de Aplicaciones**
- 6. SQL Dinámico y Metadatos**
- 7. Consideraciones de Diseño para Código PL/SQL**
- 8. Gestión de Dependencias**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Agenda

Lecciones que se deben tratar el tercer día:

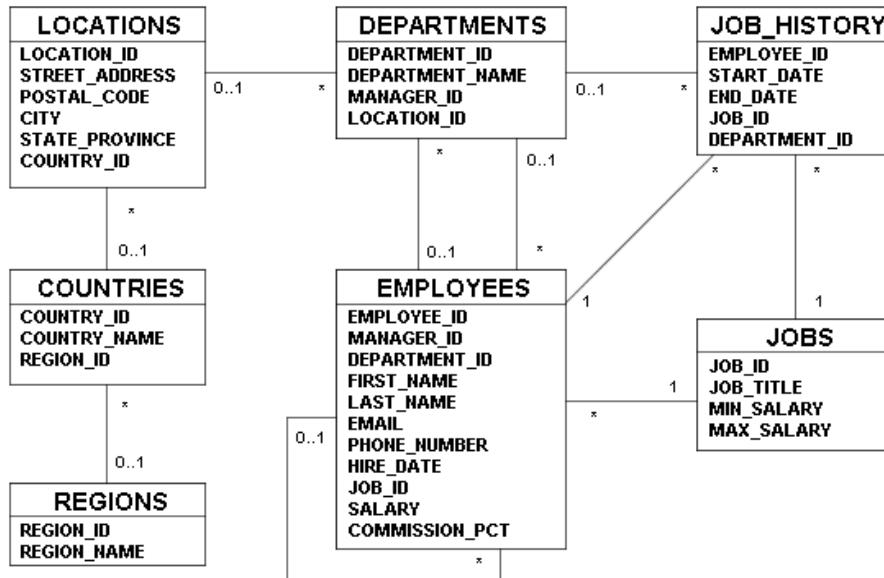
- 9. Manipulación de Objetos Grandes**
- 10. Creación de Disparadores**
- 11. Aplicaciones para Disparadores**
- 12. Descripción e Influencia del Compilador PL/SQL**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Esquema Human Resources (HR)



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

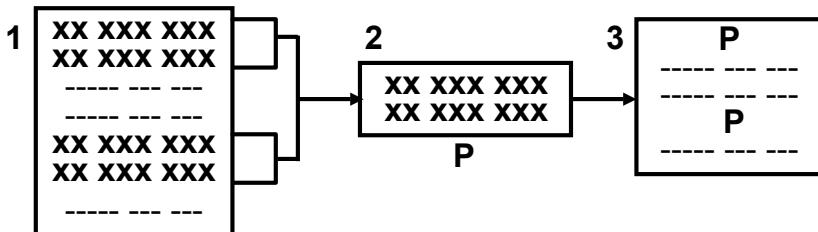
Esquema Human Resources (HR)

El esquema Human Resources (HR) forma parte de los esquemas de ejemplo de Oracle que se pueden instalar en una base de datos Oracle. Las sesiones prácticas de este curso utilizan datos del esquema HR.

Descripciones de las Tablas

- REGIONS contiene filas que representan una región, como América, Asia, etc.
- COUNTRIES contiene filas para países, que están asociados a una región.
- LOCATIONS contiene la dirección concreta de una oficina, almacén o fábrica de una compañía en un país determinado.
- DEPARTMENTS muestra detalles de los departamentos en los que trabajan los empleados. Cada departamento puede tener una relación que represente al director del departamento en la tabla EMPLOYEES.
- EMPLOYEES contiene detalles sobre cada empleado que trabaja en un departamento. Puede que algunos empleados no estén asignados a ningún departamento.
- JOBS contiene los tipos de trabajos que puede tener cada empleado.
- JOB_HISTORY contiene el historial del trabajo de los empleados. Si un empleado cambia de departamento dentro de un mismo trabajo o cambia de trabajo dentro de un mismo departamento, se insertará una nueva fila en esta tabla con la información del antiguo trabajo del empleado.

Creación de un Diseño de Subprograma Basado en Módulos y Capas



- **Crear código basado en módulos en subprogramas.**
 1. Localizar secuencias de códigos que se repiten más de una vez.
 2. Crear el subprograma P que contiene el código repetido.
 3. Modificar el código original para llamar al nuevo subprograma.
- **Crear capas de subprograma para la aplicación.**
 - Capa de subprograma de acceso a datos con SQL lógico
 - Capa de subprograma de lógica de negocio, que puede o no utilizar la capa de acceso a datos

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un Diseño de Subprograma Basado en Módulos y Capas

El diagrama ilustra el principio de creación de subprogramas basados en módulos: la creación de partes más pequeñas y gestionables de código flexible y reutilizable. La flexibilidad se consigue con el uso de subprogramas con parámetros, lo que a su vez hace que el mismo código se pueda reutilizar para distintos valores de entrada. Para basar en módulos un código existente, realice los siguientes pasos:

1. Localice e identifique secuencias repetitivas de código.
2. Mueva el código repetitivo a un subprograma PL/SQL.
3. Sustituya el código repetitivo original por llamadas al nuevo subprograma PL/SQL.

Capas de Subprograma

Ya que PL/SQL permite que las sentencias SQL se embeban en la lógica de forma ininterrumpida, resulta muy fácil tener sentencias SQL repartidas por todo el código. Sin embargo, se recomienda separar la lógica SQL de la lógica de negocio, es decir, crear un diseño de aplicación basada en capas con al menos dos capas:

- Capa de acceso a datos: Para que las subrutinas accedan a los datos mediante sentencias SQL.
- Capa de lógica de negocio: Para que los subprogramas implementen las reglas de procesamiento de negocio, que se pueden llamar en rutinas de capa de acceso a datos.

Creación de un Diseño de Subprograma Basado en Módulos y Capas (Continuación)

Si sigue este enfoque basado en módulos y capas, podrá crear código más fácil de mantener, especialmente cuando cambian las reglas de negocio. Además, al mantener la lógica SQL simple y libre de lógica de negocio compleja, el usuario se beneficiará del optimizador de base de datos Oracle, que puede reutilizar sentencias SQL analizadas para un mejor uso de los recursos del servidor.

Oracle Internal & OAI Use Only

Desarrollo Basado en Módulos con Bloques PL/SQL

- **PL/SQL es un lenguaje estructurado en bloques. Los bloques de código PL/SQL le ayudan a crear código basado en módulos mediante:**
 - Bloques anónimos
 - Procedimientos y funciones
 - Paquetes
 - Disparadores de base de datos
- **Las ventajas del uso de construcciones de programas modulares son:**
 - Mantenimiento sencillo
 - Integridad y seguridad de datos mejoradas
 - Rendimiento mejorado
 - Claridad de código mejorada

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Desarrollo Basado en Módulos con Bloques PL/SQL

Un subprograma se basa en estructuras PL/SQL estándar. Contiene una sección de declaraciones, una sección ejecutable y una sección opcional de manejo de excepciones (por ejemplo, bloques anónimos, procedimientos, funciones, paquetes y disparadores). Los subprogramas se pueden compilar y almacenar en la base de datos, lo que proporciona capacidades de organización en módulos, extensibilidad, reutilización y mantenimiento.

La organización en módulos convierte grandes bloques de código en grupos de código más pequeños denominados módulos. Después de este proceso, los módulos se pueden reutilizar en el mismo programa o compartir con otros programas. Es más fácil mantener y depurar código compuesto de pequeños módulos que mantener código de un único programa grande. Los módulos se pueden ampliar fácilmente para su personalización mediante la incorporación de más funcionalidad, si es necesario, sin que ello afecte al resto de los módulos del programa.

Los subprogramas proporcionan un mantenimiento sencillo, ya que el código se ubica en un sólo lugar y por lo tanto cualquier modificación necesaria para el subprograma se puede realizar únicamente en este lugar. Los subprogramas proporcionan seguridad e integridad mejoradas de los datos. Se accede a los objetos de datos a través del subprograma y un usuario puede llamar al subprograma sólo si se le otorga el privilegio de acceso adecuado.

Nota: El conocimiento de cómo desarrollar bloques anónimos es un requisito de este curso. Para obtener más información sobre bloques anónimos, consulte el curso *Oracle 10g: Conceptos Básicos de PL/SQL*.

Revisión de Bloques Anónimos

Bloques anónimos:

- **Forman la estructura de bloque PL/SQL básica**
- **Inician las tareas de procesamiento de PL/SQL desde las aplicaciones**
- **Se pueden anidar en la sección ejecutable de cualquier bloque PL/SQL**

```
[DECLARE      -- Declaration Section (Optional)
     variable declarations; ... ]
BEGIN         -- Executable Section (Mandatory)
     SQL or PL/SQL statements;
[EXCEPTION    -- Exception Section (Optional)
     WHEN exception THEN statements; ]
END;          -- End of Block (Mandatory)
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Revisión de Bloques Anónimos

Los bloques anónimos se utilizan normalmente para:

- Escribir código del disparador para componentes de Oracle Forms
- Iniciar llamadas a procedimientos, funciones y construcciones de paquetes
- Aislarse el manejo de excepciones dentro de un bloque de código
- Anidarse dentro de otros bloques PL/SQL para gestionar el control del flujo de código

La palabra clave DECLARE es opcional, pero se necesita si el usuario declara variables, constantes y excepciones que se van a utilizar dentro del bloque PL/SQL.

BEGIN y END son obligatorias y necesitan al menos una sentencia entre ellas, ya sea SQL, PL/SQL o ambas.

La sección de excepciones es opcional y se utiliza para manejar errores que se producen en el ámbito del bloque PL/SQL. Las excepciones se pueden propagar al emisor de la llamada del bloque anónimo excluyendo un manejador de excepciones de la excepción en particular y creando lo que se conoce como una excepción no tratada.

Introducción a Procedimientos PL/SQL

Los procedimientos son bloques PL/SQL con nombre que realizan una secuencia de acciones.

```
CREATE PROCEDURE getemp IS -- header
    emp_id    employees.employee_id%type;
    lname     employees.last_name%type;
BEGIN
    emp_id := 100;
    SELECT last_name INTO lname
    FROM EMPLOYEES
    WHERE employee_id = emp_id;
    DBMS_OUTPUT.PUT_LINE('Last name: ' || lname);
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Introducción a Procedimientos PL/SQL

Un procedimiento es un bloque PL/SQL con nombre que se crea con una sentencia CREATE PROCEDURE. Cuando se llama, el procedimiento realiza una secuencia de acciones. La anatomía del procedimiento incluye:

- Cabecera: “PROCEDURE getemp IS”
- Sección de declaraciones: Con las declaraciones de variables emp_id y lname.
- Sección ejecutable: Contiene las sentencias PL/SQL y SQL utilizadas para obtener el apellido del empleado 100. La sección ejecutable utiliza el paquete DBMS_OUTPUT para imprimir el apellido del empleado.
- Sección de excepciones: Opcional. No se utiliza en el ejemplo.

Nota: La codificación del valor 100 para emp_id no es flexible. El procedimiento sería más reutilizable si se utilizara como parámetro para obtener el valor de identificador del empleado. El uso de parámetros se aborda en la lección titulada “Creación de Procedimientos Almacenados”.

Para llamar a un procedimiento a través de un bloque anónimo, utilice:

```
BEGIN
    getemp;
END;
```

Introducción a Funciones PL/SQL

Las funciones son bloques PL/SQL con nombre que realizan una secuencia de acciones y devuelven un valor. Se puede llamar a una función desde:

- **Cualquier bloque PL/SQL**
- **Una sentencia SQL (opción sujeta a ciertas restricciones)**

```
CREATE FUNCTION avg_salary RETURN NUMBER IS
    avg_sal employees.salary%type;
BEGIN
    SELECT AVG(salary) INTO avg_sal
    FROM EMPLOYEES;
    RETURN avg_sal;
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Introducción a Funciones PL/SQL

Una función es un bloque PL/SQL con nombre que se crea con una sentencia CREATE FUNCTION. Las funciones se utilizan para calcular un valor que se debe devolver al emisor de la llamada.

Las funciones PL/SQL siguen la misma estructura de bloque que los procedimientos. Sin embargo, la cabecera empieza por la palabra clave FUNCTION seguida del nombre de la función. La cabecera incluye el tipo de dato de retorno después del nombre de la función (con la palabra clave RETURN seguida del tipo de dato). El ejemplo declara una variable denominada v_total_salary en la sección de declaraciones y utiliza la sentencia RETURN para devolver el valor recuperado de la sentencia SELECT. El valor devuelto representa el salario total de todos los empleados.

Se puede llamar a una función desde:

- Otro bloque PL/SQL donde el valor de retorno se puede almacenar en una variable o proporcionar como parámetro para un procedimiento.
- Una sentencia SQL, con sujeción a restricciones. (Este tema se aborda en la lección titulada “Creación de Funciones Almacenadas”.)

Para llamar a una función desde un bloque anónimo, utilice lo siguiente:

```
BEGIN
    dbms_output.put_line('Average Salary: ' ||
    avg_salary);
END;
```

Introducción a Paquetes PL/SQL

Los paquetes PL/SQL tienen una especificación y un cuerpo opcional. Los paquetes agrupan subprogramas relacionados.

```
CREATE PACKAGE emp_pkg IS
  PROCEDURE getemp;
  FUNCTION avg_salary RETURN NUMBER;
END emp_pkg;
/
CREATE PACKAGE BODY emp_pkg IS
  PROCEDURE getemp IS ...
  BEGIN ... END;

  FUNCTION avg_salary RETURN NUMBER IS ...
  BEGIN ... RETURN avg_sal; END;
END emp_pkg;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Introducción a Paquetes PL/SQL

Un paquete PL/SQL está compuesto normalmente de dos partes:

- Una especificación del paquete que declara los componentes públicos (accesibles) del paquete.
- Un cuerpo del paquete que implementa los componentes públicos y privados del paquete.

Un paquete se utiliza para agrupar construcciones y código PL/SQL relacionados. Esto ayuda a los desarrolladores a gestionar y mantener el código relacionado en un solo lugar. Los paquetes son construcciones potentes y proporcionan una forma de organizar código en módulos de forma lógica dentro de grupos funcionales o específicos de aplicaciones.

El ejemplo declara un paquete denominado `emp_package` que comprende un procedimiento `getemp` y una función `total_salary`. La especificación define la cabecera de la función y procedimiento. El cuerpo del paquete proporciona la implementación completa del procedimiento y la función declarados en la especificación del paquete. El ejemplo está incompleto pero proporciona una introducción al concepto de paquete PL/SQL. Los detalles de los paquetes PL/SQL se incluyen en la lección titulada “Creación de Paquetes”.

Para llamar al procedimiento del paquete desde un bloque anónimo, utilice lo siguiente:

```
BEGIN
  emp_pkg.getemp;
END;
```

Introducción a Disparadores PL/SQL

Los disparadores PL/SQL son bloques de código que se ejecutan cuando se produce un evento de tabla, base de datos o aplicación concreto.

- **Los disparadores de la aplicación Oracle Forms son bloques anónimos estándar.**
- **Los disparadores de la base de datos Oracle tienen una estructura concreta.**

```
CREATE TRIGGER check_salary
BEFORE INSERT OR UPDATE ON employees
FOR EACH ROW
DECLARE
    c_min constant number(8,2) := 1000.0;
    c_max constant number(8,2) := 500000.0;
BEGIN
    IF :new.salary > c_max OR
        :new.salary < c_min THEN
        RAISE_APPLICATION_ERROR(-20000,
            'New salary is too small or large');
    END IF;
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Introducción a Disparadores PL/SQL

Un disparador es un bloque PL/SQL que se ejecuta cuando se produce un evento concreto.

Existen dos formas de disparador PL/SQL:

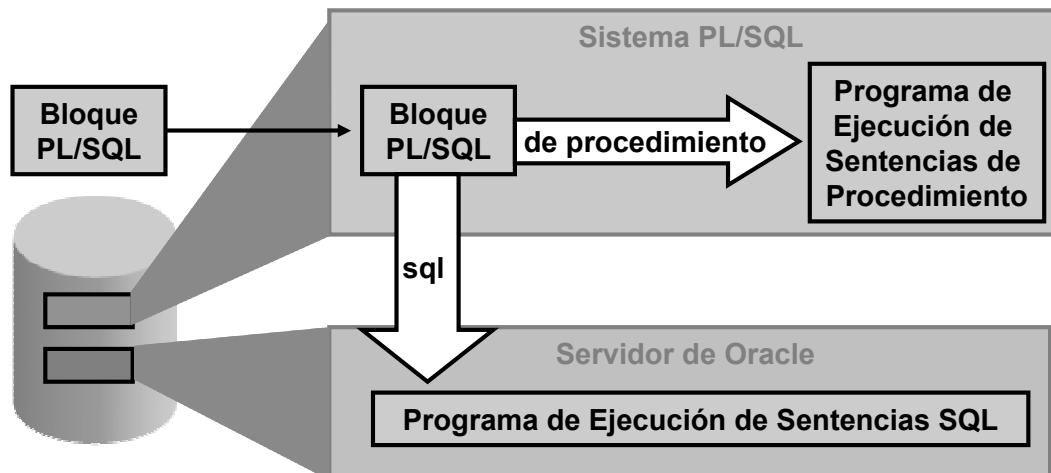
- Disparadores de la aplicación, que se crean como bloques de código que se ejecutan cuando se produce un evento de aplicación con nombre en un entorno de ejecución de Oracle Forms.
- Disparadores de la base de datos, que son bloques de código con nombre asociados que se ejecutan cuando se produce un evento de tabla o base de datos con nombre.

El ejemplo del disparador `check_salary` muestra un disparador de base de datos que se ejecuta antes de que se produzca una operación `INSERT` o `UPDATE` en la tabla `EMPLOYEES`. El código del disparador comprueba si el valor de la columna `salary` está dentro de un rango aceptable. Si el valor está fuera del rango especificado, el código utiliza el procedimiento incorporado `RAISE_APPLICATION_ERROR` para que falle la operación. La sintaxis `:new`, que se utiliza en el ejemplo, es una variable ligada o del host especial que se puede utilizar en disparadores de nivel de fila.

Los disparadores se abordan en detalle en la lección titulada “Creación de Disparadores”.

Entorno de Ejecución de PL/SQL

Arquitectura de tiempo de ejecución de PL/SQL:



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Entorno de Ejecución de PL/SQL

El diagrama muestra la ejecución de un bloque PL/SQL por parte de un sistema PL/SQL.

El sistema PL/SQL reside en:

- La base de datos Oracle para ejecutar subprogramas almacenados
- El cliente Oracle Forms al ejecutar aplicaciones de cliente/servidor o en Oracle Application Server al utilizar Oracle Forms Services para ejecutar Forms en la Web

Independientemente del entorno de tiempo de ejecución de PL/SQL, la arquitectura básica se mantiene igual. Por lo tanto, todas las sentencias PL/SQL se procesan en el programa de ejecución de sentencias de procedimiento y todas las sentencias SQL se deben enviar al programa de ejecución de sentencias SQL para que el servidor de Oracle las procese.

El sistema PL/SQL es un sistema virtual que reside en memoria y procesa las instrucciones de código m de PL/SQL. Cuando el sistema PL/SQL detecta una sentencia SQL, se produce un cambio de contexto para transferir la sentencia SQL a los procesos del servidor de Oracle. El sistema PL/SQL espera a que termine la sentencia SQL y se devuelvan los resultados antes de continuar con el procesamiento de las siguientes sentencias del bloque PL/SQL.

El sistema PL/SQL de Oracle Forms se ejecuta en el cliente para la implementación del cliente o servidor y en el servidor de aplicaciones para la implementación de Forms Services. En cualquier caso, las sentencias SQL se envían normalmente por una red a un servidor de Oracle para su procesamiento.

Entornos de Desarrollo de PL/SQL

Este curso proporciona las siguientes herramientas para desarrollar código PL/SQL:

- **Oracle SQL*Plus (tanto versión GUI como versión de línea de comandos)**
- **Oracle iSQL*Plus (utilizado desde un explorador)**
- **Oracle JDeveloper IDE**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

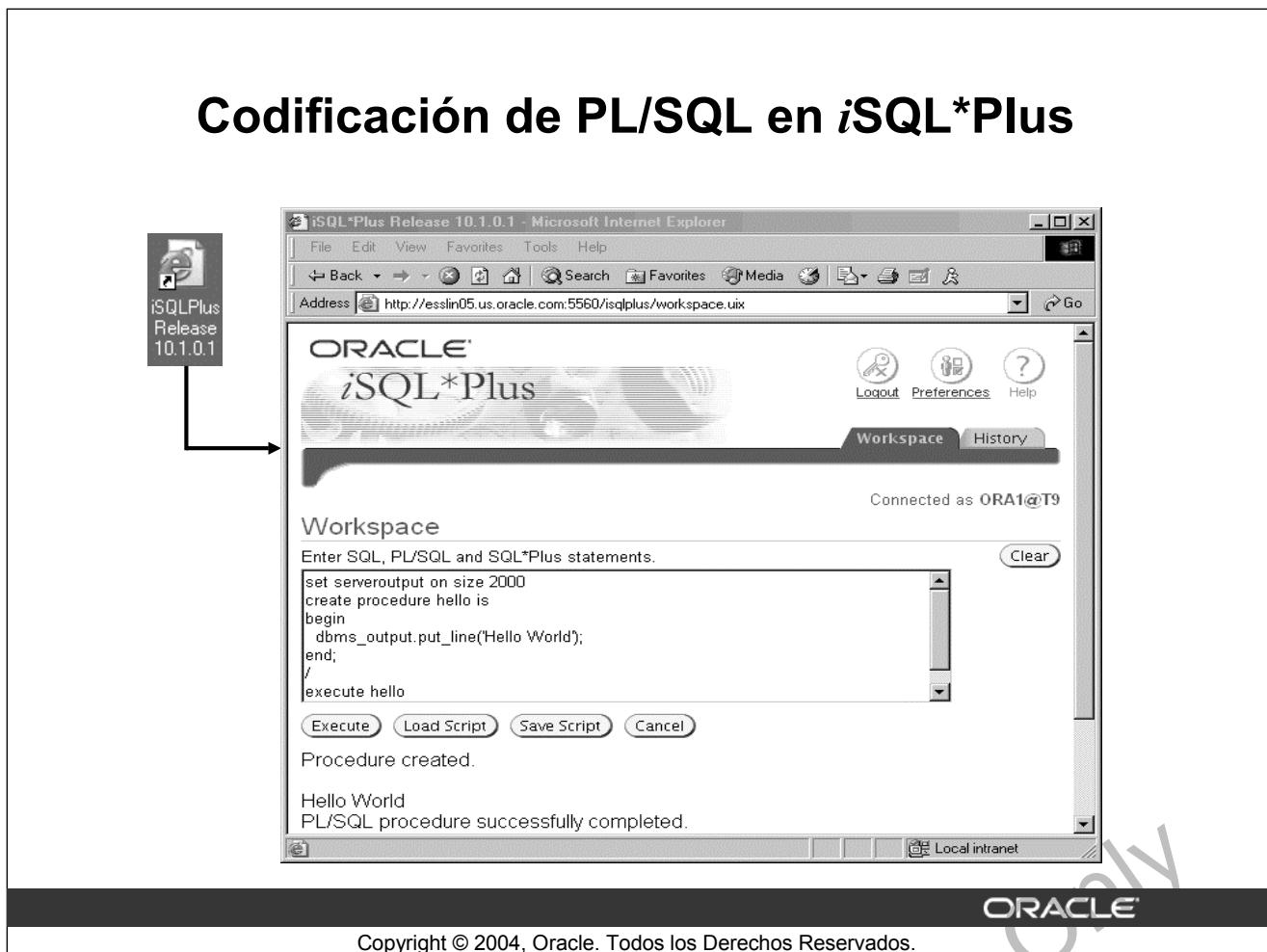
Entornos de Desarrollo de PL/SQL

Existen numerosas herramientas que proporcionan entornos de desarrollo de código PL/SQL. Oracle proporciona varias herramientas que se pueden utilizar para escribir código PL/SQL. Algunas de las herramientas de desarrollo disponibles en este curso son:

- Oracle iSQL*Plus: aplicación basada en explorador
- Oracle SQL*Plus: aplicación de ventanas o línea de comandos
- Oracle JDeveloper: entorno de desarrollo de integración (IDE) basado en ventanas

Nota: Los ejemplos de código y pantalla que se presentan en las notas del curso se han generado a partir de la salida del entorno iSQL*Plus.

Codificación de PL/SQL en iSQL*Plus



Codificación de PL/SQL en iSQL*Plus

Oracle iSQL*Plus es una aplicación Web que permite ejecutar sentencias SQL y bloques PL/SQL para ejecución y recibir los resultados en un explorador Web estándar.

iSQL*Plus:

- Se incluye con la base de datos.
- Se instala en la capa media.
- Es accesible desde un explorador Web mediante un formato de dirección URL similar al del siguiente ejemplo:

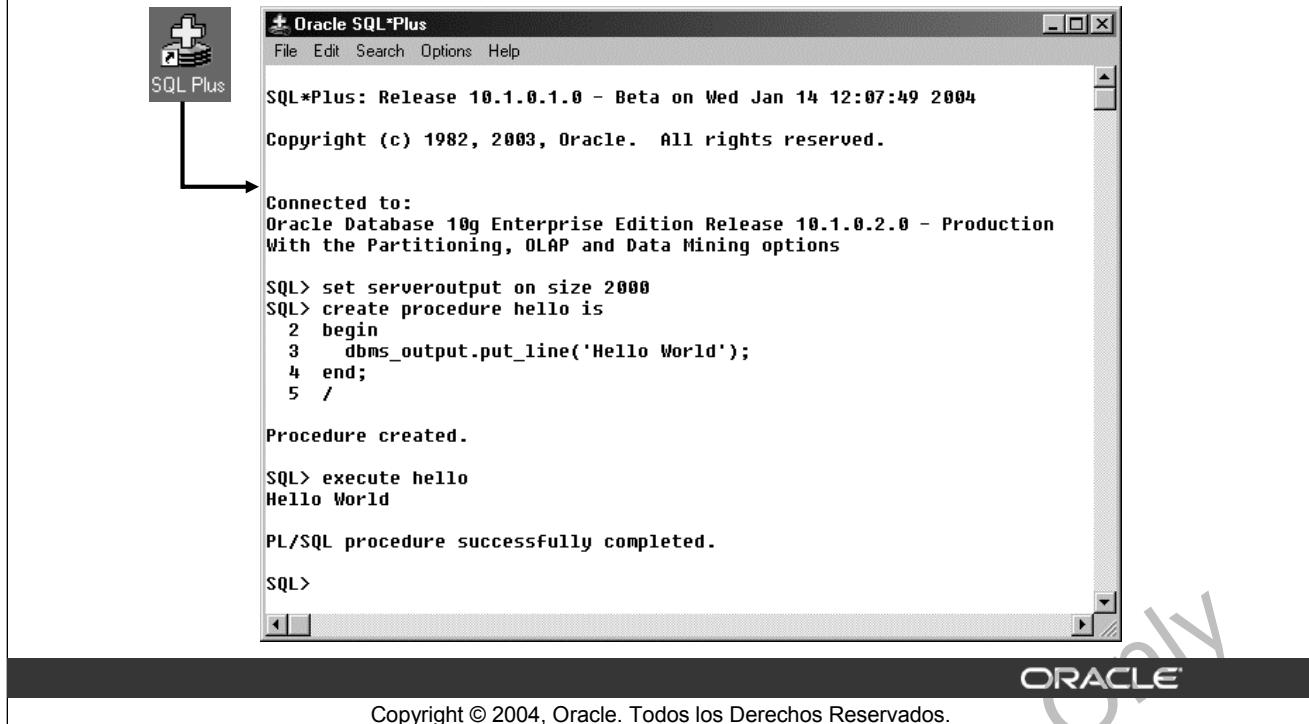
http://host:port/isqlplus

Los valores host y port corresponden al nombre del servidor Web y al puerto del listener HTTP.

Al codificar subprogramas PL/SQL en la herramienta iSQL*Plus, tenga en cuenta lo siguiente:

- Los subprogramas se crean utilizando la sentencia CREATE_SQL.
- Los subprogramas se ejecutan con un bloque PL/SQL anónimo o el comando EXECUTE.
- Si se utilizan los procedimientos del paquete DBMS_OUTPUT para imprimir texto en la pantalla, primero debe ejecutar el comando SET SERVEROUTPUT ON en la sesión.

Codificación de PL/SQL en SQL*Plus



Codificación de PL/SQL en SQL*Plus

Oracle SQL*Plus es una interfaz gráfica de usuario (GUI) o aplicación de línea de comandos que permite ejecutar sentencias SQL y bloques PL/SQL para ejecución y recibir los resultados en una ventana de comando o de aplicación.

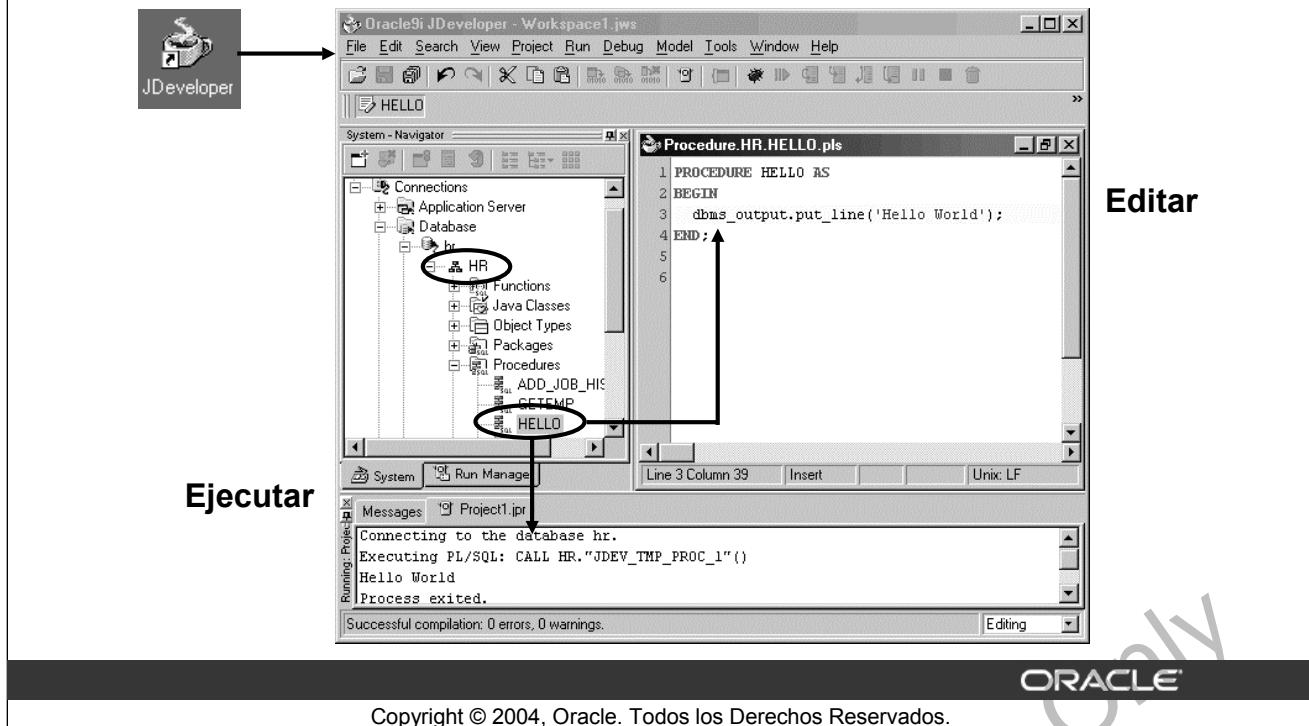
SQL*Plus:

- Se incluye con la base de datos.
- Se instala en un sistema de cliente y servidor de bases de datos.
- Es accesible desde un ícono o la línea de comandos.

Al codificar subprogramas PL/SQL con SQL*Plus, tenga en cuenta lo siguiente:

- Los subprogramas se crean utilizando la sentencia CREATE SQL.
- Los subprogramas se ejecutan con un bloque PL/SQL anónimo o el comando EXECUTE.
- Si se utilizan los procedimientos del paquete DBMS_OUTPUT para imprimir texto en la pantalla, primero debe ejecutar el comando SET SERVEROUTPUT ON en la sesión.

Codificación de PL/SQL en Oracle JDeveloper



Codificación de PL/SQL en Oracle JDeveloper

Oracle JDeveloper permite a los desarrolladores crear, editar, probar y depurar código PL/SQL con una GUI sofisticada. Oracle JDeveloper forma parte de Oracle Developer Suite y está también disponible como producto independiente.

Al codificar PL/SQL en JDeveloper, tenga en cuenta lo siguiente:

- Primero debe crear una conexión a la base de datos para permitir que JDeveloper acceda a un propietario de esquema de base de datos para los subprogramas.
- A continuación, puede utilizar los menús contextuales de JDeveloper en la conexión a la base de datos para crear una nueva construcción de subprograma con el editor de códigos de JDeveloper incorporado. Este editor de códigos proporciona un entorno excelente para el desarrollo de PL/SQL, con funciones como las siguientes:
 - Distintos colores para componentes sintácticos del lenguaje PL/SQL
 - Perspectiva de código para buscar rápidamente procedimientos y funciones en los paquetes proporcionados

Se llama a un subprograma utilizando el comando Run del menú contextual del subprograma con nombre. El resultado aparece en la ventana JDeveloper Log Message, tal como se muestra en la parte inferior de la captura de pantalla.

Nota: JDeveloper proporciona sintaxis con codificación de color en el editor de códigos de JDeveloper y es sensible a las sentencias y construcciones del lenguaje PL/SQL.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- **Declarar bloques PL/SQL con nombre:
Procedimientos, funciones, paquetes y
disparadores**
- **Utilizar bloques PL/SQL anónimos (sin nombre)
para llamar a las funciones y procedimientos
almacenados**
- **Utilizar iSQL*Plus o SQL*Plus para desarrollar
código PL/SQL**
- **Explicar el entorno de ejecución de PL/SQL:**
 - **El sistema PL/SQL del cliente para ejecutar código
PL/SQL en Oracle Forms y Oracle Reports**
 - **El sistema PL/SQL del servidor para ejecutar código
PL/SQL almacenado en una base de datos Oracle**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen

El lenguaje PL/SQL proporciona distintas construcciones de programas para bloques de código reutilizable. Los bloques PL/SQL sin nombre o anónimos se pueden utilizar para llamar a acciones, procedimientos, funciones y componentes de paquetes SQL y PL/SQL. Los bloques PL/SQL con nombre, también denominados subprogramas, incluyen:

- Procedimientos
- Funciones
- Funciones y procedimientos de paquetes
- Disparadores

Oracle proporciona varias herramientas para desarrollar la funcionalidad PL/SQL.

Oracle proporciona un entorno de tiempo de ejecución de PL/SQL de capa media o de cliente para Oracle Forms y Oracle Reports y proporciona un sistema de tiempo de ejecución de PL/SQL dentro de la base de datos Oracle. Los procedimientos y funciones de la base de datos se pueden llamar desde cualquier código de aplicación que pueda conectar a una base de datos Oracle y ejecutar código PL/SQL.

Práctica I: Visión General

En esta práctica se abordan los siguientes temas:

- **Exploración de tablas de HR**
- **Creación de un procedimiento PL/SQL simple**
- **Creación de una función PL/SQL simple**
- **Uso de un bloque anónimo para ejecutar la función y el procedimiento PL/SQL**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica I: Visión General

En esta práctica, se utiliza *iSQL*Plus* para ejecutar sentencias SQL para examinar los datos del esquema HR. También se crea un procedimiento y función simples que se llama mediante un bloque anónimo o el comando EXECUTE en *iSQL*Plus*. Opcionalmente, puede intentar crear y ejecutar el código PL/SQL en *SQL*Plus*.

Nota: Todas las prácticas escritas utilizan *iSQL*Plus* como entorno de desarrollo. Sin embargo, puede utilizar cualquiera de las herramientas que se proporcionan en el entorno del curso.

Práctica I

1. Inicie iSQL*Plus con el icono que aparece en el escritorio.
 - a. Conéctese a la base de datos utilizando el nombre de usuario y los datos de la cadena de conexión de la base de datos que le ha proporcionado el instructor (también puede escribir aquí la información para tenerla guardada):
Username: **ora_**
Password: **oracle**
Database Connect String/Tnsname: **T_**
 - b. Ejecute sentencias SELECT básicas para consultar los datos de las tablas DEPARTMENTS, EMPLOYEES y JOBS. Dedique unos cuantos minutos a familiarizarse con los datos o consulte el apéndice B que proporciona una descripción y algunos datos de cada tabla del esquema Human Resources.
2. Cree un procedimiento denominado HELLO para mostrar el texto “Hello World”.
 - a. Cree un procedimiento denominado HELLO.
 - b. En la sección ejecutable, utilice el procedimiento DBMS_OUTPUT.PUT_LINE para imprimir el texto “Hello Word” y guardar el código en la base de datos.
Nota: Si obtiene errores en tiempo de compilación, edite PL/SQL para corregir el código y sustituir la palabra clave CREATE con el texto CREATE OR REPLACE.
 - c. Ejecute el comando SET SERVEROUTPUT ON para asegurarse de que la salida del procedimiento DBMS_OUTPUT.PUT_LINE se muestra en iSQL*Plus.
 - d. Cree un bloque anónimo para llamar al procedimiento almacenado.
3. Cree una función denominada TOTAL_SALARY para calcular la suma de los sueldos de todos los empleados.
 - a. Cree una función denominada TOTAL_SALARY que devuelve un NUMBER.
 - b. En la sección ejecutable, ejecute una consulta para almacenar el salario total de todos los empleados en una variable local que se declara en la sección de declaraciones. Devuelva el valor almacenado en la variable local. Guarde y compile el código.
 - c. Utilice un bloque anónimo para llamar a la función. Para mostrar el resultado calculado por la función utilice el procedimiento DBMS_OUTPUT.PUT_LINE.
Indicación: Anide la llamada de función dentro del parámetro DBMS_OUTPUT.PUT_LINE o almáocene el resultado de la función en una variable local del bloque anónimo y utilice la variable local en el procedimiento DBMS_OUTPUT.PUT_LINE.

Si tiene tiempo, realice el siguiente ejercicio:

4. Inicie SQL*Plus con el icono que aparece en el escritorio.
 - a. Llame al procedimiento y a la función que ha creado en los ejercicios 2 y 3.
 - b. Cree un procedimiento nuevo denominado HELLO AGAIN para imprimir “Hello World again”.
 - c. Llame al procedimiento HELLO AGAIN con un bloque anónimo.

Oracle Internal & OAI Use Only

Creación de Procedimientos Almacenados

1

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- Describir y crear un procedimiento
- Crear procedimientos con parámetros
- Diferenciar entre parámetros formales y reales
- Utilizar diferentes modos de transferencia de parámetros
- Llamar a un procedimiento
- Manejar excepciones en procedimientos
- Eliminar un procedimiento

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetivos

En esta lección, aprenderá a crear, ejecutar y eliminar procedimientos con o sin parámetros. Los procedimientos son la base de la programación modular en PL/SQL. Para hacer que los procedimientos sean más flexibles, es importante que los datos variables se calculen o transfieran a un procedimiento mediante parámetros de entrada. Los resultados calculados se pueden devolver al emisor de la llamada de un procedimiento con parámetros de salida.

Para hacer que los programas sean robustos, siempre debe manejar las condiciones de excepción con las funciones de manejo de excepciones de PL/SQL.

¿Qué es un Procedimiento?

Un procedimiento:

- **Es un tipo de subprograma que realiza una acción**
- **Se puede almacenar en la base de datos como objeto de esquema**
- **Fomenta la capacidad de reutilización y mantenimiento**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Definición de un Procedimiento

Un procedimiento es un bloque PL/SQL con nombre que puede aceptar parámetros (a veces denominados argumentos). Generalmente, un procedimiento se utiliza para realizar una acción. Tiene una cabecera, una sección de declaraciones, una sección ejecutable y una sección de manejo de excepciones opcional. Para llamar a un procedimiento, se utiliza el nombre del procedimiento en la sección ejecutable de otro bloque PL/SQL.

Un procedimiento se compila y almacena en la base de datos como objeto de esquema. Si está utilizando los procedimientos con Oracle Forms y Reports, éstos se pueden compilar dentro de los ejecutables de Oracle Forms u Oracle Reports.

Los procedimientos fomentan la capacidad de reutilización y mantenimiento. Al validarlos, se pueden utilizar en cualquier número de aplicaciones. Si los requisitos cambian, sólo es necesario actualizar el procedimiento.

Sintaxis para Crear Procedimientos

- Utilice **CREATE PROCEDURE** seguido del nombre, los parámetros opcionales y la palabra clave **IS** o **AS**.
- Agregue la opción **OR REPLACE** para sustituir un procedimiento existente.
- Escriba un bloque PL/SQL que contenga variables locales, un valor **BEGIN** y un valor **END** (o **END procedure_name**).

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [(parameter1 [mode] datatype1,
    parameter2 [mode] datatype2, ...)]
  IS|AS
    [local_variable_declarations; ...]
  BEGIN
    -- actions;
  END [procedure_name];
```

→ Bloque PL/SQL

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Sintaxis para Crear Procedimientos

Se crean nuevos procedimientos con la sentencia CREATE PROCEDURE, que puede declarar una lista de parámetros y debe definir las acciones que debe realizar el bloque PL/SQL estándar. La cláusula CREATE permite crear procedimientos autónomos que se almacenan en una base de datos Oracle.

- Los bloques PL/SQL empiezan por BEGIN, precedido opcionalmente por la declaración de variables locales. Los bloques PL/SQL terminan en END o END *procedure_name*.
- La opción REPLACE indica que si el procedimiento existe, se borra y se sustituye por la nueva versión creada por la sentencia.

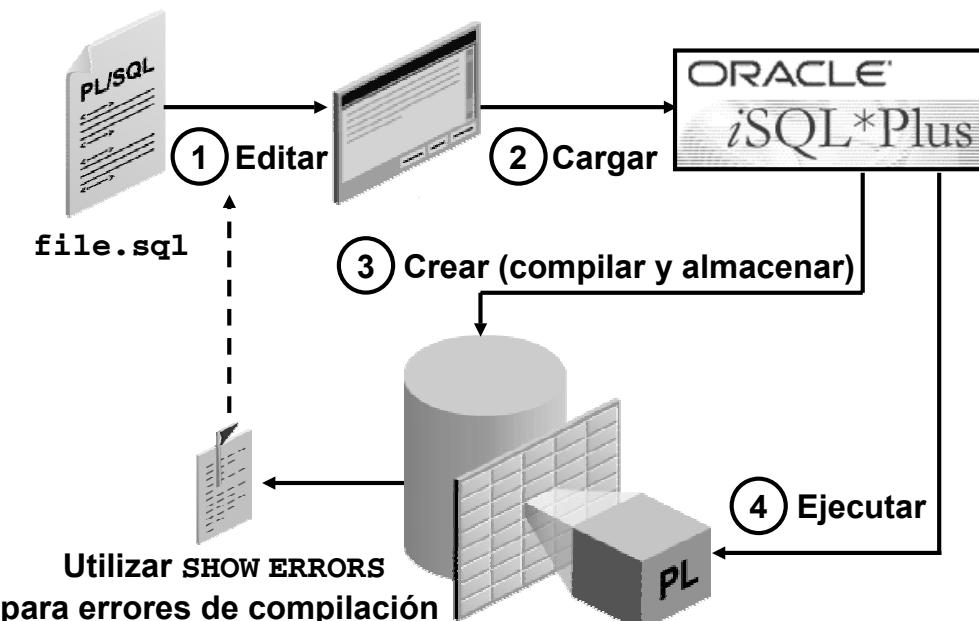
Otros Elementos Sintácticos

- *parameter1* representa el nombre de un parámetro.
- La opción *mode* define cómo se utiliza un parámetro: IN (valor por defecto), OUT o IN OUT.
- *datatype1* especifica el tipo de dato del parámetro, sin ninguna precisión.

Nota: Los parámetros se pueden considerar como variables locales. No se puede hacer referencia a las variables de sustitución ni de host (ligadas) en la definición de un procedimiento PL/SQL almacenado.

La opción OR REPLACE no necesita ningún cambio en la seguridad de los objetos, siempre y cuando el usuario sea el propietario del objeto y tenga el privilegio CREATE [ANY] PROCEDURE.

Desarrollo de Procedimientos



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Desarrollo de Procedimientos

Para desarrollar un procedimiento almacenado, realice los siguientes pasos:

1. Escriba el código para crear un procedimiento en un editor o procesador de texto y, a continuación, guárdelo como archivo de comandos SQL (normalmente con una extensión .sql).
2. Cargue el código en una de las herramientas de desarrollo como SQL*Plus o iSQL*Plus.
3. Cree el procedimiento en la base de datos. La sentencia CREATE PROCEDURE compila y almacena el código de origen y el valor de *m-code* compilado en la base de datos. Si se produce un error de compilación, el valor de *m-code* no se almacena y debe editar el código de origen para realizar las correcciones. No se puede llamar a un procedimiento que contenga errores de compilación. Para ver los errores de compilación en SQL*Plus o iSQL*Plus, utilice:
 - SHOW ERRORS para el procedimiento compilado más recientemente (el último)
 - SHOW ERRORS PROCEDURE *procedure_name* para cualquier procedimiento compilado anteriormente
4. Después de terminar la compilación correctamente, ejecute el procedimiento para realizar la acción deseada. Utilice el comando EXECUTE de iSQL*Plus o un bloque PL/SQL anónimo desde entornos que soportan PL/SQL.

Nota: Si se producen errores de compilación, utilice una sentencia CREATE OR REPLACE PROCEDURE para sustituir el código existente si ha utilizado anteriormente una sentencia CREATE PROCEDURE. De lo contrario, realice una operación DROP en el procedimiento y, a continuación, ejecute la sentencia CREATE PROCEDURE.

¿Qué son los Parámetros?

Parámetros:

- **Se declaran después del nombre del subprograma en la cabecera PL/SQL**
- **Transfieren o comunican datos entre el emisor de la llamada y el subprograma**
- **Se utilizan como variables locales, pero dependen del modo de transferencia de parámetros:**
 - Un parámetro **IN** (valor por defecto) proporciona valores para que un subprograma los procese.
 - Un parámetro **OUT** devuelve un valor al emisor de la llamada.
 - Un parámetro **IN OUT** proporciona un valor de entrada, que se puede devolver (salida) como valor modificado.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

¿Qué son los Parámetros?

Los parámetros se utilizan para transferir valores de datos a y desde el entorno de llamada y el procedimiento (o subprograma). Los parámetros se declaran en la cabecera del subprograma, después del nombre y antes de la sección de declaraciones de las variables locales.

Los parámetros están sujetos a uno de los tres modos de transferencia de parámetros: **IN**, **OUT** o **IN OUT**.

- Un parámetro **IN** transfiere un valor constante desde el entorno de llamada al procedimiento.
- Un parámetro **OUT** transfiere un valor del procedimiento al entorno de llamada.
- Un parámetro **IN OUT** transfiere un valor del entorno de llamada al procedimiento y un valor posiblemente distinto del procedimiento al entorno de llamada mediante el mismo parámetro.

Los parámetros se pueden considerar como una forma especial de variable local, cuyos valores de entrada inicializa el entorno de llamada al llamar al subprograma y cuyos valores de salida se devuelven al entorno de llamada cuando el subprograma devuelve el control al emisor de la llamada.

Parámetros Formales y Reales

- **Parámetros formales:** Variables locales que se declaran en la lista de parámetros de una especificación de subprograma

Ejemplo:

```
CREATE PROCEDURE raise_sal(id NUMBER,sal NUMBER) IS  
BEGIN ...  
END raise_sal;
```

- **Parámetros reales:** Valores literales, variables o expresiones utilizadas en la lista de parámetros del subprograma llamado

Ejemplo:

```
emp_id := 100;  
raise_sal(emp_id, 2000)
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Parámetros Formales y Reales

Los parámetros formales son variables locales que se declaran en la lista de parámetros de una especificación de subprograma. En el primer ejemplo, en el procedimiento `raise_sal`, los identificadores `id` y `sal` variables representan los parámetros formales.

Los parámetros reales pueden ser valores literales, variables o expresiones que se proporcionan en la lista de parámetros de un subprograma llamado. En el segundo ejemplo, se realiza una llamada a `raise_sal`, donde la variable `emp_id` proporciona el valor de parámetro real para el parámetro formal `id` y `2000` se proporciona como el valor de parámetro real para `sal`.

Parámetros reales:

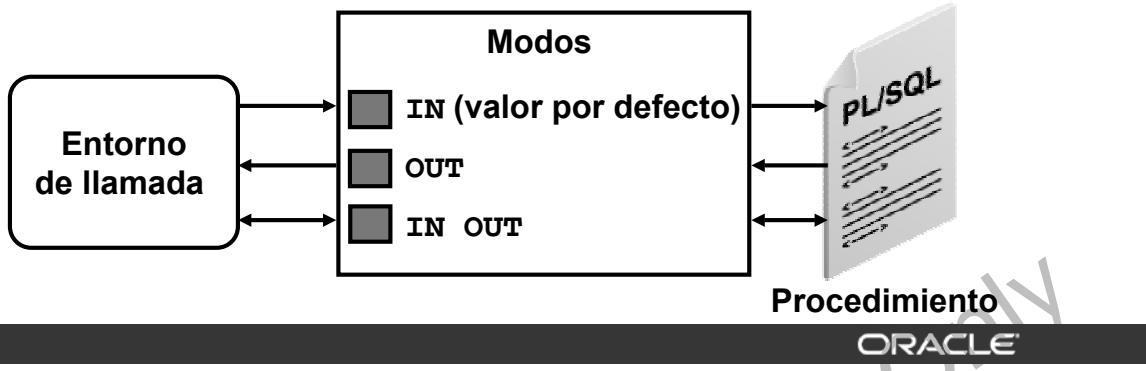
- Se asocian a parámetros formales durante la llamada al subprograma.
- Pueden ser también expresiones, como en el siguiente ejemplo:
`raise_sal(emp_id, raise+100);`

Los parámetros formales y reales deben ser de tipos de dato compatibles. Si es necesario, antes de asignar el valor, PL/SQL convierte el tipo de dato del valor de parámetro real al del parámetro formal.

Modos de Parámetros de Procedimiento

- Los modos de parámetros se especifican en la declaración de parámetros formales, después del nombre del parámetro y antes del tipo de dato.
- El modo **IN** es el valor por defecto si no se especifica ningún modo.

```
CREATE PROCEDURE procedure(param [mode] datatype)
...
```



Copyright © 2004, Oracle. Todos los Derechos Reservados.

Modos de Parámetros de Procedimiento

Al crear el procedimiento, el parámetro formal define un nombre de variable cuyo valor se utiliza en la sección ejecutable del bloque PL/SQL. El parámetro real se utiliza al llamar al procedimiento para proporcionar los valores de entrada o recibir los resultados de salida.

El modo de parámetro **IN** es el modo de transferencia por defecto. Es decir, si no se especifica ningún modo con una declaración de parámetro, el parámetro se considera un parámetro **IN**. Los modos de parámetros **OUT** y **IN OUT** se deben especificar explícitamente en las declaraciones de parámetros.

El parámetro *datatype* se especifica sin especificación de tamaño. Se puede especificar:

- Como tipo de dato explícito
- Con la definición %TYPE
- Con la definición %ROWTYPE

Nota: Se pueden declarar uno o más parámetros formales, separándolos entre sí mediante una coma.

Uso de Parámetros IN: Ejemplo

```
CREATE OR REPLACE PROCEDURE raise_salary
  →(id      IN employees.employee_id%TYPE,
    percent IN NUMBER) ←
  IS
  BEGIN
    UPDATE employees
    SET salary = salary * (1 + percent/100)
    WHERE employee_id = id;
  END raise_salary;
  /
```



```
EXECUTE raise_salary(176,10)
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de Parámetros IN: Ejemplo

El ejemplo muestra un procedimiento con dos parámetros IN. Al ejecutar esta primera sentencia en iSQL*Plus se crea el procedimiento `raise_salary` en la base de datos. El segundo ejemplo llama a `raise_salary` y proporciona el primer valor de parámetro 176 para el identificador de empleado y un aumento del porcentaje del salario del 10% para el segundo valor de parámetro.

Para llamar a un procedimiento en iSQL*Plus, utilice el siguiente comando EXECUTE:

```
EXECUTE raise_salary (176, 10)
```

Para llamar a un procedimiento desde otro procedimiento, utilice una llamada directa dentro de una sección ejecutable del bloque de llamada. En la ubicación de la llamada al nuevo procedimiento, introduzca el nombre del procedimiento y los parámetros reales. Por ejemplo:

```
...
BEGIN
  raise_salary (176, 10);
END;
```

Nota: Los parámetros IN se transfieren como valores de sólo lectura del entorno de llamada al procedimiento. Cualquier intento de cambiar el valor de un parámetro IN puede dar como resultado un error en tiempo de compilación.

Uso de Parámetros OUT: Ejemplo

```
CREATE OR REPLACE PROCEDURE query_emp
  (id      IN employees.employee_id%TYPE,
   name    OUT employees.last_name%TYPE,
   salary  OUT employees.salary%TYPE) IS
BEGIN
  SELECT last_name, salary INTO name, salary
  FROM employees
  WHERE employee_id = id;
END query_emp;
```

```
DECLARE
  emp_name employees.last_name%TYPE;
  emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(171, emp_name, emp_sal); ...
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de Parámetros OUT: Ejemplo

En este ejemplo, se crea un procedimiento con parámetros OUT para recuperar información sobre un empleado. El procedimiento acepta el valor 171 para el identificador de empleado y recupera el nombre y el salario del empleado con identificador 171 en los dos parámetros OUT. El procedimiento `query_emp` tiene tres parámetros formales. Dos de ellos son parámetros OUT que devuelven valores al entorno de llamada, que se muestra en el recuadro de código situado en la parte inferior de la transparencia. El procedimiento acepta un valor de identificador de empleado a través del parámetro `id`. Las variables `emp_name` y `emp_salary` se llenan con la información recuperada de la consulta en los dos parámetros OUT correspondientes.

Si imprime los valores devueltos en variables PL/SQL del bloque de llamada que se muestra en el segundo bloque de código, las variables contendrán los siguientes valores:

- `emp_name` contiene el valor Smith.
- `emp_salary` contiene el valor 7600.

Nota: Asegúrese de que el tipo de dato de las variables de parámetros reales utilizadas para recuperar valores de los parámetros OUT tiene el tamaño suficiente para contener los valores de datos devueltos.

Si intenta utilizar o leer parámetros OUT dentro del procedimiento que los declara, se producirá un error de compilación. Los parámetros OUT pueden ser valores asignados únicamente en el cuerpo del procedimiento en el que se declaran.

Visualización de Parámetros OUT con iSQL*Plus

- Utilice las variables PL/SQL que se imprimen con llamadas al procedimiento DBMS_OUTPUT.PUT_LINE.

```
SET SERVEROUTPUT ON
DECLARE
    emp_name employees.last_name%TYPE;
    emp_sal  employees.salary%TYPE;
BEGIN
    query_emp(171, emp_name, emp_sal);
    DBMS_OUTPUT.PUT_LINE('Name: ' || emp_name);
    DBMS_OUTPUT.PUT_LINE('Salary: ' || emp_sal);
END;
```

- Utilice variables del host de iSQL*Plus, ejecute QUERY_EMP con variables del host e imprima las variables del host.

```
VARIABLE name VARCHAR2(25)
VARIABLE sal NUMBER
EXECUTE query_emp(171, :name, :sal)
PRINT name sal
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Visualización de Parámetros OUT con iSQL*Plus

Los ejemplos muestran dos formas de visualizar los valores devueltos de los parámetros OUT.

- La primera técnica utiliza variables PL/SQL en un bloque anónimo para recuperar los valores del parámetro OUT. El procedimiento DBMS_OUTPUT.PUT_LINE se llama para imprimir los valores que contienen las variables PL/SQL. SET SERVEROUTPUT debe estar en modo ON.
- La segunda técnica muestra cómo utilizar las variables iSQL*Plus que se crean con el comando VARIABLE. Las variables iSQL*Plus son externas al bloque PL/SQL y se conocen como variables ligadas o de host. Para hacer referencia a variables de host desde un bloque PL/SQL, debe anteponer dos puntos (:) a los nombres. Para mostrar los valores almacenados en las variables de host, debe utilizar el comando PRINT de iSQL*Plus seguido del nombre de la variable iSQL*Plus (sin los dos puntos, ya que no se trata de un comando PL/SQL o un contexto).

Para utilizar variables de host y iSQL*Plus al llamar a un procedimiento con los parámetros OUT, realice los siguientes pasos:

1. Cree un archivo de comandos iSQL*Plus mediante un editor.
2. Agregue comandos para crear las variables, ejecute el procedimiento e imprima las variables.
3. Cargue y ejecute el archivo de comandos iSQL*Plus.

Nota: Para obtener más información sobre el comando VARIABLE, consulte iSQL*Plus Command Reference.

Llamada a PL/SQL con Variables de Host

Una variable de host (también denominada variable ligada o global):

- **Se declara y existe fuera del subprograma PL/SQL. Se puede crear en:**
 - iSQL*Plus con el comando **VARIABLE**
 - **Variables de interfaz de usuario e internas de Oracle Forms**
 - **Variables Java**
- **Está precedida de dos puntos (:) cuando se hace referencia a ella en código PL/SQL**
- **Se puede hacer referencia a ella en un bloque anónimo, pero no en un subprograma almacenado**
- **Proporciona un valor a un bloque PL/SQL y recibe un valor de un bloque PL/SQL**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Llamada a PL/SQL con Variables de Host

El código PL/SQL que se almacena en la base de datos se puede llamar desde una gran variedad de entornos, como:

- SQL*Plus o iSQL*Plus
- Oracle Forms y Oracle Reports
- Aplicaciones Java y C

Cada uno de los entornos indicados proporciona distintas formas de declarar variables para almacenar datos en la memoria. Los valores de variable de estas aplicaciones se definen y mantienen fuera del código PL/SQL almacenado. Cada entorno proporciona una forma de transferir los datos de las variables a PL/SQL y recibir valores actualizados del código PL/SQL. En general, la mayoría de los idiomas alojan llamadas a subprogramas o bloques PL/SQL.

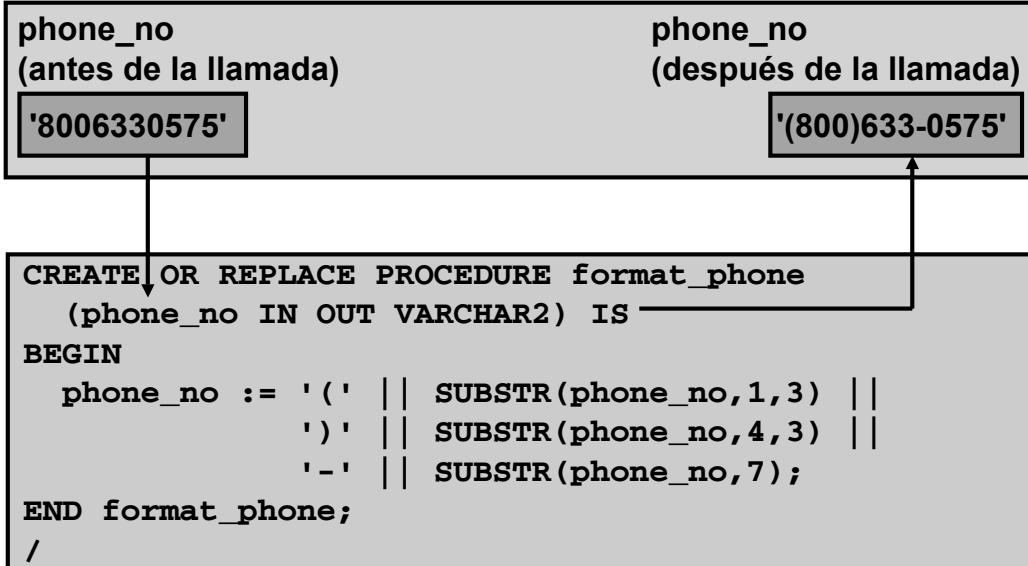
El sistema PL/SQL utiliza una técnica llamada enlace para asociar los valores proporcionados desde ubicaciones externas a variables PL/SQL o parámetros declarados en los subprogramas PL/SQL.

A diferencia de Java, PL/SQL reconoce las variables de host por la presencia de los dos puntos antes del nombre de la variable externa cuando se utiliza en un bloque PL/SQL.

No puede almacenar código PL/SQL con variables de host, ya que el compilador no puede resolver las referencias a variables de host. El proceso de enlace se realiza en tiempo de ejecución.

Uso de Parámetros IN OUT: Ejemplo

Entorno de llamada



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de Parámetros IN OUT: Ejemplo

Con un parámetro IN OUT, puede transferir un valor a un procedimiento que se puede actualizar. El valor de parámetro real proporcionado desde el entorno de llamada se puede devolver de las siguientes formas:

- Como el valor original sin cambiar
- Como un nuevo valor que se define dentro del procedimiento

Nota: Un parámetro IN OUT funciona como variable inicializada.

El ejemplo de la transparencia crea un procedimiento con un parámetro IN OUT para aceptar una cadena de 10 caracteres que contenga los dígitos de un número de teléfono. El procedimiento devuelve el número de teléfono con los tres primeros caracteres entre paréntesis y un guión después del sexto dígito. Por ejemplo, la cadena de teléfono 8006330575 se devuelve como (800) 633-0575.

El siguiente código utiliza la variable de host phone_no de iSQL*Plus para proporcionar el valor de entrada transferido al procedimiento FORMAT_PHONE. El procedimiento se ejecuta y devuelve una cadena actualizada en la variable de host phone_no.

```
VARIABLE phone_no VARCHAR2(15)
EXECUTE :phone_no := '8006330575'
PRINT phone_no
EXECUTE format_phone (:phone_no)
PRINT phone_no
```

Sintaxis de Transferencia de Parámetros

- **Posicional:**
 - Muestra los parámetros reales en el mismo orden que los parámetros formales
- **Con nombre:**
 - Muestra los parámetros reales en orden arbitrario y utiliza el operador de asociación ($=>$) para asociar un parámetro formal con nombre a su parámetro real
- **Combinación:**
 - Muestra algunos de los parámetros reales como posicionales y otros como con nombre

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Sintaxis de Transferencia de Parámetros

Para un procedimiento que contiene varios parámetros, puede utilizar diferentes métodos para especificar los valores de los parámetros. Los métodos son:

- Posicional, que muestra los valores de parámetro real en el orden en el que se declaran los parámetros formales.
- Con nombre, que muestra los valores reales en orden arbitrario y utiliza el operador de asociación para asociar cada parámetro real a su parámetro formal por nombre. El operador de asociación PL/SQL es un signo igual seguido de un signo mayor que, sin espacios: $=>$.
- Combinación, que muestra los primeros valores del parámetro por posición y el resto mediante la sintaxis especial del método con nombre.

La siguiente página muestra algunos ejemplos de los dos primeros métodos.

Transferencia de Parámetros: Ejemplos

```
CREATE OR REPLACE PROCEDURE add_dept(
    name IN departments.department_name%TYPE,
    loc  IN departments.location_id%TYPE) IS
BEGIN
    INSERT INTO departments(department_id,
                           department_name, location_id)
    VALUES (departments_seq.NEXTVAL, name, loc);
END add_dept;
/
```

- Transferencia por notación posicional

```
EXECUTE add_dept ('TRAINING', 2500)
```

- Transferencia por notación con nombre

```
EXECUTE add_dept (loc=>2400, name=>'EDUCATION')
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Transferencia de Parámetros: Ejemplos

En el ejemplo, el procedimiento `add_dept` declara dos parámetros IN: `name` y `loc`. Los valores de estos parámetros se utilizan en la sentencia `INSERT` para definir las columnas `department_name` y `location_id`, respectivamente.

La transferencia de parámetros por posición se muestra en la primera llamada para ejecutar `add_dept` debajo de la definición del procedimiento. El primer parámetro real proporciona el valor '`TRAINING`' para el parámetro `name`. El segundo valor de parámetro real `2500` se asigna por posición al parámetro `loc`.

La transferencia de parámetros con la notación con nombre se muestra en el último ejemplo. Aquí se hace referencia al último parámetro `loc`, que se declara como el segundo parámetro formal, según el nombre en la llamada donde se asocia al valor real `2400`. El parámetro `name` se asocia al valor '`EDUCATION`'. El orden de los parámetros reales es irrelevante si se especifican todos los valores de parámetro.

Nota: Debe proporcionar un valor para cada parámetro a menos que se asigne un valor por defecto al parámetro formal. La especificación de valores por defecto para parámetros formales se describe a continuación.

Uso de la Opción DEFAULT para Parámetros

- Define valores por defecto para parámetros:

```
CREATE OR REPLACE PROCEDURE add_dept (
    name departments.department_name%TYPE := 'Unknown',
    loc   departments.location_id%TYPE DEFAULT 1700)
IS
BEGIN
    INSERT INTO departments (... )
    VALUES (departments_seq.NEXTVAL, name, loc);
END add_dept;
```

- Proporciona flexibilidad combinando la sintaxis de transferencia de parámetros posicional y con nombre:

```
EXECUTE add_dept
EXECUTE add_dept ('ADVERTISING', loc => 1200)
EXECUTE add_dept (loc => 1200)
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de la Opción DEFAULT para Parámetros

Los ejemplos de código de la transparencia muestran dos formas de asignar un valor por defecto a un parámetro IN. Las dos formas que se muestran utilizan:

- El operador de asignación (:=), como se muestra para el parámetro name.
- La opción DEFAULT, como se muestra para el parámetro loc.

Cuando se asignan valores por defecto a parámetros formales, puede llamar al procedimiento sin proporcionar un valor de parámetro real para el parámetro. Por lo tanto, puede transferir distinta cantidad de parámetros reales a un subprograma, ya sea aceptando o sustituyendo los valores por defecto según sea necesario. Se recomienda declarar los parámetros sin valores por defecto primero. A continuación, puede agregar parámetros formales con valores por defecto sin tener que cambiar cada llamada al procedimiento.

Nota: No se pueden asignar valores por defecto a parámetros OUT e IN OUT.

La transparencia muestra tres formas de llamar al procedimiento add_dept:

- El primer ejemplo asigna los valores por defecto para cada parámetro.
- El segundo ejemplo ilustra una combinación de notación posicional y con nombre para asignar valores. En este caso, el uso de notación con nombre se muestra como ejemplo.
- El último ejemplo utiliza el valor por defecto del parámetro name y el valor proporcionado para el parámetro loc.

Uso de la Opción `DEFAULT` para Parámetros (continuación)

Normalmente, puede utilizar la notación con nombre para sustituir los valores por defecto de los parámetros formales. Sin embargo, no se puede saltar la especificación de un parámetro real si no se ha proporcionado ningún valor por defecto para un parámetro formal.

Nota: Todos los parámetros posicionales se deben anteponer a los parámetros con nombre en una llamada de subprograma. De lo contrario, recibirá un mensaje de error, como se muestra en el siguiente ejemplo:

```
EXECUTE add_dept(name=>'new dept', 'new location')
```

Se genera el siguiente mensaje de error:

```
ERROR at line 1:  
ORA-06550: line 1, column 7:  
PLS-00306: wrong number or types of arguments in call to  
'ADD_DEPT'  
ORA-06550: line 1, column 7:  
PL/SQL: Statement ignored
```

Oracle Internal & OAI Use Only

Resumen de los Modos de Parámetros

IN	OUT	IN OUT
Modo por defecto	Se debe especificar	Se debe especificar
El valor se transfiere al subprograma	Se devuelve al entorno de llamada	Se transfiere al subprograma; se devuelve al entorno de llamada
El parámetro formal funciona como una constante	Variable no inicializada	Variable inicializada
El parámetro real puede ser un literal, una expresión, una constante o una variable inicializada	Debe ser una variable	Debe ser una variable
Se le puede asignar un valor por defecto	No se le puede asignar un valor por defecto	No se le puede asignar un valor por defecto

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen de los Modos de Parámetros

El modo de parámetro IN es el modo por defecto si no se especifica ningún modo en la declaración. Los modos de parámetro OUT e IN OUT se deben especificar explícitamente en las declaraciones de parámetros.

No se puede asignar un valor a un parámetro formal del modo IN ni modificar éste en el cuerpo del procedimiento. Por defecto, el parámetro IN se transfiere por referencia. Se puede asignar un valor por defecto a un parámetro IN en la declaración de parámetro formal, en cuyo caso el emisor de la llamada no tendrá que proporcionar un valor para el parámetro si se aplica el valor por defecto.

Los parámetros OUT o IN OUT deben tener un valor asignado antes de volver al entorno de llamada. No se pueden asignar valores por defecto a los parámetros OUT e IN OUT. Para mejorar el rendimiento con parámetros OUT e IN OUT, se puede utilizar la indicación del compilador NOCOPY para solicitar la transferencia por referencia.

Nota: El uso de NOCOPY se describe más adelante en este curso.

Llamada a los Procedimientos

Puede llamar a los parámetros mediante:

- Bloques anónimos
- Otro procedimiento, como en el siguiente ejemplo:

```
CREATE OR REPLACE PROCEDURE process_employees
IS
    CURSOR emp_cursor IS
        SELECT employee_id
        FROM employees;
BEGIN
    FOR emp_rec IN emp_cursor
    LOOP
        raise_salary(emp_rec.employee_id, 10);
    END LOOP;
    COMMIT;
END process_employees;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Llamada a los Procedimientos

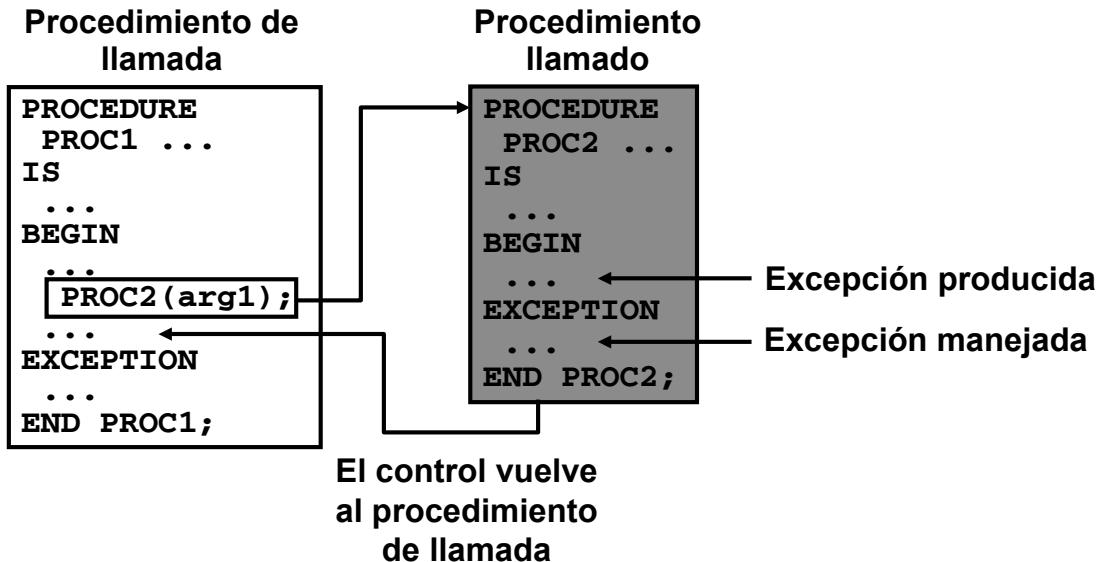
Puede llamar a los procedimientos mediante:

- Bloques anónimos
- Otro procedimiento o subprograma PL/SQL

Algunos ejemplos de las páginas anteriores ilustran cómo utilizar bloques anónimos (o el comando EXECUTE en iSQL*Plus).

Este ejemplo muestra cómo llamar a un procedimiento desde otro procedimiento almacenado. El procedimiento almacenado PROCESS_EMPS utiliza un cursor para procesar todos los registros de la tabla EMPLOYEES y transfiere cada identificador de empleado al procedimiento RAISE_SALARY, lo que da como resultado un incremento de salario del 10% en la compañía.

Excepciones Manejadas



ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Excepciones Manejadas

Al desarrollar procedimientos llamados desde otros procedimientos, debe tener en cuenta los efectos que tienen las excepciones manejadas y no manejadas en la transacción y el procedimiento de llamada.

Cuando se produce una excepción en un procedimiento llamado, el control va inmediatamente a la sección de excepciones de dicho bloque. Una excepción se considera manejada si la sección de excepciones proporciona un manejador para la excepción producida.

Cuando se produce una excepción y ésta se maneja, tiene lugar el siguiente flujo de código:

1. Se produce la excepción.
2. Se transfiere el control al manejador de excepciones.
3. El bloque se termina.
4. El programa o bloque de llamada se sigue ejecutando como si no hubiera ocurrido nada.

Si se ha iniciado una transacción (es decir, si se han ejecutado sentencias de lenguaje de manipulación de datos (DML) antes de ejecutar el procedimiento en el que se ha producido la excepción), la transacción no se verá afectada. Se realiza un rollback de una operación DML si ésta se ha realizado dentro del procedimiento antes de la excepción.

Nota: Puede terminar explícitamente una transacción ejecutando una operación COMMIT o ROLLBACK en la sección de excepciones.

Excepciones Manejadas: Ejemplo

```
CREATE PROCEDURE add_department(
    name VARCHAR2, mgr NUMBER, loc NUMBER) IS
BEGIN
    INSERT INTO DEPARTMENTS (department_id,
        department_name, manager_id, location_id)
    VALUES (DEPARTMENTS_SEQ.NEXTVAL, name, mgr, loc);
    DBMS_OUTPUT.PUT_LINE('Added Dept: ' || name);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Err: adding dept: ' || name);
END;
```

```
CREATE PROCEDURE create_departments IS
BEGIN
    add_department('Media', 100, 1800);
    add_department('Editing', 99, 1800);
    →add_department('Advertising', 101, 1800);
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Excepciones Manejadas: Ejemplo

Los dos procedimientos del ejemplo son los siguientes:

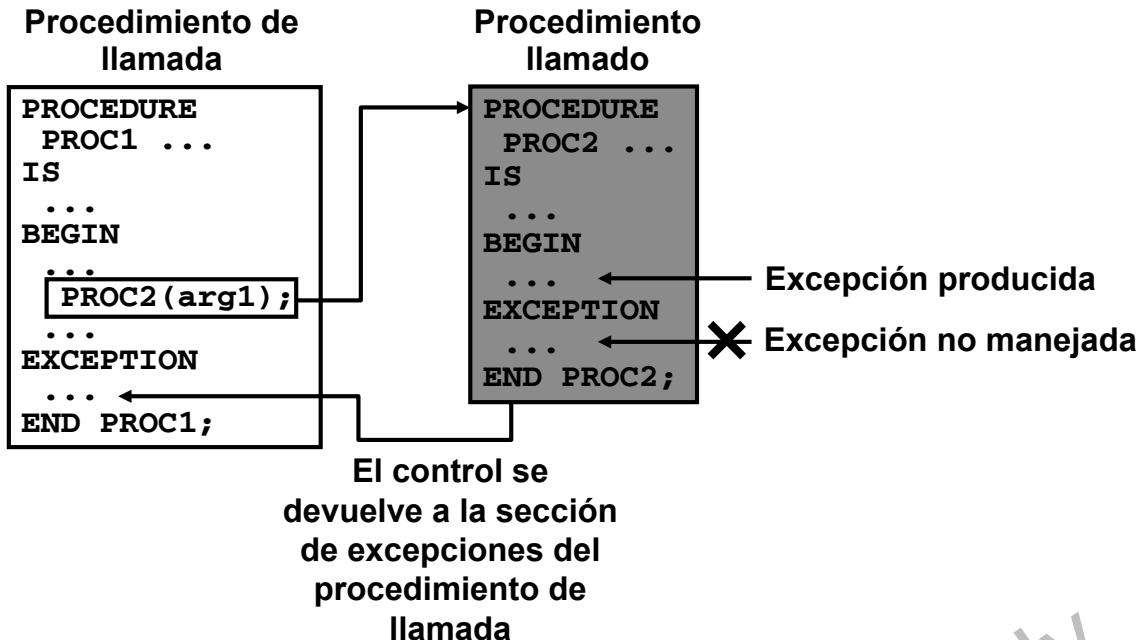
- El procedimiento `add_department` crea un nuevo registro de departamento asignando un nuevo número de departamento de una secuencia Oracle y define los valores de columna `department_name`, `manager_id` y `location_id` con los parámetros `name`, `mgr` y `loc`, respectivamente.
- El procedimiento `create_departments` crea más de un departamento utilizando llamadas al procedimiento `add_department`.

El procedimiento `add_department` recopila todas las excepciones producidas en su propio manejador. Cuando se ejecuta `create_departments`, se genera la siguiente salida:

```
Added Dept: Media
Err: Adding Dept: Editing
Added Dept: Advertising
```

El departamento `Editing` con el valor 99 para `manager_id` no se ha insertado debido a una violación de la restricción de integridad de clave ajena en `manager_id`. Puesto que la excepción se ha manejado en el procedimiento `add_department`, el procedimiento `create_department` se sigue ejecutando. Una consulta de la tabla `DEPARTMENTS` donde el valor de `location_id` es 1800 muestra que se han agregado los registros `Media` y `Advertising`, pero no el registro `Editing`.

Excepciones No Manejadas



ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Excepciones No Manejadas

Tal como se ha descrito, cuando se produce una excepción en un procedimiento llamado, el control va inmediatamente a la sección de excepciones de dicho bloque. Si la sección de excepciones no proporciona un manejador para la excepción producida, ésta no se maneja. Se produce el siguiente flujo de código:

1. Se produce la excepción.
2. El bloque termina porque no existe ningún manejador de excepciones; se realiza un rollback de cualquier operación DML realizada en el procedimiento.
3. La excepción se propaga a la sección de excepciones del procedimiento de llamada. Es decir, el control se devuelve a la sección de excepciones del bloque de llamada, si existe.

Si no se maneja una excepción, se realiza un rollback de todas las sentencias DML del procedimiento de llamada y el procedimiento llamado, así como de los cambios realizados en cualquier variable de host. Las sentencias DML que no se ven afectadas son sentencias que se han ejecutado antes de llamar al código PL/SQL cuyas excepciones no se han manejado.

Excepciones No Manejadas: Ejemplo

```
CREATE PROCEDURE add_department_noex(
    name VARCHAR2, mgr NUMBER, loc NUMBER) IS
BEGIN
    INSERT INTO DEPARTMENTS (department_id,
        department_name, manager_id, location_id)
    VALUES (DEPARTMENTS_SEQ.NEXTVAL, name, mgr, loc);
    DBMS_OUTPUT.PUT_LINE('Added Dept: ' || name);
END;
```

```
CREATE PROCEDURE create_departments_noex IS
BEGIN
    add_department_noex('Media', 100, 1800);
    add_department_noex('Editing', 99, 1800);
    add_department_noex('Advertising', 101, 1800);
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Excepciones No Manejadas: Ejemplo

El ejemplo de código de la transparencia muestra `add_department_noex`, que no tiene una sección de excepciones. En este caso, la excepción se produce cuando se agrega el departamento `Editing`. Debido a la falta de manejo de excepciones en los subprogramas, no se agrega ningún registro de departamento nuevo a la tabla `DEPARTMENTS`. Al ejecutar el procedimiento `create_departments_noex` se produce un resultado similar al siguiente:

```
Added Dept: Media
BEGIN create_departments_noex; END;

*
ERROR at line 1:
ORA-02291: integrity constraint (ORA1.DEPT_MGR_FK)
violated - parent key not
found
ORA-06512: at "ORA1.ADD_DEPARTMENT_NOEX", line 4
ORA-06512: at "ORA1.CREATE_DEPARTMENTS_NOEX", line 4
ORA-06512: at line 1
```

Aunque los resultados muestran que se ha agregado el departamento `Media`, se ha realizado un rollback de la operación ya que la excepción no se ha manejado en ninguno de los subprogramas llamados.

Eliminación de Procedimientos

Puede eliminar un procedimiento que está almacenado en la base de datos.

- **Sintaxis:**

```
DROP PROCEDURE procedure_name
```

- **Ejemplo:**

```
DROP PROCEDURE raise_salary;
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Eliminación de Procedimientos

Cuando ya no necesite un procedimiento almacenado, puede utilizar la sentencia SQL DROP PROCEDURE para eliminarlo.

Nota: Al ejecutar un comando de lenguaje de definición de datos (DDL) como DROP PROCEDURE, tanto si es correcto como si no, se confirma cualquier transacción pendiente de la que no se puede realizar un rollback.

Visualización de Procedimientos en el Diccionario de Datos

La información de los procedimientos PL/SQL se guarda en las siguientes vistas del diccionario de datos:

- **Vista del código de origen en la tabla USER_SOURCE para ver los subprogramas de los que es propietario o la tabla ALL_SOURCE para ver los procedimientos que son propiedad de otros usuarios que le han otorgado el privilegio EXECUTE.**

```
SELECT text
FROM user_source
WHERE name='ADD_DEPARTMENT' and type='PROCEDURE'
ORDER BY line;
```

- **Vista de los nombres de procedimientos en USER_OBJECTS.**

```
SELECT object_name
FROM user_objects
WHERE object_type = 'PROCEDURE';
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Visualización de Procedimientos en el Diccionario de Datos

El código de origen para los subprogramas PL/SQL se almacena en las tablas del diccionario de datos. Los procedimientos PL/SQL compilados correcta o incorrectamente pueden acceder al código de origen. Para ver el código de origen PL/SQL almacenado en el diccionario de datos, ejecute una sentencia SELECT en las siguientes tablas:

- En la tabla USER_SOURCE para mostrar el código PL/SQL del que es propietario.
- En la tabla ALL_SOURCE para mostrar el código PL/SQL para el que el propietario de dicho código de subprograma le ha otorgado el derecho EXECUTE.

El ejemplo de consulta muestra todas las columnas que proporciona la tabla USER_SOURCE:

- La columna TEXT contiene una línea del código de origen PL/SQL.
- La columna NAME contiene el nombre del subprograma en mayúsculas.
- La columna TYPE contiene el tipo de subprograma como PROCEDURE, FUNCTION.
- La columna LINE almacena el número de línea de cada línea del código de origen.

La tabla ALL_SOURCE proporciona una columna OWNER además de las columnas anteriores.

Nota: No puede mostrar el código de origen de los paquetes integrados PL/SQL de Oracle ni PL/SQL cuyo código de origen se ha ajustado mediante una utilidad WRAP. La utilidad WRAP convierte el código de origen PL/SQL en un formato que los humanos no pueden descifrar.

Ventajas de los Subprogramas

- **Mantenimiento sencillo**
- **Integridad y seguridad de datos mejorada**
- **Rendimiento mejorado**
- **Claridad de código mejorada**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ventajas de los Subprogramas

Los procedimientos y las funciones tienen numerosas ventajas gracias a la organización en módulos del código:

- El **mantenimiento sencillo** se consigue porque los subprogramas están ubicados en un solo lugar. Sólo es necesario realizar las modificaciones en un lugar para que se apliquen a varias aplicaciones. Esto minimiza también la realización de un número excesivo de pruebas.
- La **seguridad de datos mejorada** se puede lograr controlando el acceso indirecto a los objetos de la base de datos por parte de usuarios sin privilegios mediante privilegios de seguridad. Los subprogramas ejecutados tienen por defecto los derechos del responsable de la definición. El privilegio de ejecución no permite que un usuario que realice una llamada acceda directamente a los objetos a los que puede acceder el subprograma.
- La **integridad de los datos** se gestiona realizando al mismo tiempo todas las acciones relacionadas o no realizando ninguna.

Ventajas de los Subprogramas (continuación)

- El **rendimiento mejorado** se consigue al reutilizar código PL/SQL analizado que pasa a estar disponible en el área SQL compartida del servidor. Las llamadas posteriores al subprograma evitan tener que analizar el código otra vez. Puesto que el código PL/SQL se analiza durante la compilación, la sobrecarga de análisis de sentencias SQL se evita en tiempo de ejecución. Se puede escribir código para reducir el número de llamadas de red a la base de datos y, por lo tanto, disminuir el tráfico de red.
- La **claridad de código mejorada** se alcanza con el uso de nombres y convenciones adecuadas para describir la acción de las rutinas, la reducción de la necesidad de comentarios y la mejora de la claridad del código.

Oracle Internal & OAI Use Only

Resumen

En esta lección, debe haber aprendido lo siguiente:

- **Escribir un procedimiento para realizar una tarea o una acción**
- **Crear, compilar y guardar procedimientos en la base de datos con el comando SQL CREATE PROCEDURE**
- **Utilizar parámetros para transferir datos del entorno de llamada al procedimiento con tres modos de parámetros distintos: IN (valor por defecto), OUT e IN OUT.**
- **Reconocer el efecto que tiene manejar y no manejar excepciones en transacciones y procedimientos de llamada**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen

Un procedimiento es un subprograma que realiza una acción concreta. Puede compilar y guardar un procedimiento como procedimiento almacenado en la base de datos. Un procedimiento puede devolver cero o más valores mediante sus parámetros al entorno de llamada. Existen tres modos de parámetros: IN, OUT e IN OUT.

Debe ser capaz de manejar y no manejar excepciones y comprender cómo afecta el manejo de excepciones a las transacciones y los procedimientos de llamada. Las excepciones se manejan en la sección de excepciones de un subprograma.

Resumen

- **Eliminar procedimientos de la base de datos con el comando SQL `DROP PROCEDURE`**
- **Organizar en módulos el código de la aplicación utilizando los procedimientos como bloques de construcción**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen (continuación)

Puede modificar y eliminar procedimientos. Los procedimientos son componentes modulares que conforman los bloques de construcción de una aplicación. También puede crear procedimientos de cliente para utilizar con aplicaciones de cliente.

Práctica 1: Visión General

En esta práctica se abordan los siguientes temas:

- **Creación de procedimientos almacenados para:**
 - Insertar nuevas filas en una tabla con los valores de parámetros proporcionados
 - Actualizar los datos de una tabla para las filas que coinciden con los valores de parámetros proporcionados
 - Suprimir filas de una tabla que coinciden con los valores de parámetros proporcionados
 - Consultar una tabla y recuperar datos según los valores de parámetros proporcionados
- **Manejar excepciones en procedimientos**
- **Compilar y llamar a procedimientos**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica 1: Visión General

En esta práctica, se crean procedimientos que emiten comandos DML y de consulta.

Si se producen errores de compilación durante el uso de iSQL*Plus, utilice el comando SHOW ERRORS. El uso del comando SHOW ERRORS se describe en detalle en la lección *Gestión de Subprogramas*.

Si corrige algún error de compilación en iSQL*Plus, hágalo en el archivo de comandos original y no en el buffer y, a continuación, vuelva a ejecutar la nueva versión del archivo. De esta forma, la nueva versión del procedimiento se guardará en el diccionario de datos.

Práctica 1

Nota: Las descripciones de las tablas y los datos de ejemplo aparecen en el apéndice B, "Descripciones de las Tablas y Datos". Haga clic en el botón Save Script para guardar los subprogramas como archivos .sql en el sistema de archivos local.

No olvide activar SERVEROUTPUT si lo ha desactivado previamente.

1. Cree y llame al procedimiento ADD_JOB y tenga en cuenta los resultados.
 - a. Cree un procedimiento denominado ADD_JOB para insertar un nuevo trabajo en la tabla JOBS. Proporcione el identificador y el título del trabajo utilizando dos parámetros.
 - b. Compile el código y llame al procedimiento con IT_DBA como identificador de trabajo y Database Administrator como título. Consulte la tabla JOBS para ver los resultados.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Database Administrator		

- c. Llame al procedimiento de nuevo y transfiera un identificador de trabajo de ST_MAN y un título de Stock Manager. ¿Qué sucede? ¿Por qué?

-
2. Cree un procedimiento denominado UPD_JOB para modificar un trabajo en la tabla JOBS.
 - a. Cree un procedimiento denominado UPD_JOB para actualizar el título. Proporcione el identificador de trabajo y un título nuevo utilizando dos parámetros. Incluya el manejo de excepciones necesario si no se ha producido la actualización.
 - b. Compile el código, llame al procedimiento para cambiar el título del identificador de trabajo IT_DBA a Data Administrator. Consulte la tabla JOBS para ver los resultados.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Data Administrator		

Compruebe también el manejo de excepciones intentando actualizar un trabajo que no existe. (Puede utilizar el identificador de trabajo IT_WEB y el título Web Master.)

3. Cree un procedimiento denominado DEL_JOB para suprimir un trabajo en la tabla JOBS.
 - a. Cree un procedimiento denominado DEL_JOB para suprimir un trabajo. Incluya el manejo de excepciones necesario si no se ha suprimido ningún trabajo.
 - b. Compile el código; llame al procedimiento utilizando el identificador de trabajo IT_DBA. Consulte la tabla JOBS para ver los resultados.

no rows selected

Compruebe también el manejo de excepciones intentando suprimir un trabajo que no existe. (Utilice el identificador de trabajo IT_WEB.) Aparecerá el mensaje que utilizó en la sección manejo de excepciones del procedimiento como salida.

Práctica 1 (continuación)

4. Cree un procedimiento denominado GET_EMPLOYEE para consultar la tabla EMPLOYEES, lo que devuelve el salario y el identificador de trabajo de un empleado cuando se proporciona el identificador de empleado.
 - a. Cree un procedimiento que devuelva un valor de las columnas SALARY y JOB_ID para el identificador de empleado especificado. Compile el código y elimine los errores de sintaxis.
 - b. Ejecute el procedimiento utilizando las variables del host para los dos parámetros OUT: uno para el salario y el otro para el identificador de trabajo. Muestre el salario y el identificador de trabajo para el identificador de empleado 120.

SALARY
8000
JOB
ST_MAN

- c. Llame al procedimiento de nuevo y transfiera un EMPLOYEE_ID de 300. ¿Qué sucede? ¿Por qué?

Oracle Internal & OAI Use Only

Creación de Funciones Almacenadas

2

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- **Describir los usos de las funciones**
- **Crear funciones almacenadas**
- **Llamar a una función**
- **Eliminar una función**
- **Diferenciar entre un procedimiento y una función**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetivos

En esta lección, aprenderá a crear y llamar a funciones.

Visión General de las Funciones Almacenadas

Una función:

- Es un bloque PL/SQL con nombre que devuelve un valor**
- Se puede almacenar en la base de datos como objeto de esquema para una ejecución repetida**
- Se llama como parte de una expresión o se utiliza para proporcionar un valor de parámetro**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Visión General de las Funciones Almacenadas

Una función es un bloque PL/SQL con nombre que puede aceptar parámetros, que se puede llamar y devolver un valor. En general, una función se utiliza para calcular un valor. Las funciones y los procedimientos se estructuran de forma similar. Una función debe devolver un valor al entorno de llamada, mientras que un procedimiento devuelve cero o más valores a su entorno de llamada. Al igual que los procedimientos, las funciones tienen una cabecera, una sección de declaraciones, una sección ejecutable y una sección de manejo de excepciones opcional. Las funciones deben tener una cláusula RETURN en la cabecera y al menos una sentencia RETURN en la sección ejecutable.

Las funciones se pueden almacenar en la base de datos como objetos de esquema para una ejecución repetida. Una función que se almacena en la base de datos se denomina función almacenada. Las funciones también se pueden crear en aplicaciones de cliente.

Las funciones fomentan la capacidad de reutilización y mantenimiento. Cuando se validan, se pueden utilizar en cualquier número de aplicaciones. Si los requisitos de procesamiento cambian, sólo es necesario actualizar la función.

También se puede llamar a una función como parte de una expresión SQL o de una expresión PL/SQL. En el contexto de una expresión SQL, una función debe obedecer a reglas específicas para controlar los efectos secundarios. En una expresión PL/SQL, el identificador de función actúa como una variable cuyo valor depende de los parámetros transferidos.

Sintaxis para Crear Funciones

El bloque PL/SQL debe tener al menos una sentencia RETURN.

```
CREATE [OR REPLACE] FUNCTION function_name
[ (parameter1 [mode1] datatype1, ... ) ]
RETURN datatype IS|AS
[ local_variable_declarations; ... ]
BEGIN
    -- actions;
    RETURN expression;
END [function_name];
```

→ Bloque PL/SQL

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Sintaxis para Crear Funciones

Una función es un bloque PL/SQL que devuelve un valor. Se debe proporcionar una sentencia RETURN para devolver un valor con un tipo de dato que sea coherente con la declaración de funciones.

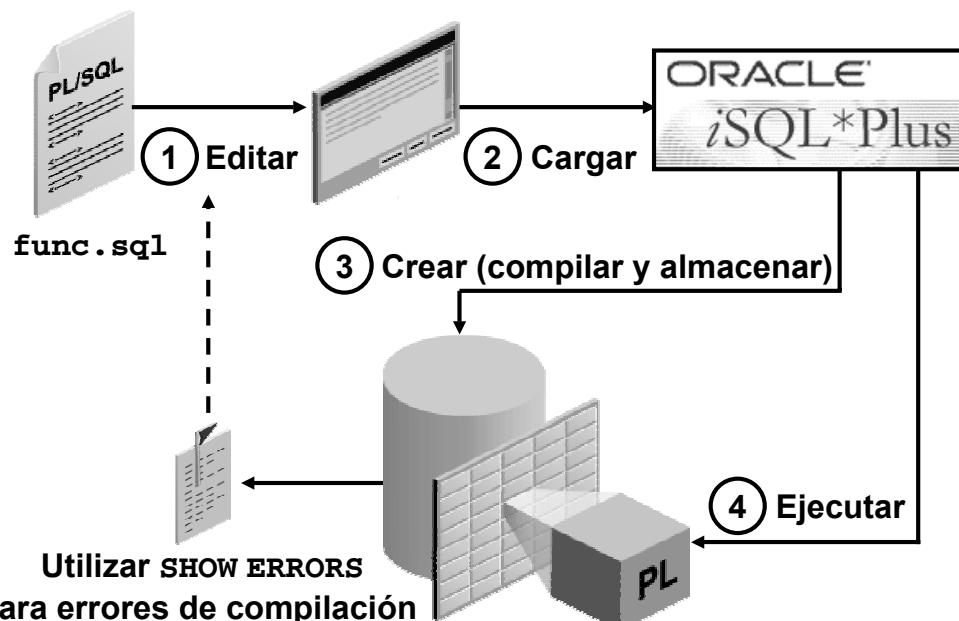
Se crean nuevas funciones con la sentencia CREATE FUNCTION, que puede declarar una lista de parámetros, debe devolver un valor y debe definir las acciones que debe realizar el bloque PL/SQL estándar.

Debe tener en cuenta los siguientes puntos sobre la sentencia CREATE FUNCTION:

- La opción REPLACE indica que si la función existe, se borrará y se sustituirá por la nueva versión creada por la sentencia.
- El tipo de dato RETURN no debe incluir una especificación de tamaño.
- El bloque PL/SQL empieza por BEGIN después de la declaración de todas las variables locales y termina en END, seguido opcionalmente por *function_name*.
- Debe existir al menos una sentencia RETURN *expression*.
- No se puede hacer referencia a variables de host o ligadas en el bloque PL/SQL de una función almacenada.

Nota: Aunque los modos de parámetro OUT e IN OUT se pueden utilizar con funciones, no resulta una práctica aconsejable desde el punto de vista de la programación. Sin embargo, si necesita devolver más de un valor de una función, considere la devolución de los valores en una estructura de datos compuesta, como un registro PL/SQL o una tabla PL/SQL.

Desarrollo de Funciones



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Desarrollo de Funciones Almacenadas

El diagrama ilustra los pasos básicos del desarrollo de una función almacenada. Para desarrollar una función almacenada, realice los siguientes pasos:

1. Cree un archivo utilizando su editor de códigos o texto favorito para editar la sintaxis de la función y guardando el código en un archivo normalmente con una extensión .sql.
2. Cargue el código de la función del archivo en el buffer con iSQL*Plus como entorno de desarrollo PL/SQL.
3. Ejecute la sentencia CREATE FUNCTION para compilar y almacenar la función en la base de datos.
4. Después de terminar la compilación correctamente, llame a la función desde una aplicación o entorno PL/SQL.

Devolución de Valores

- Agregue una cláusula RETURN con el tipo de dato en la cabecera de la función.
- Incluya una sentencia RETURN en la sección ejecutable.

Se permiten varias sentencias RETURN en una función (normalmente dentro de una sentencia IF). Sólo se ejecuta una sentencia RETURN ya que, una vez que se devuelve el valor, se interrumpe el procesamiento del bloque.

Utilice los comandos de iSQL*Plus SHOW ERRORS o SHOW ERRORS FUNCTION function_name para ver los errores de compilación.

Función Almacenada: Ejemplo

- **Crear la función:**

```
CREATE OR REPLACE FUNCTION get_sal
  (id employees.employee_id%TYPE) RETURN NUMBER IS
    sal employees.salary%TYPE := 0;
BEGIN
  SELECT salary
  INTO   sal
  FROM   employees
  WHERE  employee_id = id;
  RETURN sal;
END get_sal;
/
```

- **Llamar a la función como expresión o como valor de parámetro:**

```
EXECUTE dbms_output.put_line(get_sal(100))
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Función Almacenada: Ejemplo

La función `get_sal` se crea con un único parámetro de entrada y devuelve el salario en forma de número. Ejecute el comando tal como se muestra o guárdelo en un archivo de comandos y ejecute éste para crear la función `get_sal`.

La función `get_sal` sigue la práctica de programación común de utilizar una única sentencia `RETURN` que devuelve un valor asignado a una variable local. Si la función tiene una sección de excepciones, también puede contener una sentencia `RETURN`.

Llame a la función como parte de una expresión PL/SQL, ya que la función devolverá un valor al entorno de llamada. El segundo recuadro de código utiliza el comando `EXECUTE` de *i*SQL*Plus para llamar al procedimiento `DBMS_OUTPUT.PUT_LINE` cuyo argumento es el valor de retorno de la función `get_sal`. En este caso, `get_sal` se llama primero para calcular el salario del empleado con el identificador 100. El valor del salario devuelto se proporciona como el valor del parámetro `DBMS_OUTPUT.PUT_LINE`, que muestra el resultado (si ha ejecutado `SET SERVEROUTPUT ON`).

Nota: Una función debe devolver siempre un valor. El ejemplo no devuelve un valor si no se encuentra una fila para un valor `id` determinado. Lo ideal es crear un manejador de excepciones para que también devuelva un valor.

Modos de Ejecutar Funciones

- Llamarlas como parte de una expresión PL/SQL
 - Utilizar una variable de host para obtener el resultado

```
VARIABLE salary NUMBER  
EXECUTE :salary := get_sal(100)
```

- Utilizar una variable local para obtener el resultado

```
DECLARE sal employees.salary%type;  
BEGIN  
    sal := get_sal(100); ...  
END;
```

- Utilizarlas como parámetro de otro subprograma

```
EXECUTE dbms_output.put_line(get_sal(100))
```

- Utilizarlas en una sentencia SQL (opción sujeta a restricciones)

```
SELECT job_id, get_sal(employee_id) FROM employees;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Modos de Ejecutar Funciones

Si las funciones se diseñan de forma coherente, se pueden convertir en construcciones potentes.

Se puede llamar a las funciones de las siguientes maneras:

- **Como parte de expresiones PL/SQL:** Puede utilizar variables de host o locales que contengan el valor devuelto de una función. El primer ejemplo de la transparencia utiliza una variable de host y el segundo ejemplo utiliza una variable local en un bloque anónimo.
- **Como parámetro de otro subprograma:** El tercer ejemplo de la transparencia ilustra este caso. La función `get_sal` con todos sus argumentos se anida en el parámetro que necesita el procedimiento `DBMS_OUTPUT.PUT_LINE`. Esto procede del concepto de anidar funciones que se describe en el curso *Base de Datos Oracle 10g: Conceptos Fundamentales de SQL I*.
- **Como expresión de una sentencia SQL:** El último ejemplo muestra cómo una función se puede utilizar como función de una sola fila en una sentencia SQL.

Nota: Las ventajas y restricciones que se aplican a las funciones cuando se utilizan en una sentencia SQL se describen en las siguientes páginas.

Ventajas del Uso de Funciones Definidas por el Usuario en Sentencias SQL

- **Pueden ampliar SQL cuando las actividades sean demasiado complejas, demasiado extrañas o cuando no estén disponibles con SQL**
- **Pueden aumentar la eficacia si se utilizan en la cláusula WHERE para filtrar datos, en lugar de filtrar los datos en la aplicación**
- **Pueden manipular valores de datos**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ventajas del Uso de Funciones Definidas por el Usuario en Sentencias SQL

Las sentencias SQL pueden hacer referencia a funciones PL/SQL definidas por el usuario en cualquier lugar en el que se permitan las expresiones SQL. Por ejemplo, una función definida por el usuario se puede utilizar en cualquier lugar en el que se pueda colocar una función SQL integrada, como UPPER ().

Ventajas

- Permite realizar cálculos demasiado complejos, extraños o que no estén disponibles con SQL.
- Aumenta la independencia de los datos mediante el procesamiento de un análisis de datos complejo en el servidor de Oracle, en lugar de mediante la recuperación de los datos en una aplicación.
- Aumenta la eficacia de las consultas realizando funciones en la consulta y no en la aplicación.
- Manipula nuevos tipos de dato (por ejemplo, latitud y longitud) con la codificación de cadenas de caracteres y el uso de funciones para trabajar con las cadenas.

Funciones en Expresiones SQL: Ejemplo

```
CREATE OR REPLACE FUNCTION tax(value IN NUMBER)
  RETURN NUMBER IS
BEGIN
  RETURN (value * 0.08);
END tax;
/
SELECT employee_id, last_name, salary, tax(salary)
FROM   employees
WHERE  department_id = 100;
```

Function created.

EMPLOYEE_ID	LAST_NAME	SALARY	TAX(SALARY)
108	Greenberg	12000	960
109	Faviet	9000	720
110	Chen	8200	656
111	Sciarras	7700	616
112	Urman	7800	624
113	Popp	6900	552

6 rows selected.

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Funciones en Expresiones SQL: Ejemplo

El ejemplo de la transparencia muestra cómo crear una función `tax` para calcular el impuesto sobre la renta. La función acepta un parámetro NUMBER y devuelve el impuesto sobre la renta calculado sobre una única tasa de interés fijo del 8%.

En *iSQL*Plus*, la función `tax` se llama como una expresión de la cláusula SELECT junto con el identificador de empleado, el apellido y el salario de los empleados de un departamento con identificador 100. El resultado devuelto de la función `tax` se muestra con la salida normal de la consulta.

Ubicaciones de Funciones Definidas por el Usuario

Las funciones definidas por el usuario actúan como funciones integradas de una sola fila y se pueden utilizar en:

- **La cláusula o lista SELECT de una consulta**
- **Expresiones condicionales de las cláusulas WHERE y HAVING**
- **Las cláusulas CONNECT BY, START WITH, ORDER BY y GROUP BY de una consulta**
- **La cláusula VALUES de la sentencia INSERT**
- **La cláusula SET de la sentencia UPDATE**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ubicaciones de Funciones Definidas por el Usuario

Se puede llamar a una función PL/SQL definida por el usuario desde cualquier expresión SQL donde se pueda llamar a una función integrada de una sola fila.

Ejemplo:

```
SELECT employee_id, tax(salary)
  FROM employees
 WHERE tax(salary) > (SELECT MAX(tax(salary))
                           FROM employees
                          WHERE department_id = 30)
 ORDER BY tax(salary) DESC;
```

EMPLOYEE_ID	TAX(SALARY)
100	1920
101	1360
102	1360
145	1120
146	1080
201	1040
...	

10 rows selected.

Restricciones para Llamar a Funciones desde Expresiones SQL

- **Las funciones que se pueden llamar desde expresiones SQL deben:**
 - Estar almacenadas en la base de datos
 - Aceptar sólo parámetros **IN** con tipos de dato SQL válidos y no tipos específicos de PL/SQL
 - Devolver tipos de dato SQL válidos y no tipos específicos de PL/SQL
- **Al llamar a funciones en sentencias SQL:**
 - Los parámetros se deben especificar con notación posicional
 - El usuario debe ser propietario de la función o tener el privilegio **EXECUTE**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Restricciones para Llamar a Funciones desde Expresiones SQL

Las funciones PL/SQL definidas por el usuario que se pueden llamar desde expresiones SQL deben cumplir los siguientes requisitos:

- La función debe estar almacenada en la base de datos.
- Los parámetros de la función deben ser tipos de dato SQL válidos y de entrada sólo.
- Las funciones deben devolver tipos de dato SQL válidos. No pueden ser tipos de dato específicos de PL/SQL como BOOLEAN, RECORD o TABLE. La misma restricción se aplica a los parámetros de la función.

La siguiente restricción se aplica al llamar a una función en una sentencia SQL:

- Los parámetros deben utilizar la notación posicional. La notación con nombre no está soportada.
- El usuario debe ser propietario de la función o tener el privilegio **EXECUTE**.

Otras restricciones aplicables a las funciones definidas por el usuario son las siguientes:

- No se pueden llamar desde la cláusula de restricción CHECK de una sentencia CREATE TABLE o ALTER TABLE.
- No se pueden utilizar para especificar un valor por defecto para una columna.

Nota: Sólo se puede llamar a funciones almacenadas desde sentencias SQL. No se puede llamar a procedimientos almacenados a menos que se haga desde una función que cumpla todos los requisitos anteriores.

Control de Efectos Secundarios al Llamar a Funciones desde Expresiones SQL

Las funciones que se llaman desde:

- **Una sentencia SELECT no pueden contener sentencias DML**
- **Una sentencia UPDATE o DELETE de una tabla T no pueden realizar consultas o contener DML en la misma tabla T**
- **Las sentencias SQL no pueden terminar transacciones (es decir, no pueden ejecutar operaciones COMMIT o ROLLBACK)**

Nota: Tampoco se permiten en la función las llamadas a subprogramas que violen estas restricciones.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Control de Efectos Secundarios al Llamar a Funciones desde Expresiones SQL

Para ejecutar una sentencia SQL que llame a una función almacenada, el servidor de Oracle debe saber si la función está libre de efectos secundarios específicos. Los efectos secundarios son cambios inaceptables realizados en tablas de bases de datos.

Se aplican restricciones adicionales cuando se llama a una función desde expresiones de sentencias SQL. En concreto, cuando se llama a una función desde:

- Una sentencia SELECT o una sentencia UPDATE o DELETE paralela, la función no puede modificar ninguna tabla de la base de datos.
- Una sentencia UPDATE o DELETE, la función no puede realizar consultas ni modificar ninguna tabla de la base de datos modificada por dicha sentencia.
- Una sentencia SELECT, INSERT, UPDATE o DELETE, la función no se puede ejecutar directa o indirectamente a través de otro subprograma o de sentencias de control de transacciones SQL como:
 - Una sentencia COMMIT o ROLLBACK
 - Una sentencia de control de sesión (como SET ROLE)
 - Una sentencia de control del sistema (como ALTER SYSTEM)
 - Cualquier sentencia DDL (como CREATE), ya que van seguidas de una confirmación automática.

Restricciones para Llamar a Funciones desde SQL: Ejemplo

```
CREATE OR REPLACE FUNCTION dml_call_sql(sal NUMBER)
  RETURN NUMBER IS
BEGIN
  INSERT INTO employees(employee_id, last_name,
                        email, hire_date, job_id, salary)
  VALUES(1, 'Frost', 'jfrost@company.com',
         SYSDATE, 'SA_MAN', sal);
  RETURN (sal + 100);
END;

UPDATE employees
  SET salary = dml_call_sql(2000)
 WHERE employee_id = 170;

UPDATE employees SET salary = dml_call_sql(2000)
*
ERROR at line 1:
ORA-04091: table PLSQL.EMPLOYEES is mutating,
trigger/function may not see it
ORA-06512: at "PLSQL.DML_CALL_SQL", line 4
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Restricciones para Llamar a Funciones desde SQL: Ejemplo

La función `dml_call_sql` de la transparencia contiene una sentencia `INSERT` que inserta un nuevo registro en la tabla `EMPLOYEES` y devuelve el valor de entrada del salario incrementado en 100. Esta función se llama desde la sentencia `UPDATE`, que modifica el salario del empleado 170 a la cantidad devuelta desde la función. La sentencia `UPDATE` genera un error que indica que la tabla está mutando (es decir, los cambios ya están en curso en la misma tabla). En el siguiente ejemplo, la función `query_call_sql` consulta la columna `SALARY` de la tabla `EMPLOYEES`:

```
CREATE OR REPLACE FUNCTION query_call_sql(a NUMBER)
  RETURN NUMBER IS
  s NUMBER;
BEGIN
  SELECT salary INTO s FROM employees
  WHERE employee_id = 170;
  RETURN (s + a);
END;
```

Cuando se llama desde la siguiente sentencia `UPDATE`, devuelve un mensaje de error similar al mensaje de error que se muestra en la transparencia:

```
UPDATE employees SET salary = query_call_sql(100)
WHERE employee_id = 170;
```

Eliminación de Funciones

Eliminación de una función almacenada:

- Puede borrar una función almacenada utilizando la siguiente sintaxis:

```
DROP FUNCTION function_name
```

Ejemplo:

```
DROP FUNCTION get_sal;
```

- Todos los privilegios que se otorgan para una función se revocan al borrar la función.
- La sintaxis CREATE OR REPLACE equivale a borrar una función y volver a crearla. Los privilegios otorgados para la función siguen siendo los mismos cuando se utiliza esta sintaxis.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Eliminación de Funciones

Cuando ya no se necesita una función almacenada, puede utilizar una sentencia SQL en iSQL*Plus para borrarla. Para eliminar una función almacenada con iSQL*Plus, ejecute el comando SQL DROP FUNCTION.

CREATE OR REPLACE frente a DROP y CREATE

La cláusula REPLACE de la sintaxis CREATE OR REPLACE equivale a borrar una función y volver a crearla. Cuando utiliza la sintaxis CREATE OR REPLACE, los privilegios otorgados a otros usuarios para este objeto siguen siendo los mismos. Cuando borra una función y, a continuación, vuelve a crearla, todos los privilegios otorgados para esta función se revocan automáticamente.

Visualización de Funciones en el Diccionario de Datos

La información de las funciones PL/SQL se almacena en las siguientes vistas del diccionario de datos de Oracle:

- **Puede ver el código de origen en la tabla `USER_SOURCE` para consultar los subprogramas de los que es propietario o la tabla `ALL_SOURCE` para ver las funciones que son propiedad de otros usuarios que le han otorgado el privilegio `EXECUTE`.**

```
SELECT text
  FROM user_source
 WHERE type = 'FUNCTION'
 ORDER BY line;
```

- **Puede ver los nombres de las funciones con `USER_OBJECTS`.**

```
SELECT object_name
  FROM user_objects
 WHERE object_type = 'FUNCTION';
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Visualización de Funciones en el Diccionario de Datos

El código de origen para las funciones PL/SQL se almacena en las tablas del diccionario de datos. Las funciones PL/SQL compiladas correcta o incorrectamente pueden acceder al código de origen. Para ver el código de la función PL/SQL almacenado en el diccionario de datos, ejecute una sentencia SELECT en las siguientes tablas en las que el valor de la columna TYPE es 'FUNCTION':

- En la tabla `USER_SOURCE` para mostrar el código PL/SQL del que es propietario.
- En la tabla `ALL_SOURCE` para mostrar el código PL/SQL para el que el propietario de dicho código de subprograma le ha otorgado el derecho `EXECUTE`.

El primer ejemplo de consulta muestra cómo visualizar el código de origen de todas las funciones del esquema. La segunda consulta, que utiliza la vista `USER_OBJECTS` del diccionario de datos, muestra los nombres de todas las funciones de las que se es propietario.

Procedimientos frente a Funciones

Procedimientos	Funciones
Se ejecutan como una sentencia PL/SQL	Se llaman como parte de una expresión
No contienen una cláusula RETURN en la cabecera	Deben contener una cláusula RETURN en la cabecera
Pueden devolver valores (si existen) en parámetros de salida	Deben devolver un único valor
Pueden contener una sentencia RETURN sin valores	Deben contener al menos una sentencia RETURN

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Diferencias entre Procedimientos y Funciones

Un procedimiento se crea para almacenar una serie de acciones para su posterior ejecución.

Un procedimiento puede contener cero o más parámetros que se pueden transferir a y desde el entorno de llamada, pero no tiene que devolver un valor. Un procedimiento puede llamar a una función para ayudar con sus acciones.

Nota: Es mejor que un procedimiento que contiene un único parámetro OUT se vuelva a escribir como una función que devuelve el valor.

Las funciones se crean cuando se desea calcular un valor que se debe devolver al entorno de llamada. Una función puede contener cero o más parámetros que se transfieren desde el entorno de llamada. Las funciones normalmente devuelven sólo un valor y éste se devuelve mediante una sentencia RETURN. Las funciones utilizadas en sentencias SQL no deben utilizar parámetros de modos OUT o IN OUT. Aunque se puede utilizar una función con parámetros de salida en un bloque o procedimiento PL/SQL, no se puede utilizar en sentencias SQL.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- **Escribir una función PL/SQL para calcular y devolver un valor con la sentencia SQL CREATE FUNCTION**
- **Llamar a una función como parte de una expresión PL/SQL**
- **Utilizar funciones PL/SQL almacenadas en sentencias SQL**
- **Eliminar una función de la base de datos con la sentencia SQL DROP FUNCTION**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen

Una función es un bloque PL/SQL con nombre que debe devolver un valor. Normalmente, una función se crea para calcular y devolver un valor, y un procedimiento, para realizar una acción.

Se puede crear y borrar una función.

Una función se llama como parte de una expresión.

Práctica 2: Visión General

En esta práctica se abordan los siguientes temas:

- Creación de funciones almacenadas
 - Para realizar consultas en una tabla de la base de datos y devolver valores concretos
 - Para utilizar en sentencias SQL
 - Para insertar una nueva fila, con los valores de parámetro especificados, en una tabla de la base de datos
 - Con valores de parámetro por defecto
- Llamada a una función almacenada desde una sentencia SQL
- Llamada a una función almacenada desde un procedimiento almacenado

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica 2: Visión General

Si se producen errores de compilación durante el uso de *iSQL*Plus*, utilice el comando SHOW ERRORS.

Si corrige algún error de compilación en *iSQL*Plus*, hágalo en el archivo de comandos original y no en el buffer y, a continuación, vuelva a ejecutar la nueva versión del archivo. De esta forma, la nueva versión de la unidad de programa se guardará en el diccionario de datos.

Práctica 2

1. Cree y llame a la función GET_JOB para devolver un título.
 - a. Cree y compile la función denominada GET_JOB para devolver un título.
 - b. Cree una variable de host VARCHAR2 denominada TITLE, que permita una longitud de 35 caracteres. Llame a la función con identificador de trabajo SA REP para que devuelva el valor de la variable del host. Imprima la variable del host para ver el resultado.

TITLE
Sales Representative

2. Cree una función denominada GET_ANNUAL_COMP para devolver el salario anual de un empleado calculado a partir del salario mensual y la comisión transferidos como parámetros.
 - a. Desarrolle y almacene la función GET_ANNUAL_COMP, aceptando valores de parámetros para salario mensual y comisión. Uno o ambos valores transferidos pueden ser NULL, pero la función deberá devolver un salario anual no NULL. Utilice la siguiente fórmula básica para calcular el salario anual:
$$(\text{salary} * 12) + (\text{commission_pct} * \text{salary} * 12)$$
 - b. Utilice la función en una sentencia SELECT en la tabla EMPLOYEES para los empleados del departamento 30.

EMPLOYEE_ID	LAST_NAME	Annual Compensation
114	Raphaely	132000
115	Khoo	37200
116	Baida	34800
117	Tobias	33600
118	Himuro	31200
119	Colmenares	30000

6 rows selected.

3. Cree un procedimiento, ADD_EMPLOYEE, para insertar un nuevo empleado en la tabla EMPLOYEES. El procedimiento llamará a una función VALID_DEPTID para comprobar si el identificador de departamento especificado para el nuevo empleado existe en la tabla DEPARTMENTS.
 - a. Cree una función VALID_DEPTID para validar el identificador de departamento especificado y devolver un valor BOOLEAN de TRUE si existe el departamento.
 - b. Cree el procedimiento ADD_EMPLOYEE para agregar un empleado a la tabla EMPLOYEES. La fila se agregará a la tabla EMPLOYEES si la función VALID_DEPTID devuelve TRUE; de lo contrario, alertará al usuario con un mensaje adecuado. Proporcione los siguientes parámetros (con los valores por defecto especificados entre paréntesis): first_name, last_name, email, job (SA REP), mgr (145), sal (1000), comm (0) y deptid (30). Utilice la secuencia EMPLOYEES_SEQ para definir la columna employee_id y definir hire_date en TRUNC (SYSDATE).

- c. Llame a ADD_EMPLOYEE para el nombre Jane Harris del departamento 15, dejando otros parámetros con los valores por defecto. ¿Cuál es el resultado?
- d. Agregue otro empleado llamado Joe Harris en el departamento 80, dejando los parámetros restantes con sus valores por defecto. ¿Cuál es el resultado?

Oracle Internal & OAI Use Only

Creación de Paquetes

3

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- **Describir paquetes y enumerar sus componentes**
- **Crear un paquete para agrupar las variables, cursores, constantes, excepciones, procedimientos y funciones relacionadas**
- **Designar una construcción de paquetes como pública o privada**
- **Llamar a una construcción de paquetes**
- **Describir el uso de un paquete sin cuerpo**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

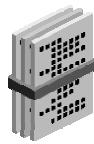
Objetivos

En esta lección, aprenderá lo que es un paquete y cuáles son sus componentes. También aprenderá a crear y a utilizar paquetes.

Paquetes PL/SQL: Visión General

Paquetes PL/SQL:

- **Agrupan componentes relacionados de forma lógica:**
 - Tipos PL/SQL
 - Variables, estructuras de datos y excepciones
 - Subprogramas: procedimientos y funciones
- **Constan de dos partes:**
 - Una especificación
 - Un cuerpo
- **Activan el servidor de Oracle para que lea varios objetos en memoria a la vez**



ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Paquetes PL/SQL: Visión General

Los paquetes PL/SQL le permiten agrupar en un contenedor tipos PL/SQL, variables, estructuras de datos, excepciones y subprogramas relacionados. Por ejemplo, un paquete de recursos humanos puede contener procedimientos de contratación y despido, funciones de comisiones e incentivos y variables de desgravación de impuestos.

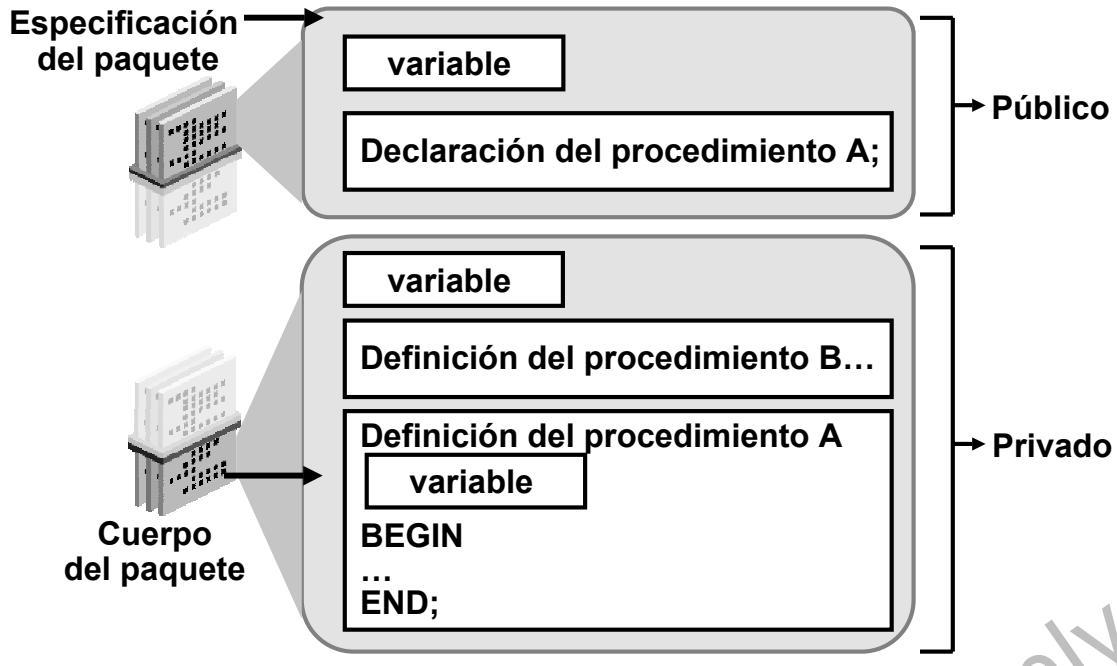
Un paquete consta normalmente de dos partes almacenadas por separado en la base de datos:

- Una especificación
- Un cuerpo (opcional)

El paquete en sí no se puede llamar, parametrizar ni anidar. Después de escribirlo y compilarlo, el contenido se puede compartir con numerosas aplicaciones.

Cuando se hace referencia por primera vez a una construcción de paquete PL/SQL, se carga en la memoria el paquete completo. Para acceder posteriormente a otras construcciones del mismo paquete no es necesaria una entrada/salida (E/S) en disco.

Componentes de un Paquete PL/SQL



Copyright © 2004, Oracle. Todos los Derechos Reservados.

Componentes de un Paquete PL/SQL

Un paquete tiene dos partes:

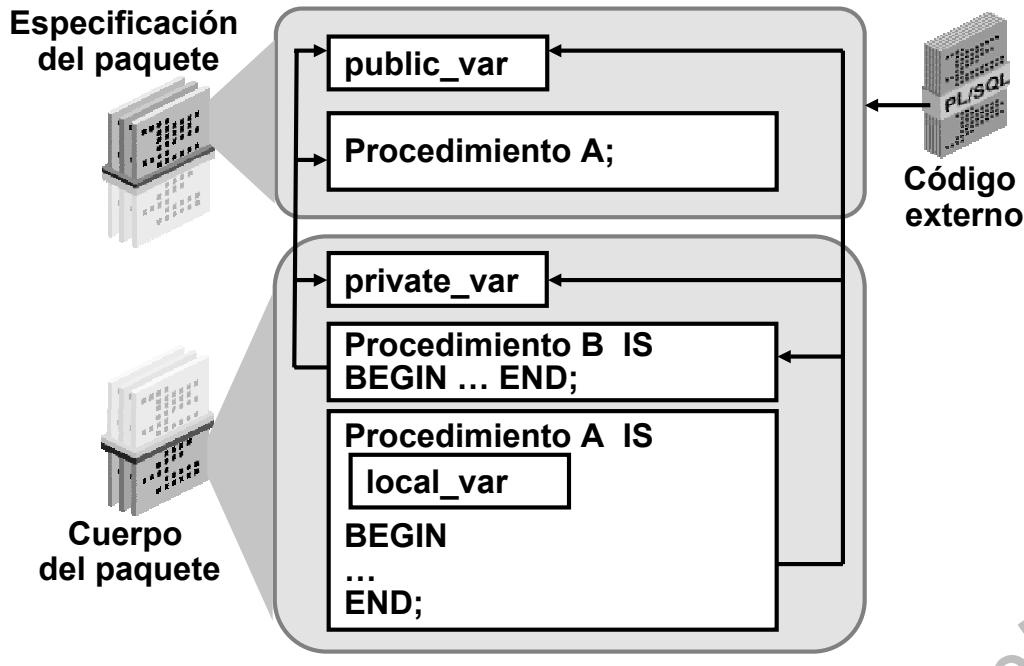
- La **especificación del paquete** es la interfaz para las aplicaciones. Declara los tipos, variables, constantes, excepciones, cursoros y subprogramas públicos disponibles para su uso. La especificación del paquete puede incluir también PRAGMAS, que son directivas para el compilador.
- El **cuerpo del paquete** define sus propios subprogramas y debe implementar completamente los subprogramas declarados en la parte de la especificación. El cuerpo del paquete también puede definir construcciones PL/SQL, como variables de tipos, constantes, excepciones y cursoros.

Los componentes públicos se declaran en la especificación del paquete. La especificación define una interfaz de programación de aplicaciones (API) pública para usuarios de funciones y funcionalidad de paquetes. Es decir, se puede hacer referencia a los componentes públicos desde cualquier entorno de servidor de Oracle que esté fuera del paquete.

Los componentes privados se colocan en el cuerpo del paquete y sólo pueden hacer referencia a ellos otras construcciones del mismo cuerpo del paquete. Los componentes privados pueden hacer referencia a los componentes públicos del paquete.

Nota: Si una especificación de paquete no contiene declaraciones de subprograma, no es necesario un cuerpo de paquete.

Visibilidad de Componentes de Paquete



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Visibilidad de Componentes de Paquete

La *visibilidad* de un componente describe si éste se puede ver, es decir, si otros componentes u objetos hacen referencia a él y lo utilizan. La visibilidad de los componentes depende de si se han declarado local o globalmente.

Los componentes locales se pueden ver dentro de la estructura en la que se han declarado, por ejemplo:

- Se puede hacer referencia a las variables definidas en un subprograma dentro de dicho subprograma y que éstas no estén visibles para los componentes externos; por ejemplo, `local_var` se puede utilizar en el procedimiento A.
- Otros componentes del mismo cuerpo del paquete pueden hacer referencia a las variables de paquetes privadas, que se declaran en el cuerpo de un paquete. No están visibles para los subprogramas u objetos que están fuera del paquete. Por ejemplo, `private_var` se puede utilizar en los procedimientos A y B dentro del cuerpo del paquete, pero no fuera del paquete.

Los componentes que se declaran globalmente son visibles interna y externamente en el paquete, de esta manera:

- Se puede hacer referencia y modificar una variable pública, que se declara en una especificación de paquete, fuera del paquete (por ejemplo, se puede hacer referencia a `public_var` externamente).

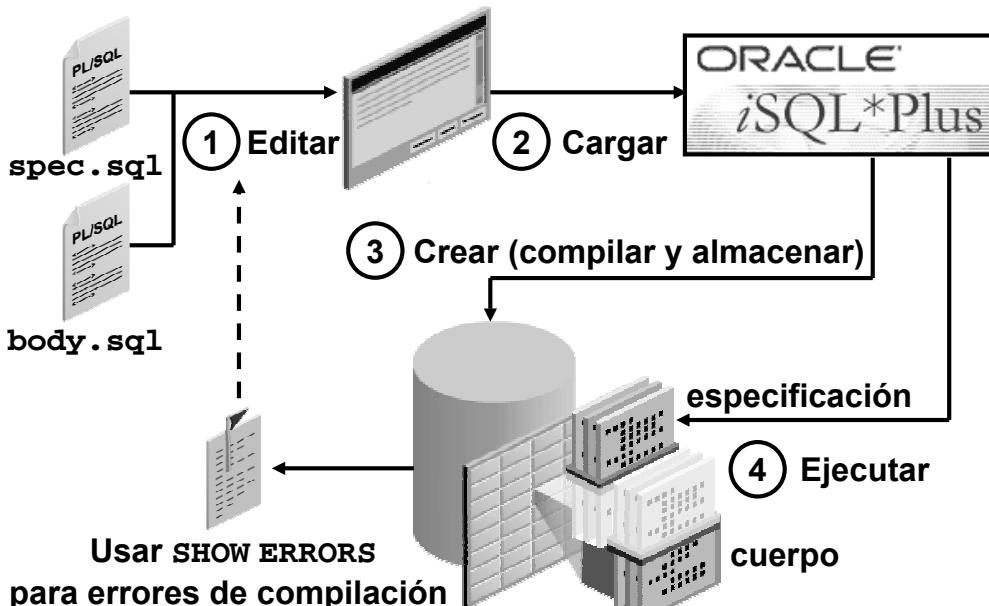
Visibilidad de Componentes de Paquete (continuación)

- Se puede llamar a un subprograma de paquete de la especificación desde orígenes de código externo (por ejemplo, se puede llamar al procedimiento A desde un entorno situado fuera del paquete).

Nota: Se puede llamar a los subprogramas privados, como el procedimiento B, sólo con subprogramas públicos, como el procedimiento A, u otras construcciones de paquete privadas.

Oracle Internal & OAI Use Only

Desarrollo de Paquetes PL/SQL



Copyright © 2004, Oracle. Todos los Derechos Reservados.

ORACLE

Desarrollo de Paquetes PL/SQL

Para desarrollar un paquete, realice los siguientes pasos:

1. Edite el texto para la especificación utilizando la sentencia CREATE PACKAGE dentro de un archivo de comandos SQL. Edite el texto para el cuerpo (si es necesario, consulte las instrucciones que se muestran a continuación) utilizando la sentencia CREATE PACKAGE BODY dentro de un archivo de comandos SQL.
2. Cargue los archivos de comandos en una herramienta como *iSQL*Plus*.
3. Ejecute los archivos de comandos para crear (es decir, para compilar y almacenar) el paquete y el cuerpo del paquete en la base de datos.
4. Ejecute cualquier construcción pública dentro de la especificación de paquete desde un entorno de servidor de Oracle.

Instrucciones para el Desarrollo de Paquetes

- Guarde el texto para la especificación del paquete y para el cuerpo del paquete en dos archivos de comandos distintos para facilitar las modificaciones en el paquete o en el cuerpo del paquete.
- Puede existir una especificación de paquete sin cuerpo; es decir, cuando la especificación de paquete no declara subprogramas, no es necesario un cuerpo. Sin embargo, no puede existir un cuerpo de paquete sin especificación de paquete.

Nota: El servidor de Oracle almacena la especificación y el cuerpo de un paquete por separado. Esto permite cambiar la implementación de una construcción de programa en el cuerpo del paquete sin invalidar otros objetos de esquema que llaman o hacen referencia a la construcción de programa.

Creación de la Especificación del Paquete

Sintaxis:

```
CREATE [OR REPLACE] PACKAGE package_name IS | AS  
    public type and variable declarations  
    subprogram specifications  
END [package_name];
```

- La opción OR REPLACE borra y vuelve a crear la especificación del paquete.
- Las variables declaradas en la especificación del paquete se inicializan con el valor NULL como valor por defecto.
- Todas las construcciones declaradas en una especificación de paquete son visibles para los usuarios con privilegios en el paquete.

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de la Especificación del Paquete

Para crear paquetes, debe declarar todas las construcciones públicas en la especificación del paquete.

- Especifique la opción OR REPLACE si está sobrescribiendo una especificación del paquete existente.
- Inicialice una variable con una fórmula o un valor constante en la declaración, si es necesario; de lo contrario, la variable se inicializa implícitamente en NULL.

A continuación aparecen definiciones de elementos en la sintaxis del paquete:

- **package_name**: Especifica un nombre para el paquete que debe ser único entre los objetos del esquema propietario. Incluir el nombre del paquete después de la clave END es opcional.
- **public type and variable declarations**: Declara tipos y subtipos definidos por el usuario, excepciones, cursor, constantes y variables públicas.
- **subprogram specification**: Especifica las declaraciones de funciones o el procedimiento público.

Nota: La especificación del paquete debe contener cabeceras de funciones y procedimientos terminados en un punto y coma sin la palabra clave IS (o AS) y su bloque PL/SQL. La implementación de un procedimiento o función declarados en la especificación de un paquete se realiza en el cuerpo del paquete.

Ejemplo de Especificación de un Paquete: comm_pkg

```
CREATE OR REPLACE PACKAGE comm_pkg IS
    std_comm NUMBER := 0.10; --initialized to 0.10
    PROCEDURE reset_comm(new_comm NUMBER);
END comm_pkg;
/
```

- **STD_COMM** es una variable global inicializada en 0,10.
- **RESET_COMM** es un procedimiento público que se utiliza para restablecer la comisión estándar basada en algunas reglas de negocios. Se implementa en el cuerpo del paquete.

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ejemplo de Especificación de un Paquete: comm_pkg

El ejemplo de la transparencia crea un paquete llamado comm_pkg que se utiliza para gestionar reglas de procesamiento de negocios para el cálculo de comisiones.

La variable std_comm pública (global) se declara para contener un máximo de comisiones de porcentaje permitidas para la sesión de usuario y se inicializa en 0,10 (es decir, 10%).

El procedimiento público reset_comm se declara para aceptar un nuevo porcentaje de comisión que actualiza el porcentaje de comisión estándar si se aceptan las reglas de validación de comisiones. Las reglas de validación para el restablecimiento de la comisión no se hacen públicas ni aparecen en la especificación del paquete. Las reglas de validación se gestionan con una función privada en el cuerpo del paquete.

Creación del Cuerpo del Paquete

Sintaxis:

```
CREATE [OR REPLACE] PACKAGE BODY package_name IS|AS  
    private type and variable declarations  
    subprogram bodies  
    [BEGIN initialization statements]  
END [package_name];
```

- La opción OR REPLACE borra y vuelve a crear el cuerpo del paquete.
- Los identificadores definidos en el cuerpo del paquete son privados y no son visibles fuera del cuerpo del paquete.
- Todas las construcciones privadas se deben declarar antes de hacer referencia a ellas.
- Las construcciones públicas son visibles en el cuerpo del paquete.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación del Cuerpo del Paquete

Cree un cuerpo de paquete para definir e implementar todos los subprogramas públicos y las construcciones privadas soportadas. Al crear el cuerpo de un paquete, realice lo siguiente:

- Especifique la opción OR REPLACE para sobrescribir el cuerpo del paquete existente.
- Defina los subprogramas en un orden adecuado. El principio básico es que se debe declarar una variable o subprograma antes de que otros componentes puedan hacer referencia a ella en el mismo cuerpo del paquete. Es común ver todos los subprogramas y variables privadas definidas primero y los subprogramas públicos definidos en último lugar en el cuerpo del paquete.
- El cuerpo del paquete debe terminar la implementación de todos los procedimientos o funciones declaradas en la especificación del paquete.

A continuación aparecen definiciones de elementos de la sintaxis del cuerpo del paquete:

- **package_name**: Especifica un nombre para el paquete que debe ser el mismo que el de la especificación del paquete. El uso del nombre del paquete después de la palabra clave END es opcional.
- **public type and variable declarations**: Declara tipos y subtipos definidos por el usuario, excepciones, cursos, constantes y variables privadas.
- **subprogram specification**: Especifica la implementación completa de funciones o procedimientos públicos y/o privados.
- **[BEGIN initialization statements]**: Es un bloque opcional de código de inicialización que se ejecuta cuando se hace referencia al paquete por primera vez.

Ejemplo del Cuerpo del Paquete: comm_pkg

```
CREATE OR REPLACE PACKAGE BODY comm_pkg IS
  FUNCTION validate(comm NUMBER) RETURN BOOLEAN IS
    max_comm employees.commission_pct%type;
  BEGIN
    SELECT MAX(commission_pct) INTO max_comm
    FROM employees;
    RETURN (comm BETWEEN 0.0 AND max_comm);
  END validate;
  PROCEDURE reset_comm (new_comm NUMBER) IS BEGIN
    IF validate(new_comm) THEN
      std_comm := new_comm; -- reset public var
    ELSE
      RAISE_APPLICATION_ERROR(
        -20210, 'Bad Commission');
    END IF;
  END reset_comm;
END comm_pkg;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ejemplo del Cuerpo del Paquete: comm_pkg

La transparencia muestra el cuerpo del paquete completo para comm_pkg con una función privada denominada validate para comprobar las comisiones válidas. La validación necesita que la comisión sea positiva y mayor que la comisión más alta de los empleados existentes. El procedimiento reset_comm llama a la función de validación privada antes de cambiar la comisión estándar en std_comm. En el ejemplo, tenga en cuenta lo siguiente:

- La variable std_comm a la que se hace referencia en el procedimiento reset_comm es una variable pública. A las variables declaradas en la especificación del paquete, como std_comm, se les puede hacer referencia directamente sin cualificación.
- El procedimiento reset_comm implementa la definición pública en la especificación.
- En el cuerpo comm_pkg, la función validate es privada y se hace referencia a ella directamente desde el procedimiento reset_comm sin cualificación.

Nota: La función validate aparece antes del procedimiento reset_comm, ya que el procedimiento reset_comm hace referencia a la función validate. Es posible crear declaraciones anticipadas para subprogramas en el cuerpo del paquete si fuera necesario cambiar el orden de aparición. Si la especificación de un paquete sólo declara tipos, constantes, variables y excepciones sin ninguna especificación de subprogramas, entonces el cuerpo del paquete no es necesario. Sin embargo, el cuerpo se puede utilizar para inicializar elementos declarados en la especificación del paquete.

Llamada a Subprogramas de Paquete

- **Llamada a una función en el mismo paquete:**

```
CREATE OR REPLACE PACKAGE BODY comm_pkg IS ...
  PROCEDURE reset_comm(new_comm NUMBER) IS
    BEGIN
      IF validate(new_comm) THEN
        std_comm := new_comm;
      ELSE ...
      END IF;
    END reset_comm;
END comm_pkg;
```

- **Llamada a un procedimiento de paquete desde iSQL*Plus:**

```
EXECUTE comm_pkg.reset_comm(0.15)
```

- **Llamada a un procedimiento de paquete en un esquema diferente:**

```
EXECUTE scott.comm_pkg.reset_comm(0.15)
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Llamada a Subprogramas de Paquete

Después de almacenar el paquete en la base de datos, puede llamar a subprogramas privados o públicos del mismo paquete o subprogramas públicos que estén fuera del paquete. Indique el nombre de paquete totalmente cualificado del subprograma al llamarlo desde fuera del paquete. Utilice la sintaxis `package_name.subprogram`.

Indicar el nombre de subprograma totalmente cualificado al llamarlo desde el mismo paquete es opcional.

Ejemplo 1: Llama a la función `validate` desde el procedimiento `reset_comm` en el mismo paquete. El prefijo de nombre del paquete no es necesario, es opcional.

Ejemplo 2: Llama al procedimiento `reset_comm` desde iSQL*Plus (un entorno que está fuera del paquete) para restablecer la comisión predominante en 0,15 para la sesión de usuario.

Ejemplo 3: Llama al procedimiento `reset_comm` que es propiedad de un usuario de esquema llamado SCOTT. Con iSQL*Plus, el procedimiento de paquete cualificado lleva como prefijo el nombre del esquema. Esto se puede simplificar utilizando un sinónimo que haga referencia a `schema.package_name`.

Si se ha creado un enlace de base de datos llamado NY para una base de datos remota en la que se ha creado el procedimiento de paquete `reset_comm`, para llamar al procedimiento remoto, utilice:

```
EXECUTE comm_pkg.reset_comm@NY(0.15)
```

Creación y Uso de Paquetes sin Cuerpo

```
CREATE OR REPLACE PACKAGE global_consts IS
    mile_2_kilo CONSTANT NUMBER := 1.6093;
    kilo_2_mile CONSTANT NUMBER := 0.6214;
    yard_2_meter CONSTANT NUMBER := 0.9144;
    meter_2_yard CONSTANT NUMBER := 1.0936;
END global_consts;
```

```
BEGIN DBMS_OUTPUT.PUT_LINE('20 miles = ' ||
                           20 * global_consts.mile_2_kilo || ' km');
END;
```

```
CREATE FUNCTION mtr2yrd(m NUMBER) RETURN NUMBER IS
BEGIN
    RETURN (m * global_consts.meter_2_yard);
END mtr2yrd;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(mtr2yrd(1))
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación y Uso de Paquetes sin Cuerpo

Las variables y constantes declaradas en subprogramas autónomos existen solamente mientras se ejecuta el subprograma. Para proporcionar datos que existan durante la sesión de usuario, cree una especificación de paquete que contenga variables públicas (globales) y declaraciones de constantes. En este caso, cree una especificación de paquete sin un cuerpo de paquete, esto se conoce como un *paquete sin cuerpo*. Tal como se ha descrito anteriormente en esta lección, si una especificación sólo declara tipos, constantes, variables y excepciones, entonces el cuerpo del paquete no es necesario.

Ejemplos

El primer recuadro de código de la transparencia crea una especificación de paquete sin cuerpo con varias constantes que se utilizarán para los ratios de conversión. No es necesario el cuerpo del paquete para soportar esta especificación de paquete.

El segundo recuadro de código hace referencia a la constante `mile_2_kilo` en el paquete `global_consts` agregando el nombre del paquete como prefijo del identificador de la constante.

El tercer ejemplo crea una función autónoma `mtr2yrd` para convertir metros en yardas y utiliza el ratio de conversión de constantes `meter_2_yard` declarado en el paquete `global_consts`. La función se llama en un parámetro `DBMS_OUTPUT.PUT_LINE`.

Regla a seguir: Cuando se hace referencia a una variable, cursor, constante o excepción desde fuera de un paquete, debe cualificarlos con el nombre del paquete.

Eliminación de Paquetes

- **Para eliminar la especificación y el cuerpo del paquete, utilice la siguiente sintaxis:**

```
DROP PACKAGE package_name;
```

- **Para eliminar el cuerpo del paquete, utilice la siguiente sintaxis:**

```
DROP PACKAGE BODY package_name;
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Eliminación de Paquetes

Cuando ya no se necesita un paquete, se puede utilizar una sentencia SQL en iSQL*Plus para eliminarlo. Un paquete está compuesto por dos partes, por lo tanto, puede eliminar el paquete entero o puede eliminar el cuerpo del paquete y conservar la especificación del mismo.

Visualización de Paquetes en el Diccionario de Datos

El código de origen para paquetes PL/SQL se mantiene y es visible a través de las tablas `USER_SOURCE` y `ALL_SOURCE` en el diccionario de datos.

- Para ver la especificación del paquete, utilice:

```
SELECT text
FROM   user_source
WHERE  name = 'COMM_PKG' AND type = 'PACKAGE';
```

- Para ver el cuerpo del paquete, utilice:

```
SELECT text
FROM   user_source
WHERE  name = 'COMM_PKG' AND type = 'PACKAGE BODY';
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Visualización de Paquetes en el Diccionario de Datos

El código de origen de paquetes PL/SQL también se almacena en tablas del diccionario de datos como funciones y procedimientos autónomos. El código de origen es visible en el diccionario de datos cuando se ejecuta una sentencia SELECT en las tablas `USER_SOURCE` y `ALL_SOURCE`.

Al consultar el paquete, utilice una condición en la que la columna `TYPE` sea:

- Igual a '`PACKAGE`' para mostrar el código de origen de la especificación del paquete.
- Igual a '`PACKAGE BODY`' para mostrar el código de origen del cuerpo del paquete.

Nota: Los valores de las columnas `NAME` y `TYPE` deben estar en mayúsculas.

Instrucciones para la Escritura de Paquetes

- **Construir paquetes para uso general.**
- **Definir la especificación del paquete antes del cuerpo.**
- **La especificación del paquete debe contener solamente las construcciones que desee que sean públicas.**
- **Colocar elementos en la parte de la declaración del cuerpo del paquete cuando deba mantenerlos a los largo de una sesión o entre transacciones.**
- **Para realizar cambios en la especificación del paquete es necesario recompilar cada subprograma de referencia.**
- **La especificación del paquete debe contener tantas construcciones como sea posible.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Instrucciones para la Escritura de Paquetes

Mantenga los paquetes tan generales como sea posible, para que se puedan volver a utilizar en aplicaciones futuras. Evite también escribir paquetes que dupliquen las funciones proporcionadas por el servidor de Oracle.

Las especificaciones de paquetes reflejan el diseño de la aplicación; definalas antes de definir el cuerpo de los paquetes. La especificación del paquete debe contener sólo las construcciones que deban ser visibles a los usuarios del paquete. Por lo tanto, otros desarrolladores no pueden hacer un uso incorrecto del paquete basando el código en detalles irrelevantes.

Coloque elementos en la parte de la declaración del cuerpo del paquete cuando deba mantenerlos a los largo de una sesión o entre transacciones. Por ejemplo, declare una variable llamada NUMBER_EMPLOYED como una variable privada si es necesario mantener las llamadas a procedimientos que utilicen la variable. Al declarar una variable como global en la especificación del paquete, el valor de esa variable global se inicializa en una sesión la primera vez que se llama a una construcción del paquete.

Los cambios en el cuerpo del paquete no necesitan que se recompilen las construcciones dependientes mientras que los cambios en la especificación del paquete sí necesitan la recompilación de todos los subprogramas almacenados a los que hace referencia el paquete. Para reducir la necesidad de recompilación cuando se cambia el código, coloque un par de construcciones como posibles en una especificación de paquete.

Ventajas del Uso de Paquetes

- **Capacidad de organización en módulos:**
Encapsulamiento de construcciones relacionadas
- **Fácil mantenimiento:** **Mantenimiento de funcionalidades relacionadas de forma lógica**
- **Fácil diseño de aplicaciones:** **Codificación y compilación de la especificación y el cuerpo por separado**
- **Ocultación de información:**
 - **Sólo las declaraciones de la especificación del paquete son visibles y accesibles para las aplicaciones.**
 - **Las construcciones privadas del cuerpo del paquete están ocultas y son inaccesibles.**
 - **Toda la codificación está oculta en el cuerpo del paquete.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ventajas del Uso de Paquetes

Los paquetes proporcionan una alternativa a la creación de procedimientos y funciones como objetos de esquema autónomos y ofrecen varias ventajas.

Capacidad de organización en módulos y fácil mantenimiento: Puede encapsular estructuras de programas relacionados de forma lógica en un módulo con nombre. Cada paquete es fácil de entender y la interfaz entre paquetes es sencilla, clara y está bien definida.

Fácil diseño de aplicaciones: Todo lo que necesita en principio es la información de la interfaz en la especificación del paquete. Puede codificar y compilar una especificación sin su cuerpo. A continuación, también puede compilar los subprogramas almacenados que hacen referencia al paquete. No es necesario definir el cuerpo del paquete por completo hasta que esté listo para terminar la aplicación.

Ocultación de información: El usuario decide qué construcciones son públicas (visibles y accesibles) y cuáles son privadas (ocultas e inaccesibles). Las declaraciones de la especificación del paquete son visibles y accesibles para las aplicaciones. El cuerpo del paquete oculta la definición de las construcciones privadas para que sólo se vea afectado el paquete (pero no la aplicación ni los programas de llamada) si cambia la definición. Esto le permite cambiar la implementación sin tener que recompilar los programas de llamada. Además, ocultando los detalles de la implementación a los usuarios, se protege la integridad del paquete.

Ventajas del Uso de Paquetes

- **Funcionalidad adicional: Persistencia de variables y cursosres**
- **Mejor rendimiento:**
 - El paquete completo se carga en la memoria cuando se hace referencia al mismo por primera vez.
 - Sólo existe una copia en la memoria para todos los usuarios.
 - La jerarquía de dependencia se simplifica.
- **Sobrecarga: Varios subprogramas con el mismo nombre**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ventajas del Uso de Paquetes (continuación)

Funcionalidad adicional: Los cursosres y variables públicos empaquetados se mantienen durante una sesión. Por lo tanto, los pueden compartir todos los subprogramas que se ejecutan en el entorno. También le permiten mantener datos entre transacciones sin tener que almacenarlos en la base de datos. Las construcciones privadas también se mantienen durante la sesión pero sólo se puede acceder a ellas en el paquete.

Mejor rendimiento: Al llamar por primera vez a un subprograma empaquetado, el paquete completo se carga en memoria. Por lo tanto, las llamadas posteriores a subprogramas relacionados en el paquete no necesitan más E/S del disco. Los subprogramas empaquetados también pueden parar dependencias en cascada y evitar compilaciones innecesarias.

Sobrecarga: Con paquetes, se pueden sobrecargar procedimientos y funciones, lo que significa que se pueden crear varios subprogramas con el mismo nombre en el mismo paquete, cada uno con parámetros de tipo de dato o número diferente.

Nota: Las dependencias se abordan en detalle en la lección titulada “Gestión de Dependencias”.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- **Mejorar la organización, gestión, seguridad y rendimiento del código mediante el uso de paquetes**
- **Crear y eliminar cuerpos y especificaciones de paquetes**
- **Agrupar funciones y procedimientos relacionados en un paquete**
- **Encapsular el código en un cuerpo de paquete**
- **Definir y utilizar componentes en paquetes sin cuerpo**
- **Cambiar el cuerpo de un paquete sin que la especificación se vea afectada**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen

Agrupe funciones y procedimientos relacionados en un paquete. Los paquetes mejoran la organización, gestión, seguridad y rendimiento.

Un paquete está compuesto por una especificación de paquete y el cuerpo de un paquete. Puede cambiar el cuerpo de un paquete sin que la especificación se vea afectada.

Los paquetes le permiten ocultar a los usuarios el código de origen. Al llamar por primera vez a un paquete, el paquete completo se carga en memoria. Esto reduce el acceso del disco para llamadas posteriores.

Resumen

Comando	Tarea
CREATE [OR REPLACE] PACKAGE	Crear [o modificar] una especificación de paquete existente.
CREATE [OR REPLACE] PACKAGE BODY	Crear [o modificar] un cuerpo de paquete existente.
DROP PACKAGE	Eliminar la especificación del paquete y el cuerpo del paquete.
DROP PACKAGE BODY	Eliminar el cuerpo del paquete solamente.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen (continuación)

Puede crear, suprimir y modificar paquetes. Puede eliminar el cuerpo y la especificación del paquete con el comando **DROP PACKAGE**. Puede borrar el cuerpo de un paquete sin que la especificación se vea afectada.

Práctica 3: Visión General

En esta práctica se abordan los siguientes temas:

- **Creación de paquetes**
- **Llamada a unidades de programa de paquete**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica 3: Visión General

En esta práctica creará cuerpos y especificaciones de paquetes. Llame a las construcciones de los paquetes con datos sencillos.

Práctica 3

1. Cree una especificación y cuerpo del paquete denominado JOB_PKG que contenga una copia de los procedimientos ADD_JOB, UPD_JOB y DEL_JOB así como la función GET_JOB.

Consejo: Puede guardar la especificación y el cuerpo del paquete en dos archivos separados (por ejemplo, p3q1_s.sql y p3q1_b.sql para la especificación y el cuerpo del paquete respectivamente). Incluya una sentencia SHOW ERRORS después de la sentencia CREATE PACKAGE en cada archivo. Asimismo, coloque todo el código en un sólo archivo.

Nota: Utilice el código guardado previamente en los archivos de comandos al crear el paquete.

- a. Cree la especificación del paquete incluidos los procedimientos y las cabeceras de función como construcciones públicas.

Nota: Piense si aún necesita los procedimientos y funciones autónomos que acaba de empaquetar.

- b. Cree el cuerpo del paquete con las implementaciones de cada uno de los subprogramas.
- c. Llame al procedimiento empaquetado ADD_JOB transfiriendo como parámetros los valores IT_SYSAN y SYSTEMS ANALYST.
- d. Consulte la tabla JOBS para ver el resultado.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_SYSAN	Systems Analyst		

2. Cree y llame a un paquete que contenga construcciones públicas y privadas.

- a. Cree una especificación del paquete y un cuerpo del paquete denominados EMP_PKG que contengan los procedimientos ADD_EMPLOYEE y GET_EMPLOYEE como construcciones públicas e incluyan la función VALID_DEPTID como construcción privada.
- b. Llame al procedimiento EMP_PKG.GET_EMPLOYEE y utilice el identificador de departamento 15 para la empleada Jane Harris con correo electrónico JAHARRIS. Como el identificador de departamento 15 no existe recibirá un mensaje de error como se especifica en el manejador de excepciones del procedimiento.
- c. Llame al procedimiento empaquetado GET_EMPLOYEE utilizando el identificador de departamento 80 para el empleado David Smith con correo electrónico DASMITH.

Uso de Más Conceptos de Paquete

4

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- **Sobrecargar funciones y procedimientos de paquetes**
- **Usar declaraciones anticipadas**
- **Crear un bloque de inicialización en un cuerpo del paquete**
- **Gestionar estados de datos de paquete persistentes mientras dure una sesión**
- **Usar tablas y registros PL/SQL en paquetes**
- **Ajustar el código de origen almacenado en el diccionario de datos para que no sea legible**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetivos

Esta lección presenta las funciones más avanzadas de PL/SQL, incluida la sobrecarga, la referencia anticipada, el procedimiento que se realiza sólo una vez y la persistencia de las variables, constantes, excepciones y cursor. También explica el efecto de las funciones de empaquetado utilizadas en sentencias SQL.

Sobrecarga de Subprogramas

La función de sobregarga en PL/SQL:

- Permite crear dos o más subprogramas con el mismo nombre
- Necesita que los parámetros formales del subprograma varíen en número, orden o tipo de dato
- Permite crear formas flexibles para llamar a los subprogramas con datos diferentes
- Proporciona una forma de ampliar la funcionalidad sin perder el código existente

Nota: La sobrecarga se puede realizar con subprogramas locales, subprogramas de paquete y métodos de tipo, pero no con subprogramas autónomos.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Sobrecarga de Subprogramas

La función de sobrecarga en PL/SQL permite desarrollar dos o más subprogramas empaquetados con el mismo nombre. La sobrecarga es útil cuando desea que un subprograma acepte juegos similares de parámetros que tienen diferentes tipos de dato. Por ejemplo, la función TO_CHAR se puede denominar de varias formas, permitiéndole convertir un número o una fecha en una cadena de caracteres.

PL/SQL permite la sobrecarga de nombres de subprogramas de paquete y métodos de tipo de objeto.

La regla clave es que puede utilizar el mismo nombre para diferentes subprogramas mientras que sus parámetros formales varíen en número, orden o tipo de dato.

Considere el uso de la sobrecarga:

- Si las reglas de procesamiento para dos o más subprogramas son similares, pero el tipo o número de parámetros utilizados varía.
- Si desea proporcionar formas alternativas para buscar diferentes datos con diversos criterios de búsqueda. Por ejemplo, puede que desee buscar empleados por su identificador de empleado y proporcionar una forma de buscar empleados por su apellido. La lógica es esencialmente la misma, pero los parámetros o criterios de búsqueda varían.

Sobrecarga de Subprogramas (continuación)

- Si desea ampliar la funcionalidad cuando no desea sustituir el código existente.

Nota: Los subprogramas autónomos no se pueden sobrecargar. La escritura de subprogramas locales en métodos de tipo de objeto no se analiza en este curso.

Restricciones

No puede sobrecargar:

- Dos subprogramas en los que sus parámetros formales sólo varían en el tipo de dato y, además, los diferentes tipos de dato son de la misma familia (NUMBER y DECIMAL pertenecen a la misma familia.)
- Dos subprogramas en los que sus parámetros formales sólo varían en el subtipo y, además, los diferentes subtipos están basados en tipos de la misma familia (VARCHAR y STRING son subtipos PL/SQL de VARCHAR2.)
- Dos funciones en las que sólo varía el tipo de retorno, incluso si los tipos son de diferentes familias

Si sobrecarga subprogramas con las funciones anteriores, obtendrá un mensaje de error en tiempo de ejecución.

Nota: Las restricciones anteriores se aplicarán si los nombres de los parámetros también son los mismos. Si utiliza nombres diferentes para el parámetro, puede llamar a los subprogramas utilizando la notación con nombre para los parámetros.

Resolución de Llamadas

El compilador intenta buscar una declaración que coincida con la llamada. Busca primero en el ámbito actual y, a continuación, si es necesario, en los ámbitos sucesivos de delimitación. El compilador deja de buscar si encuentra una o más declaraciones de subprograma en las que el nombre coincide con el nombre del subprograma llamado. Para subprogramas con nombres similares al mismo nivel del ámbito, el compilador necesita una coincidencia exacta en número, orden y tipo de dato entre los parámetros reales y formales.

Sobrecarga: Ejemplo

```
CREATE OR REPLACE PACKAGE dept_pkg IS
    PROCEDURE add_department(deptno NUMBER,
                             name VARCHAR2 := 'unknown', loc NUMBER := 1700);
    PROCEDURE add_department(
                             name VARCHAR2 := 'unknown', loc NUMBER := 1700);
END dept_pkg;
/
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Sobrecarga: Ejemplo

Esta transparencia muestra la especificación de paquete dept_pkg con un procedimiento sobrecargado denominado add_department. La primera declaración considera tres parámetros que se utilizan para proporcionar datos para un nuevo registro de departamento insertado en la tabla de departamentos. La segunda declaración sólo considera dos parámetros, porque esta versión genera de forma interna el identificador de departamento a través de una secuencia Oracle.

Nota: El ejemplo utiliza tipos de dato básicos para sus argumentos para asegurar que el ejemplo se ajusta al espacio proporcionado. Es mejor especificar tipos de dato utilizando el atributo %TYPE para variables que se utilizan para llenar columnas en las tablas de base de datos, como en el siguiente ejemplo:

```
PROCEDURE add_department
    (deptno departments.department_id%TYPE,
     name departments.department_name%TYPE := 'unknown',
     loc departments.location_id%TYPE := 1700);
```

Sobrecarga: Ejemplo

```
CREATE OR REPLACE PACKAGE BODY dept_pkg IS
  PROCEDURE add_department (deptno NUMBER,
    name VARCHAR2:='unknown', loc NUMBER:=1700) IS
  BEGIN
    INSERT INTO departments(department_id,
      department_name, location_id)
    VALUES (deptno, name, loc);
  END add_department;

  PROCEDURE add_department (
    name VARCHAR2:='unknown', loc NUMBER:=1700) IS
  BEGIN
    INSERT INTO departments (department_id,
      department_name, location_id)
    VALUES (departments_seq.NEXTVAL, name, loc);
  END add_department;
END dept_pkg;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Sobrecarga: Ejemplo (continuación)

Si llama a add_department con un identificador de departamento proporcionado explícitamente, PL/SQL utiliza la primera versión del procedimiento. Considere el siguiente ejemplo:

```
EXECUTE dept_pkg.add_department(980,'Education',2500)
SELECT * FROM departments
WHERE department_id = 980;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
980	Education		2500

Si llama a add_department sin ningún identificador de departamento, PL/SQL utiliza la segunda versión:

```
EXECUTE dept_pkg.add_department ('Training', 2400)
SELECT * FROM departments
WHERE department_name = 'Training';
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
320	Training		2400

Sobrecarga y el Paquete STANDARD

- Un paquete denominado STANDARD define el entorno PL/SQL y las funciones incorporadas.
- La mayoría de las funciones incorporadas están sobrecargadas. Un ejemplo es la función TO_CHAR:

```
FUNCTION TO_CHAR (p1 DATE) RETURN VARCHAR2;
FUNCTION TO_CHAR (p2 NUMBER) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 DATE, p2 VARCHAR2) RETURN
VARCHAR2;
FUNCTION TO_CHAR (p1 NUMBER, p2 VARCHAR2) RETURN
VARCHAR2;
```

- Un subprograma PL/SQL con el mismo nombre que el subprograma incorporado sustituye la declaración estándar en el contexto local, a menos que cualifique el subprograma incorporado con su nombre de paquete.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Sobrecarga y el Paquete STANDARD

Un paquete denominado STANDARD define el entorno PL/SQL y declara globalmente los tipos, excepciones y subprogramas que están disponibles automáticamente para los programas PL/SQL. La mayoría de las funciones incorporadas del paquete STANDARD están sobrecargados. Por ejemplo, la función TO_CHAR tiene cuatro declaraciones diferentes, tal y como se muestra en la transparencia. La función TO_CHAR puede obtener el tipo de dato DATE o NUMBER y convertirlo en el tipo de dato de carácter. El formato al que se debe convertir la fecha o el número también se puede especificar en la llamada de función.

Si vuelve a declarar un subprograma incorporado en otro programa PL/SQL, la declaración local sustituye al subprograma estándar o incorporado. Para poder acceder al subprograma incorporado, debe cualificarlo con su nombre de paquete. Por ejemplo, si vuelve a declarar la función TO_CHAR para acceder a la función incorporada, se debe referir a ella como STANDARD.TO_CHAR.

Si vuelve a declarar un subprograma incorporado como un subprograma autónomo, para acceder al subprograma debe cualificarlo con el nombre de esquema. Por ejemplo, SCOTT.TO_CHAR.

Uso de Declaraciones Anticipadas

- Los lenguajes estructurados por bloques (como PL/SQL) deben declarar identificadores antes de que se haga referencia a ellos.
- Ejemplo de un problema de referencia:

```
CREATE OR REPLACE PACKAGE BODY forward_pkg IS
  PROCEDURE award_bonus( . . . ) IS
    BEGIN
      calc_rating( . . . );      --illegal reference
    END;

  PROCEDURE calc_rating( . . . ) IS
    BEGIN
      . . .
    END;
END forward_pkg;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de Declaraciones Anticipadas

En general, PL/SQL es como otros lenguajes estructurados por bloques y no permite referencias anticipadas. Debe declarar un identificador antes de utilizarlo. Por ejemplo, un subprograma se debe declarar antes de poder llamarlo.

Normalmente, los estándares de codificación necesitan que los subprogramas se guarden alfabéticamente para que sea más fácil encontrarlos. En este caso, puede que surjan problemas, como se muestra en el ejemplo de la transparencia, en el que no se puede hacer referencia al procedimiento `calc_rating` porque todavía no se ha declarado.

Puede resolver el problema de referencia no válida invirtiendo el orden de los dos procedimientos. Sin embargo, esta fácil solución no funciona si las reglas de codificación necesitan subprogramas que se declaren por orden alfabético.

La solución en este caso es utilizar declaraciones anticipadas proporcionadas en PL/SQL. Una declaración anticipada permite declarar la cabecera de un subprograma, es decir, la especificación de subprograma terminada en un punto y coma.

Nota: El error de compilación de `calc_rating` sólo se produce si `calc_rating` es un procedimiento empaquetado de forma privada. Si `calc_rating` se declara en la especificación de paquete, ya se ha declarado como si fuera una declaración anticipada y el compilador puede resolver su referencia.

Uso de Declaraciones Anticipadas

En el cuerpo del paquete, una declaración anticipada es una especificación de subprograma privado terminada en un punto y coma.

```
CREATE OR REPLACE PACKAGE BODY forward_pkg IS
  PROCEDURE calc_rating (...) -- forward declaration
    -- Subprograms defined in alphabetical order
    PROCEDURE award_bonus (...) IS
    BEGIN
      calc_rating (...); -- reference resolved!
      . . .
    END;
    PROCEDURE calc_rating (...) IS -- implementation
    BEGIN
      . . .
    END;
  END forward_pkg;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de Declaraciones Anticipadas (continuación)

Como se ha mencionado anteriormente, PL/SQL permite crear una declaración de subprograma especial denominada declaración anticipada. Una declaración anticipada puede ser necesaria para subprogramas privados en el cuerpo del paquete. Está formada por la especificación de subprograma terminada en un punto y coma. Las declaraciones anticipadas ayudan a:

- Definir subprogramas en orden lógico o alfabético
- Definir subprogramas mutuamente recusivos. Los programas mutuamente recursivos son programas que se llaman entre sí, directa o indirectamente.
- Agrupar y organizar de forma lógica los subprogramas en un cuerpo del paquete

Al crear una declaración anticipada:

- Los parámetros formales deben aparecer en la declaración anticipada y en el cuerpo del subprograma
- El cuerpo del subprograma puede aparecer en cualquier ubicación después de la declaración anticipada, pero ambos deben aparecer en la misma unidad de programa

Declaraciones Anticipadas y Paquetes

Normalmente, las especificaciones de subprograma se incluyen en la especificación del paquete y los cuerpos de subprograma en el cuerpo del paquete. Las declaraciones de subprograma públicas en la especificación del paquete no necesitan declaraciones anticipadas.

Bloque de Inicialización de Paquetes

El bloque del final del cuerpo del paquete se ejecuta una vez y se utiliza para inicializar variables de paquete públicas y privadas.

```
CREATE OR REPLACE PACKAGE taxes IS
    tax    NUMBER;
    ...
    -- declare all public procedures/functions
END taxes;
/
CREATE OR REPLACE PACKAGE BODY taxes IS
    ...
    -- declare all private variables
    ...
    -- define public/private procedures/functions
BEGIN
    SELECT    rate_value INTO tax
    FROM      tax_rates
    WHERE     rate_name = 'TAX';
END taxes;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Bloque de Inicialización de Paquetes

La primera vez que se hace referencia a un componente del paquete, éste se carga por completo en la memoria para la sesión de usuario. Por defecto, el valor inicial de las variables es NULL (si no se inicializa de forma explícita). Para inicializar las variables de paquete, puede:

- Utilizar operaciones de asignación en sus declaraciones para las tareas de inicialización simples
- Agregar bloque de código al final del cuerpo del paquete para realizar tareas de inicialización más complejas

Considere el bloque de código al final de un cuerpo del paquete como un bloque de inicialización de paquete que se ejecute una vez, cuando se llame al paquete por primera vez en la sesión de usuario.

El ejemplo de la transparencia muestra la variable pública `tax` que se está inicializando en el valor de la tabla `tax_rates` la primera vez que se hace referencia al paquete `taxes`.

Nota: Si inicializa la variable en la declaración utilizando una operación de asignación, ésta se sobrescribe con el código del bloque de inicialización al final del cuerpo del paquete. El bloque de inicialización se termina por la palabra clave `END` para el cuerpo del paquete.

Uso de Funciones de Paquete en SQL y Restricciones

- **Las funciones de paquete se pueden utilizar en sentencias SQL.**
- **Las funciones que se llaman desde:**
 - Una consulta o una sentencia DML no pueden terminar la transacción actual, crear o realizar un rollback a un punto de grabación o modificar el sistema o la sesión
 - Una consulta o sentencia DML paralela no puede ejecutar una sentencia DML ni modificar la base de datos
 - Una sentencia DML no puede leer ni modificar la tabla que está cambiando dicha sentencia

Nota: No están permitidos los subprogramas de llamada de función que rompen las restricciones anteriores.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de Funciones de Paquete en SQL y Restricciones

Al ejecutar una sentencia SQL que llama a una función almacenada, el servidor de Oracle debe conocer el nivel de pureza de las funciones almacenadas. Es decir, si las funciones no tienen las restricciones que se muestran en la transparencia. En general, las restricciones son cambios en las tablas de base de datos o variables de paquete públicas (aquellas declaradas en una especificación del paquete). Las restricciones pueden retrasar la ejecución de una consulta, cambiar los resultados dependientes del orden (y, por lo tanto, indeterminados) o necesitar que las variables de estado del paquete se mantengan en las sesiones de usuario. Hay varias restricciones que no están permitidas cuando una función se llama desde una consulta SQL o una sentencia DML. Por lo tanto, se aplican las siguientes restricciones a las funciones almacenadas llamadas desde las expresiones SQL:

- Una función llamada desde una consulta o una sentencia DML no pueden terminar la transacción actual, crear o realizar un rollback a un punto de grabación o modificar el sistema o la sesión.
- Una función llamada desde una sentencia de consulta o desde una sentencia DML paralela no puede ejecutar una sentencia DML ni modificar la base de datos.
- Una función llamada desde una sentencia DML no puede leer ni modificar la tabla concreta que está siendo modificada por dicha sentencia.

Nota: Antes de Oracle8i, el nivel de pureza se comprobaba en el momento de la compilación incluyendo PRAGMA RESTRICT_REFERENCES en la especificación del paquete. Después de Oracle8i, el nivel de pureza de las funciones se comprueba en tiempo de ejecución.

Función de Paquete en SQL: Ejemplo

```
CREATE OR REPLACE PACKAGE taxes_pkg IS
    FUNCTION tax (value IN NUMBER) RETURN NUMBER;
END taxes_pkg;
/
CREATE OR REPLACE PACKAGE BODY taxes_pkg IS
    FUNCTION tax (value IN NUMBER) RETURN NUMBER IS
        rate NUMBER := 0.08;
    BEGIN
        RETURN (value * rate);
    END tax;
END taxes_pkg;
/
```

```
SELECT taxes_pkg.tax(salary), salary, last_name
FROM employees;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Función de Paquete en SQL: Ejemplo

El primer recuadro de código muestra cómo crear la especificación de paquete y el cuerpo que incluye la función `tax` en el paquete `taxes_pkg`. El segundo recuadro de código muestra cómo llamar la función empaquetada `tax` en la sentencia `SELECT`. Los resultados de salida son similares a:

TAXES_PACK.TAX(SALARY)	SALARY	LAST_NAME
1920	24000	King
1360	17000	Kochhar
1360	17000	De Haan
720	9000	Hunold
480	6000	Ernst
422.4	5280	Austin
422.4	5280	Pataballa
369.6	4620	Lorentz
960	12000	Greenberg
...		

109 rows selected.

Nota: Si está utilizando versiones Oracle anteriores a 8i, debe afirmar el nivel de pureza de la función en la especificación de paquete utilizando `PRAGMA RESTRICT_REFERENCES`. Si no se especifica, obtendrá un mensaje de error que indica que la función `TAX` no garantiza que no se actualizará la base de datos al llamar a la función del paquete en una consulta.

Estado Persistente de Paquetes

La recopilación de variables de paquete y los valores definen el estado del paquete. El estado del paquete es:

- **Inicializado cuando el paquete se carga por primera vez**
- **Persistente (por defecto) mientras dure la sesión**
 - Almacenado en UGA (Área Global de Usuario)
 - Único para cada sesión
 - Sujeto a cambios cuando se llama a los subprogramas del paquete o se modifican las variables públicas
- **No persistente para la sesión, pero sí mientras dure una llamada de subprograma al utilizar PRAGMA SERIALLY_REUSEABLE en la especificación del paquete**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Estado Persistente de Paquetes

La recopilación de variables de paquete públicas y privadas representa el estado del paquete para la sesión de usuario. Es decir, el estado del paquete es el juego de valores almacenado en todas las variables del paquete en un momento determinado en el tiempo. En general, el estado del paquete existe mientras dure la sesión de usuario.

Las variables del paquete se inicializan la primera vez que se carga un paquete en la memoria para una sesión de usuario. Las variables del paquete son, por defecto, únicas para cada sesión y mantienen sus valores hasta que se termina la sesión de usuario. Es decir, las variables se almacenan en la memoria UGA asignada por la base de datos para cada sesión de usuario.

El estado del paquete cambia cuando se llama a un subprograma de paquete y su lógica modifica el estado de la variable. El estado público del paquete se puede modificar directamente por las operaciones adecuadas según su tipo.

Nota: Si agrega PRAGMA SERIALLY_RESUABLE a la especificación del paquete, la base de datos almacena las variables del paquete en el Área Global del Sistema (SGA) compartida a través de las sesiones de usuario. En este caso, el estado del paquete se mantiene mientras dure una llamada de subprograma o una referencia única a una construcción de paquetes. La directiva SERIALLY_REUSEABLE es útil si desea conservar la memoria y si el estado del paquete no necesita persistir para cada sesión de usuario.

Estado Persistente de las Variables de Paquetes: Ejemplo

Time	Events	State for: -Scott-		-Jones-	
		STD	MAX	STD	MAX
9:00	Scott> EXECUTE comm_pkg.reset_comm(0.25)	0.10 0.25	0.4	-	0.4
9:30	Jones> INSERT INTO employees(last_name,commission_pct) VALUES ('Madonna', 0.8);	0.25	0.4		0.8
9:35	Jones> EXECUTE comm_pkg.reset_comm (0.5)	0.25	0.4	0.1	0.8
10:00	Scott> EXECUTE comm_pkg.reset_comm(0.6) Err -20210 'Bad Commission'	0.25	0.4	0.5	0.8
11:00	Jones> ROLLBACK;	0.25	0.4	0.5	0.4
11:01	EXIT ...	0.25	0.4	-	0.4
12:00	EXEC comm_pkg.reset_comm(0.2)	0.25	0.4	0.2	0.4

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Estado Persistente de las Variables de Paquetes: Ejemplo

La secuencia de la transparencia está basada en dos usuarios: Scott y Jones, que ejecutan comm_pkg (tratado en la lección titulada “Creación de Paquetes”), donde el procedimiento reset_comm llama a la función validate para comprobar la nueva comisión. El ejemplo muestra cómo el estado persistente de la variable de paquete std.comm se mantiene en cada sesión de usuario.

A las 9:00: Scott llama a reset_comm con un nuevo valor de comisión de 0,25. El estado del paquete de std.comm se inicializa en 0,10 y, a continuación, se define en 0,25, el cual se valida, ya que es menor que el valor máximo de la base de datos, que es 0,4.

A las 9:30: Jones inserta una nueva fila en la tabla EMPLOYEES con un nuevo valor máximo de commission_pct: 0,8. Esto no se confirma, por lo que sólo Jones puede visualizarlo. El estado de Scott no se ve afectado.

A las 9:35: Jones llama a reset_comm con un nuevo valor de comisión de 0,5. El estado de std.comm de Jones se inicializa por primera vez en 0,10 y, a continuación, se define en el nuevo valor 0,5 que es válido para esta sesión con el valor máximo de la base de datos, 0,8.

A las 10:00: Scott llama a reset_comm con un nuevo valor de comisión de 0,6, que es mayor que el valor máximo de la base de datos visible para esta sesión, 0,4 (Jones no ha confirmado el valor 0,8.)

Entre las 11:00 y las 12:00: Jones realiza un rollback en la transacción y sale de la sesión. Jones se conecta a las 11:45 y ejecuta correctamente el procedimiento, definiendo su estado en 0,2.

Estado Persistente de un Cursor de Paquete

```
CREATE OR REPLACE PACKAGE BODY curs_pkg IS
  CURSOR c IS SELECT employee_id FROM employees;
  PROCEDURE open IS
  BEGIN
    IF NOT c%ISOPEN THEN    OPEN c;  END IF;
  END open;
  FUNCTION next(n NUMBER := 1) RETURN BOOLEAN IS
    emp_id employees.employee_id%TYPE;
  BEGIN
    FOR count IN 1 .. n LOOP
      FETCH c INTO emp_id;
      EXIT WHEN c%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('Id: ' || (emp_id));
    END LOOP;
    RETURN c%FOUND;
  END next;
  PROCEDURE close IS BEGIN
    IF c%ISOPEN THEN CLOSE c;  END IF;
  END close;
END curs_pkg;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Estado Persistente de un Cursor de Paquete

El código en la transparencia muestra el cuerpo del paquete para CURS_PKG para soportar la siguiente especificación del paquete:

```
CREATE OR REPLACE PACKAGE curs_pkg IS
  PROCEDURE open;
  FUNCTION next(n NUMBER := 1) RETURN BOOLEAN;
  PROCEDURE close;
END curs_pkg;
```

Para utilizar este paquete, realice los siguientes pasos para procesar las filas:

- Llame al procedimiento `open` para abrir el cursor.
- Llame al procedimiento `next` para recuperar una fila o un número determinado de ellas. Si solicita más filas de las que existen realmente, el procedimiento gestiona correctamente la terminación. Devuelve TRUE si se necesitan procesar más filas, de lo contrario, devuelve FALSE.
- Llame al procedimiento `close` para cerrar el cursor, antes o al principio del procesamiento de las filas.

Nota: La declaración de cursores es privada del paquete. Por lo tanto, se puede influir en el estado del cursor llamando al procedimiento del paquete y las funciones que aparecen en la transparencia.

Ejecución de CURS_PKG

```
SET SERVEROUTPUT ON
EXECUTE curs_pkg.open
DECLARE
    more BOOLEAN := curs_pkg.next(3);
BEGIN
    IF NOT more THEN
        curs_pkg.close;
    END IF;
END;
/
RUN -- repeats execution on the anonymous block
EXECUTE curs_pkg.close
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ejecución de CURS_PKG

Tenga en cuenta que el estado de una variable de paquete o cursor persiste entre las transacciones de la sesión. Sin embargo, el estado no persiste entre las diferentes sesiones para el mismo usuario. Las tablas de base de datos incluyen datos que persisten entre las sesiones y los usuarios. El comando SET SERVEROUTPUT ON prepara iSQL*Plus para mostrar los resultados de salida. La llamada a curs_pkg.open abre el cursor, el cual permanece abierto hasta que se termina la sesión o hasta que el cursor se cierra explícitamente. El bloque anónimo ejecuta la función next en la sección Declaration, inicializando la variable BOOLEAN more en TRUE, ya que hay más de tres filas en la tabla de empleados. El bloque comprueba el final del juego de resultados y cierra el cursor, si es apropiado. Cuando el bloque se ejecuta, aparecerán las primeras tres filas:

```
Id :103
Id :104
Id :105
```

El comando RUN vuelve a ejecutar el bloque anónimo y aparecen las tres filas siguientes:

```
Id :106
Id :107
Id :108
```

El comando EXECUTE curs_pkg.close cierra el cursor en el paquete.

Uso de Tablas PL/SQL de Registros en Paquetes

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  TYPE emp_table_type IS TABLE OF employees%ROWTYPE
    INDEX BY BINARY_INTEGER;
  PROCEDURE get_employees(emps OUT emp_table_type);
END emp_pkg;
/
```

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  PROCEDURE get_employees(emps OUT emp_table_type) IS
    i BINARY_INTEGER := 0;
  BEGIN
    FOR emp_record IN (SELECT * FROM employees)
    LOOP
      emps(i) := emp_record;
      i := i+1;
    END LOOP;
    END get_employees;
  END emp_pkg;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de Tablas de Registros de Procedimientos o Funciones en Paquetes

El paquete emp_pkg contiene un procedimiento get_employees que lee filas desde la tabla EMPLOYEES y devuelve las filas utilizando el parámetro OUT, que es una tabla PL/SQL de registros. Los puntos clave son los siguientes:

- employee_table_type se declara como tipo público.
- employee_table_type se utiliza para un parámetro de salida formal en el procedimiento y la variable employees en el bloque de llamada (que aparece a continuación).

En iSQL*Plus, puede llamar al procedimiento get_employees en un bloque PL/SQL anónimo utilizando la variable employees como se muestra en el siguiente ejemplo:

```
DECLARE
  employees  emp_pkg.emp_table_type;
BEGIN
  emp_pkg.get_employees(employees);
  DBMS_OUTPUT.PUT_LINE('Emp 4: ' || employees(4).last_name);
END;
/
```

Esto da como resultado:

```
Emp 4: Ernst
```

Wrapper PL/SQL

- **El wrapper PL/SQL es una utilidad autónoma que oculta elementos internos de la aplicación convirtiendo el código de origen PL/SQL en un código de objeto portable.**
- **El ajuste tiene las siguientes funciones:**
 - Independencia de plataforma
 - Carga dinámica
 - Enlace dinámico
 - Comprobación de dependencia
 - Importación y exportación normales al ser llamado

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

¿Qué es el Wrapper PL/SQL?

El wrapper PL/SQL es una utilidad autónoma que convierte el código de origen PL/SQL en un código de objeto portable. Al utilizarlo, puede desarrollar aplicaciones PL/SQL sin exponer el código de origen, que puede contener algoritmos propietarios y estructuras de datos. El wrapper convierte el código de origen legible en un código de origen no legible. Al ocultar elementos internos de la aplicación, evita que se haga un uso incorrecto de la aplicación.

El código ajustado, como los programas PL/SQL almacenados, tiene varias funciones:

- No depende de ninguna plataforma, por lo que no es necesario desarrollar varias versiones de la misma unidad de compilación.
- Permite la carga dinámica, por lo que los usuarios no necesitan cerrar y volver a enlazarse a una nueva función.
- Permite el enlace dinámico, por lo que las referencias externas se resuelven en tiempo de carga.
- Ofrece una estricta comprobación de dependencias, por lo que las unidades de programa invalidadas se recompilarán automáticamente cuando se llamen.
- Soporta importación y exportación normales, por lo que la utilidad de importación/exportación puede procesar los archivos ajustados.

Ejecución de Wrapper

La sintaxis de la línea de comandos es:

```
WRAP INAME=input_file_name [ONAME=output_file_name]
```

- Se necesita el argumento **INAME**.
- La extensión por defecto del archivo de entrada es **.sql**, a menos que se especifique en el nombre.
- El argumento **ONAME** es opcional.
- La extensión por defecto del archivo de salida es **.plb**, a menos que se especifique con el argumento **ONAME**.

Ejemplos:

```
WRAP INAME=student.sql  
WRAP INAME=student  
WRAP INAME=student.sql ONAME=student.plb
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ejecución de Wrapper

Wrapper es un ejecutable del sistema operativo denominado WRAP. Para ejecutar el wrapper, introduzca el siguiente comando en el prompt del sistema operativo:

```
WRAP INAME=input_file_name [ONAME=output_file_name]
```

Cada uno de los ejemplos mostrados en la transparencia toma un archivo denominado **student.sql** como entrada y crea un archivo de salida denominado **student.plb**.

Después de crear el archivo ajustado, ejecute el archivo **.plb** de iSQL*Plus para compilar y almacenar la versión ajustada del código de origen tal y como ejecutaría los archivos de comandos SQL.

Nota:

- Sólo se necesita el argumento **INAME**. Si no se especifica el argumento **ONAME**, el archivo de salida adquiere el mismo nombre que el archivo de entrada con una extensión **.plb**.
- No inserte ningún espacio alrededor de los signos igual de los argumentos de la línea de comandos.
- El archivo de salida puede tener cualquier extensión, pero la extensión por defecto es **.sql**.
- La sensibilidad a mayúsculas/minúsculas de los valores **INAME** y **ONAME** depende del sistema operativo.
- Generalmente, el archivo de salida es mucho más grande que el archivo de entrada.
- No inserte ningún espacio alrededor de los signos igual de los argumentos y valores **INAME** y **ONAME**.

Resultados del Ajuste

- Código de origen PL/SQL original en el archivo de entrada:

```
CREATE PACKAGE banking IS
    min_bal := 100;
    no_funds EXCEPTION;
    ...
END banking;
/
```

- Código ajustado en el archivo de salida:

```
CREATE PACKAGE banking
wrapped
012abc463e ...
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resultados del Ajuste

Al ajustar un tipo de objeto, paquete o subprograma tiene el siguiente formato: la cabecera, seguida por la palabra wrapped y, a continuación, el cuerpo cifrado.

El archivo de entrada puede contener cualquier combinación de sentencias SQL. Sin embargo, el wrapper PL/SQL sólo ajusta las siguientes sentencias CREATE:

- CREATE [OR REPLACE] TYPE
- CREATE [OR REPLACE] TYPE BODY
- CREATE [OR REPLACE] PACKAGE
- CREATE [OR REPLACE] PACKAGE BODY
- CREATE [OR REPLACE] FUNCTION
- CREATE [OR REPLACE] PROCEDURE

Todas las demás sentencias SQL CREATE se transfieren intactas al archivo de salida.

Instrucciones para el Ajuste

- **Debe ajustar sólo el cuerpo del paquete, no su especificación.**
- **El wrapper puede detectar errores sintácticos, no semánticos.**
- **El archivo de salida no se debe editar. Puede mantener el código de origen original y volverlo ajustar como sea necesario.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Instrucciones para el Ajuste

Entre estas instrucciones se incluyen:

- Al ajustar el paquete o el tipo de objeto, ajuste sólo el cuerpo, no la especificación. Por lo tanto, proporcione a otros desarrolladores la información que necesiten para utilizar el paquete sin exponer su implementación.
- Si su archivo de entrada contiene errores sintácticos, el wrapper PL/SQL los detecta y notifica. Sin embargo, el wrapper no puede detectar errores semánticos porque no resuelve referencias externas. Por ejemplo, el wrapper no notifica ningún error si no existe la tabla o vista amp;

```
CREATE PROCEDURE raise_salary (emp_id INTEGER, amount NUMBER)
AS
BEGIN
    UPDATE amp -- should be emp
        SET sal = sal + amount WHERE empno = emp_id;
END;
```

Sin embargo, el compilador PL/SQL resuelve las referencias externas. Por lo tanto, los errores semánticos se notifican al compilar el archivo de salida del wrapper (archivo .p1b).

- El archivo de salida no se debe editar porque su contenido no es legible. Para cambiar un objeto ajustado, necesita modificar el código de origen original y volver a ajustar el código.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- **Crear y llamar subprogramas sobrecargados**
- **Utilizar declaraciones anticipadas para subprogramas**
- **Escribir bloques de inicialización de paquete**
- **Mantener el estado persistente de los paquetes**
- **Utilizar el wrapper PL/SQL para ajustar el código**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen

La sobrecarga es una función que permite definir diferentes subprogramas con el mismo nombre. Es lógico asignar el mismo nombre a dos subprogramas cuando en ambos el procesamiento es el mismo, pero los parámetros transferidos a ellos varían.

PL/SQL permite para un subprograma especial una declaración denominada declaración anticipada. Una declaración anticipada permite definir subprogramas en orden lógico o alfabético, definir subprogramas recursivos mutuamente y agrupar subprogramas en un paquete.

Un bloque de inicialización de paquete se ejecuta sólo cuando se llama al paquete en la sesión de otro usuario. Puede utilizar esta función para inicializar variables sólo una vez por sesión.

Puede realizar un seguimiento del estado del cursor o variable de paquete, que persistirá a través de la sesión de usuario, desde que éste hace referencia a la variable o cursor por primera vez hasta que el usuario se desconecta.

Al utilizar el wrapper PL/SQL puede oscurecer el código de origen almacenado en la base de datos para proteger su propiedad intelectual.

Práctica 4: Visión General

En esta práctica se abordan los siguientes temas:

- **Uso de subprogramas sobrecargados**
- **Creación de un bloque de inicialización de paquete**
- **Uso de una declaración anticipada**
- **Uso de la utilidad WRAP para evitar que los usuarios descifren el código de origen**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica 4: Visión General

En esta práctica, puede modificar un paquete existente para que contenga subprogramas sobrecargados y utilizar declaraciones anticipadas. También puede crear un bloque de inicialización de paquete en el cuerpo del paquete para llenar una tabla PL/SQL. Puede utilizar la utilidad de línea de comandos WRAP para evitar que el código de origen sea legible en las tablas del diccionario de datos.

Práctica 4

1. Copie y modifique el código del paquete EMP_PKG que creó en la Práctica 3, Ejercicio 2 y sobrecargue el procedimiento ADD_EMPLOYEE.
 - a. En la especificación del paquete, agregue un nuevo procedimiento denominado ADD_EMPLOYEE, que acepte tres parámetros: nombre, apellido e identificador de departamento. Guarde y compile los cambios.
 - b. Implemente el nuevo procedimiento ADD_EMPLOYEE en el cuerpo del paquete para que formatee la dirección de correo electrónico en mayúsculas, utilizando la primera letra del nombre concatenada con las primeras siete letras del apellido. El procedimiento llamará al procedimiento ADD_EMPLOYEE existente para realizar la operación INSERT utilizando los parámetros y el correo electrónico formateado para proporcionar los valores. Guarde y compile los cambios.
 - c. Llame al nuevo procedimiento ADD_EMPLOYEE utilizando el nombre Samuel Joplin para agregarlo al departamento 30.
2. En el paquete EMP_PKG, cree dos funciones sobrecargadas denominadas GET_EMPLOYEE.
 - a. En la especificación, agregue una función GET_EMPLOYEE que acepte el parámetro denominado emp_id según el tipo employees.employee_id%TYPE y una segunda función GET_EMPLOYEE que acepte un parámetro denominado family_name del tipo employees.last_name%TYPE. Ambas funciones devolverán un EMPLOYEES%ROWTYPE. Guarde y compile los cambios.
 - b. En el cuerpo del paquete, implemente la primera función GET_EMPLOYEE para realizar una consulta sobre un empleado según su identificador y la segunda para utilizar el operador de igualdad sobre el valor proporcionado en el parámetro family_name. Guarde y compile los cambios.
 - c. Agregue un procedimiento de utilidad PRINT_EMPLOYEE al paquete que acepte EMPLOYEES%ROWTYPE como parámetro y muestre department_id, employee_id, first_name, last_name, job_id y salary para un empleado en una sola línea utilizando DBMS_OUTPUT. Guarde y compile los cambios.
 - d. Utilice un bloque anónimo para llamar a la función EMP_PKG.GET_EMPLOYEE con un identificador de empleado de 100 y con apellido 'Joplin'. Utilice el procedimiento PRINT_EMPLOYEE para mostrar los resultados para cada fila devuelta.
3. Como la compañía no cambia con frecuencia sus datos de departamento, puede mejorar el rendimiento de EMP_PKG agregando un procedimiento público INIT_DEPARTMENTS para llenar una tabla PL/SQL privada de identificadores de departamento válidos. Modifique la función VALID_DEPTID para utilizar el contenido de la tabla PL/SQL privada para validar los valores de los identificadores de departamento.
 - a. En la especificación del paquete, cree un procedimiento denominado INIT_DEPARTMENTS sin parámetros.

Práctica 4 (continuación)

- b. En el cuerpo del paquete, implemente el procedimiento INIT_DEPARTMENTS para almacenar todos los identificadores de departamento en una tabla de índice PL/SQL privada denominada valid_departments que contiene valores BOOLEAN. Utilice el valor de la columna department_id como índice para crear la entrada en la tabla de índice para indicar su presencia y asignar a la entrada un valor de TRUE. Declare la variable valid_departments y su definición de tipo boolean_tabtype antes que todos los procedimientos del cuerpo.
- c. En el cuerpo, cree un bloque de inicialización que llame al procedimiento INIT_DEPARTMENTS para inicializar la tabla. Guarde y compile los cambios.
4. Cambie el procesamiento de validación VALID_DEPTID para utilizar la tabla PL/SQL privada de identificadores de departamento.
 - a. Modifique VALID_DEPTID para realizar la validación utilizando la tabla PL/SQL de valores de identificadores de departamento. Guarde y compile los cambios.
 - b. Pruebe el código llamando a ADD_EMPLOYEE con el nombre James Bond en el departamento 15. ¿Qué sucede?
 - c. Inserte un departamento nuevo con identificador 15 y nombre Security para confirmar los cambios.
 - d. Pruebe el código llamando a ADD_EMPLOYEE con el nombre James Bond en el departamento 15. ¿Qué sucede?
 - e. Ejecute el procedimiento EMP_PKG.INIT_DEPARTMENTS para actualizar la tabla interna PL/SQL con los últimos datos del departamento.
 - f. Compruebe el código llamando a ADD_EMPLOYEE con el nombre de empleado James Bond que trabaja en el departamento 15. ¿Qué sucede?
 - g. Suprima al empleado James Bond y el departamento 15 de sus respectivas tablas, confirme los cambios y actualice los datos del departamento llamando al procedimiento EMP_PKG.INIT_DEPARTMENTS.
5. Reorganice los subprogramas en el cuerpo de la especificación del paquete para que estén en secuencia alfabética.
 - a. Edite la especificación del paquete y reorganice los subprogramas de forma alfabética. En iSQL*Plus, cargue y compile la especificación del paquete. ¿Qué sucede?
 - b. Edite el cuerpo del paquete y reorganice todos los subprogramas de forma alfabética. En iSQL*Plus, cargue y compile la especificación del paquete. ¿Qué sucede?
 - c. Corrija el error de compilación utilizando una declaración anticipada en el cuerpo para la referencia de subprograma incorrecta. Cargue y vuelva a crear el cuerpo del paquete. ¿Qué sucede?

Si tiene tiempo, realice el siguiente ejercicio:

6. Ajuste el cuerpo del paquete EMP_PKG y vuelva a crearlo.
 - a. Consulte el diccionario de datos para ver el origen para el cuerpo EMP_PKG.
 - b. Inicie una ventana de comandos y ejecute la utilidad de línea de comandos WRAP para ajustar el cuerpo del paquete EMP_PKG. Asigne una extensión .plb al archivo de salida.
Indicación: Copie el archivo (que ha guardado en el paso 5c) que contiene el cuerpo del paquete en un archivo denominado emp_pkb_b.sql.

Práctica 4 (continuación)

- c. Con *iSQL*Plus*, cargue y ejecute el archivo .plb que contiene el origen ajustado.
- d. Consulte el diccionario de datos para mostrar de nuevo el origen para el cuerpo del paquete EMP_PKG. ¿Son legibles las líneas del código de origen original?

Oracle Internal & OAI Use Only

Uso de Paquetes Proporcionados por Oracle en el Desarrollo de Aplicaciones

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- Describir cómo funciona el paquete **DBMS_OUTPUT**
- Utilizar **UTL_FILE** para dirigir la salida a los archivos del sistema operativo
- Utilizar el paquete **HTP** para generar una página Web sencilla
- Describir las principales funciones de **UTL_MAIL**
- Llamar al paquete **DBMS_SCHEDULER** para planificar el código PL/SQL para su ejecución

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetivos

En esta lección, aprenderá a utilizar algunos de los paquetes proporcionados por Oracle así como sus capacidades. Esta lección se centra en paquetes que generan texto o salida basada en Web, procesamiento de correo electrónico y la capacidad de planificación proporcionada.

Uso de Paquetes Proporcionados por Oracle

Los paquetes proporcionados por Oracle:

- Se proporcionan con el servidor de Oracle
- Amplían la funcionalidad de la base de datos
- Permiten el acceso a ciertas funciones SQL que normalmente están restringidas a PL/SQL

Por ejemplo, el paquete `DBMS_OUTPUT` se diseñó inicialmente para depurar programas PL/SQL.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de Paquetes Proporcionados por Oracle

Los paquetes se proporcionan con el servidor de Oracle para permitir algunas de las siguientes acciones:

- Acceso PL/SQL a ciertas funciones SQL
- La extensión de la funcionalidad de la base de datos

Puede utilizar la funcionalidad que proporcionan estos paquetes cuando cree la aplicación o es posible que prefiera utilizar estos paquetes como ideas cuando cree sus propios procedimientos almacenados.

La mayoría de los paquetes estándar se crean ejecutando `catproc.sql`. El paquete `DBMS_OUTPUT` es uno de los que más familiar le resultará en este curso. Conocerá este paquete si asistió al curso *Base de Datos Oracle 10g: Conceptos Básicos de PL/SQL*.

Lista de Algunos Paquetes Proporcionados por Oracle

A continuación se presenta una lista abreviada de algunos paquetes proporcionados por Oracle:

- **DBMS_ALERT**
- **DBMS_LOCK**
- **DBMS_SESSION**
- **DBMS_OUTPUT**
- **HTP**
- **UTL_FILE**
- **UTL_MAIL**
- **DBMS_SCHEDULER**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Lista de Algunos Paquetes Proporcionados por Oracle

La lista de paquetes PL/SQL proporcionados con una base de datos Oracle crece con la aparición de nuevas versiones. Sería imposible cubrir la totalidad del juego de paquetes y sus funciones en este curso. Para obtener más información, consulte el manual *PL/SQL Packages and Types Reference 10g* (conocido anteriormente como *PL/SQL Supplied Packages Reference*). En esta lección se abordan los últimos cinco paquetes de la lista.

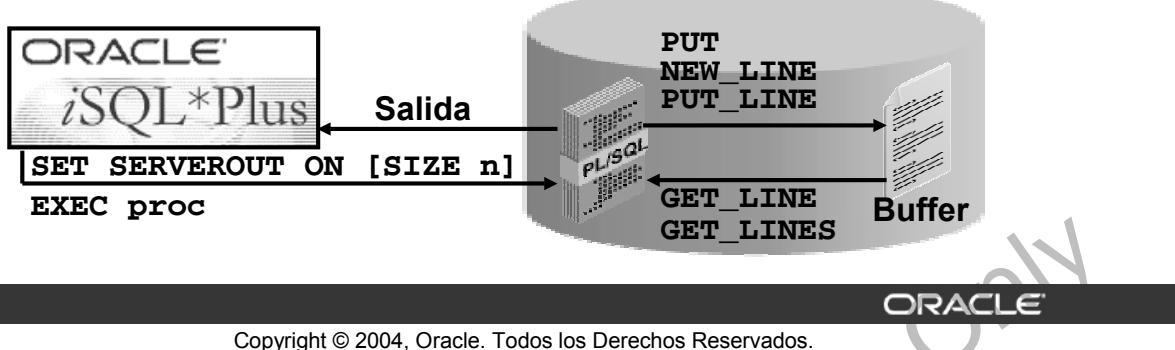
A continuación se describen brevemente todos estos paquetes:

- El paquete DBMS_ALERT soporta notificación asincrónica de eventos de base de datos. Los mensajes o las alertas se envían en un comando COMMIT.
- El paquete DBMS_LOCK se utiliza para solicitar, convertir y liberar bloqueos a través de los servicios de gestión de bloqueos de Oracle.
- El paquete DBMS_SESSION permite la utilización programática de la sentencia SQL ALTER SESSION y de otros comandos a nivel de sesión.
- El paquete DBMS_OUTPUT proporciona la depuración y el almacenamiento en búfer de los datos de texto.
- El paquete HTP escribe datos con etiquetas HTML en los buffers de la base de datos.
- El paquete UTL_FILE permite la lectura y la escritura de archivos de texto del sistema operativo.
- El paquete UTL_MAIL permite redactar y enviar mensajes de correo electrónico.
- El paquete DBMS_SCHEDULER permite la planificación y ejecución automática de bloques PL/SQL, procedimientos almacenados y procedimientos externos o ejecutables.

Funcionamiento del Paquete DBMS_OUTPUT

El paquete DBMS_OUTPUT le permite enviar mensajes desde subprogramas almacenados y disparadores.

- **PUT y PUT_LINE colocan el texto en el buffer.**
- **GET_LINE y GET_LINES leen el buffer.**
- **Los mensajes no se envían hasta que el remitente termine.**
- **Utilice SET SERVEROUTPUT ON para mostrar los mensajes en iSQL*Plus.**



Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso del Paquete DBMS_OUTPUT

El paquete DBMS_OUTPUT envía mensajes de texto desde cualquier bloque PL/SQL a un buffer de la base de datos. Los procedimientos que proporciona el paquete incluyen:

- PUT para agregar texto desde el procedimiento a la línea actual del buffer de salida de línea
- NEW_LINE para poner un marcador de final de línea en el buffer de salida
- PUT_LINE para combinar la acción de PUT y NEW_LINE para recortar los espacios iniciales
- GET_LINE para recuperar la línea actual desde el buffer a una variable de procedimiento
- GET_LINES para recuperar una matriz de líneas en una variable de matriz de procedimiento
- ENABLE/DISABLE para activar o desactivar llamadas a los procedimientos DBMS_OUTPUT

El tamaño del buffer se puede definir utilizando:

- La opción SIZE n agregada al comando SET SERVEROUTPUT ON, donde n está entre 2.000 (por defecto) y 1.000.000 (1 millón de caracteres)
- Un parámetro entero entre 2.000 y 1.000.000 en el procedimiento ENABLE

Uso del Paquete DBMS_OUTPUT (continuación)

Usos Prácticos

- Puede hacer que la salida de los resultados sea en la ventana para la depuración.
- Puede rastrear la ruta de acceso de ejecución del código para una función o un procedimiento.
- Puede enviar mensajes entre subprogramas y disparadores.

Nota: No hay ningún mecanismo para vaciar la salida durante la ejecución de un procedimiento.

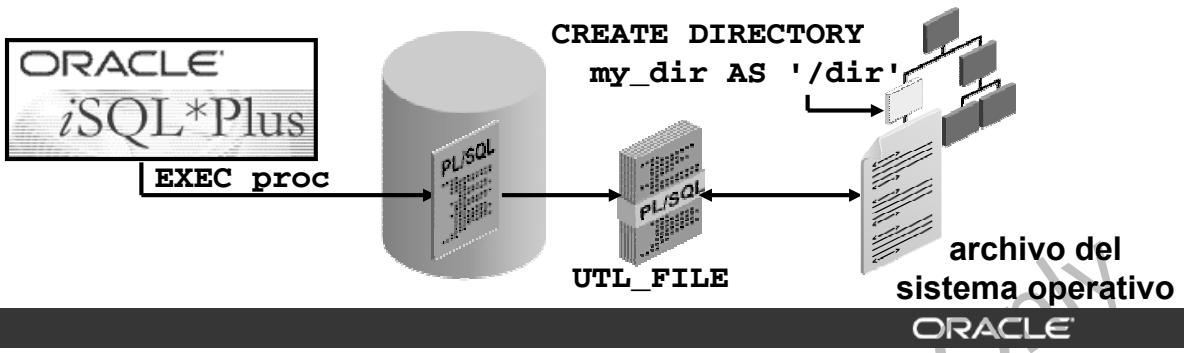
Oracle Internal & OAI Use Only

Interacción con los Archivos del Sistema Operativo

El paquete **UTL_FILE** amplía los programas PL/SQL para leer y escribir archivos de texto del sistema operativo.

UTL_FILE:

- Proporciona una versión restringida de E/S del archivo de flujo del sistema operativo para archivos de texto
- Puede acceder a los archivos de los directorios del sistema operativo definidos por una sentencia CREATE DIRECTORY. También puede utilizar el parámetro de la base de datos **utl_file_dir**.



Copyright © 2004, Oracle. Todos los Derechos Reservados.

Interacción con los Archivos del Sistema Operativo

El paquete UTL_FILE proporcionado por Oracle se utiliza para acceder a los archivos de texto del sistema operativo del servidor de base de datos. La base de datos proporciona acceso de lectura y escritura a directorios de sistema operativo específicos utilizando:

- Una sentencia CREATE DIRECTORY que asocia un alias a un directorio del sistema operativo. Al alias del directorio de la base de datos se le pueden otorgar los privilegios READ y WRITE para controlar el tipo de acceso a los archivos del sistema operativo.

Por ejemplo:

```
CREATE DIRECTORY my_dir AS '/temp/my_files';
GRANT READ, WRITE ON my_dir TO public.
```

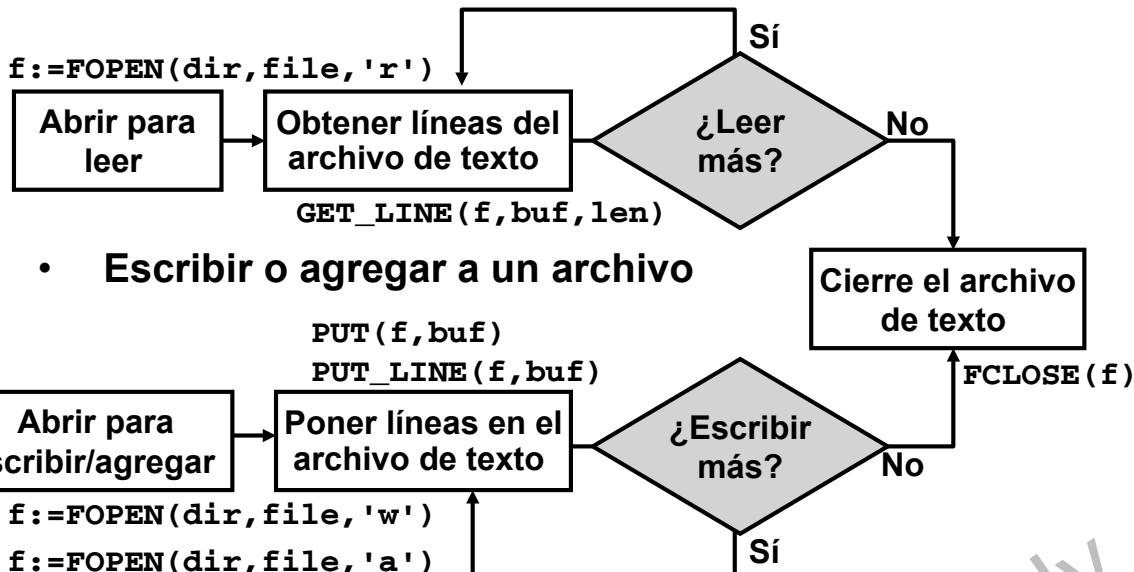
- Las rutas de acceso especificadas en el parámetro de inicialización de la base de datos **utl_file_dir**

El enfoque preferido consiste en utilizar el alias de directorio creado por la sentencia CREATE DIRECTORY, que no necesita que se reinicie la base de datos. Los directorios del sistema operativo especificados utilizando alguna de estas técnicas deberían ser accesibles y estar en la misma máquina que los procesos de servidor de la base de datos. Los nombres de las rutas de acceso (al directorio) pueden ser sensibles a mayúsculas/minúsculas en algunos sistemas operativos.

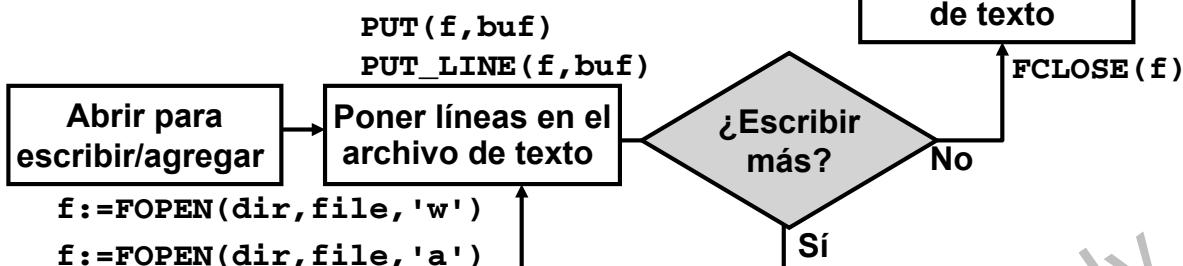
Nota: El paquete DBMS_LOB se puede utilizar para leer archivos binarios en el sistema operativo. DBMS_LOB se aborda en la lección titulada "Manipulación de Objetos Grandes".

Procesamiento de Archivos con el Paquete UTL_FILE

- Leer un archivo



- Escribir o agregar a un archivo



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Procesamiento de Archivos con el Paquete UTL_FILE

Con los procedimientos y funciones del paquete UTL_FILE, abra los archivos con la función FOPEN. A continuación, lea, escriba o agregue un elemento al archivo hasta que el procesamiento se haya terminado. Al terminar el procesamiento del archivo, ciérrelo utilizando el procedimiento FCLOSE. Los subprogramas son los siguientes:

- La función FOPEN abre un archivo en el directorio especificado para la entrada/salida (E/S) y devuelve un manejador de archivo utilizado en operaciones E/S posteriores.
- La función IS_OPEN devuelve un valor booleano cada vez que un manejador de archivo hace referencia a un archivo abierto. Utilice IS_OPEN para comprobar si el archivo ya está abierto antes de abrirlo.
- El procedimiento GET_LINE lee una línea de texto del archivo en un parámetro del buffer de salida. (El tamaño máximo de registro de entrada es de 1.023 bytes a menos que especifique un tamaño mayor en la versión sobrecargada de FOPEN.)
- Los procedimientos PUT y PUT_LINE escriben texto en el archivo abierto.
- El procedimiento PUTF proporciona una salida formateada con dos especificadores de formato: %s para sustituir un valor en la cadena de salida y \n para un carácter de nueva línea.
- El procedimiento NEW_LINE termina una línea en un archivo de salida.

Procesamiento de Archivos con el Paquete UTL_FILE (continuación)

- El procedimiento FFLUSH escribe en un archivo todos los datos almacenados en buffer en la memoria.
- El procedimiento FCLOSE cierra un archivo abierto.
- El procedimiento FCLOSE_ALL cierra todos los manejadores de archivos abiertos para la sesión.

Oracle Internal & OAI Use Only

Excepciones en el Paquete UTL_FILE

Puede que tenga que manejar una de estas excepciones al utilizar los subprogramas UTL_FILE:

- **INVALID_PATH**
- **INVALID_MODE**
- **INVALID_FILEHANDLE**
- **INVALID_OPERATION**
- **READ_ERROR**
- **WRITE_ERROR**
- **INTERNAL_ERROR**

La otra excepción que no está en el paquete UTL_FILE es:

- **NO_DATA_FOUND y VALUE_ERROR**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Excepciones en el Paquete UTL_FILE

El paquete UTL_FILE declara siete excepciones que indican una condición de error en el procesamiento del archivo del sistema operativo. Las excepciones de UTL_FILE son:

- INVALID_PATH si la ubicación del archivo o su nombre no son válidos
- INVALID_MODE si el parámetro OPEN_MODE en FOPEN no es válido
- INVALID_HANDLE si el manejador del archivo no es válido
- INVALID_OPERATION si el archivo no se ha podido abrir o no se ha podido operar en él como se ha solicitado
- READ_ERROR si se ha producido un error en el sistema operativo durante la operación de lectura
- WRITE_ERROR si se ha producido un error en el sistema operativo durante la operación de escritura
- INTERNAL_ERROR si se ha producido un error no especificado en PL/SQL

Nota: Estas excepciones siempre deben estar precedidas del nombre del paquete. Los procedimientos UTL_FILE también pueden emitir excepciones PL/SQL predefinidas tales como NO_DATA_FOUND o VALUE_ERROR.

La excepción NO_DATA_FOUND se emite cuando se lee más allá del final de un archivo utilizando UTL_FILE.GET_LINE o UTL_FILE.GET_LINES.

Parámetros de Función FOPEN e IS_OPEN

```
FUNCTION FOPEN (location IN VARCHAR2,
                filename IN VARCHAR2,
                open_mode IN VARCHAR2)
RETURN UTL_FILE.FILE_TYPE;
```

```
FUNCTION IS_OPEN (file IN FILE_TYPE)
RETURN BOOLEAN;
```

Ejemplo:

```
PROCEDURE read(dir VARCHAR2, filename VARCHAR2) IS
    file UTL_FILE.FILE_TYPE;
BEGIN
    IF NOT UTL_FILE.IS_OPEN(file) THEN
        file := UTL_FILE.FOPEN (dir, filename, 'r');
    END IF; ...
END read;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Parámetros de Función FOPEN e IS_OPEN

Entre estos parámetros se incluyen:

- Parámetro `location`: Especifica el nombre de un alias de directorio definido por una sentencia CREATE DIRECTORY o una ruta de acceso específica de un sistema operativo utilizando el parámetro de base de datos `utl_file_dir`.
- Parámetro `filename`: Especifica el nombre del archivo, incluyendo la extensión, sin ninguna información sobre la ruta de acceso.
- Cadena `open_mode`: Especifica cómo se debe abrir el archivo. Los valores son:
 - '`r`' para leer el texto (utilice `GET_LINE`)
 - '`w`' para escribir el texto (`PUT`, `PUT_LINE`, `NEW_LINE`, `PUTF`, `FFLUSH`)
 - '`a`' para agregar texto (`PUT`, `PUT_LINE`, `NEW_LINE`, `PUTF`, `FFLUSH`)

El valor de retorno de `FOPEN` es un manejador de archivo cuyo tipo es `UTL_FILE.FILE_TYPE`. El manejador se debe utilizar en llamadas posteriores a rutinas que funcionan en el archivo abierto.

El parámetro de función `IS_OPEN` es el manejador de archivo. La función `IS_OPEN` prueba un manejador de archivo para ver si identifica un archivo abierto. Devuelve un valor booleano de `TRUE` si se ha abierto el archivo, de lo contrario devuelve un valor de `FALSE` e indica que el archivo no se ha abierto. El ejemplo de la transparencia muestra cómo combinar la utilización de los dos subprogramas.

Nota: Para obtener la sintaxis completa, consulte *PL/SQL Packages and Types Reference 10g*.

Uso de UTL_FILE: Ejemplo

```
CREATE OR REPLACE PROCEDURE sal_status(
    dir IN VARCHAR2, filename IN VARCHAR2) IS
    file UTL_FILE.FILE_TYPE;
CURSOR empc IS
    SELECT last_name, salary, department_id
    FROM employees ORDER BY department_id;
    newdeptno employees.department_id%TYPE;
    olddeptno employees.department_id%TYPE := 0;
BEGIN
    file:= UTL_FILE.FOPEN (dir, filename, 'w');
    UTL_FILE.PUT_LINE(file,
        'REPORT: GENERATED ON ' || SYSDATE);
    UTL_FILE.NEW_LINE (file); ...
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de UTL_FILE: Ejemplo

En el ejemplo, el procedimiento `sal_status` crea un informe de los empleados de cada departamento, junto con sus salarios. Los datos se escriben en un archivo de texto utilizando el paquete `UTL_FILE`. En el ejemplo de código, la variable `file` se declara como `UTL_FILE.FILE_TYPE`, un tipo de paquete que es un registro con un campo denominado `ID` del tipo de dato `BINARY_INTEGER`. Por ejemplo:

```
TYPE file_type IS RECORD (id BINARY_INTEGER);
```

El campo del registro `FILE_TYPE` es privado para el paquete `UTL_FILE` y nunca se debe hacer referencia al mismo ni cambiarlo. El procedimiento `sal_status` acepta dos parámetros:

- El parámetro `dir` para el nombre del directorio en el que escribir el archivo de texto
- El parámetro `filename` para especificar el nombre del archivo

Por ejemplo, para llamar al procedimiento utilice:

```
EXECUTE sal_status ('MY_DIR', 'salreport.txt')
```

Nota: La ubicación de directorio utilizada (`MY_DIR`) debe estar en mayúsculas si es un alias de directorio creado por una sentencia `CREATE DIRECTORY`. Al leer un archivo en un bucle, éste debe salir cuando detecte la excepción `NO_DATA_FOUND`. La salida `UTL_FILE` se envía de forma síncrona. Los procedimientos `DBMS_OUTPUT` no producen salida hasta que el procedimiento se haya terminado.

Uso de UTL_FILE: Ejemplo

```
FOR emp_rec IN empc LOOP
    IF emp_rec.department_id <> olddeptno THEN
        UTL_FILE.PUT_LINE (file,
            'DEPARTMENT: ' || emp_rec.department_id);
    END IF;
    UTL_FILE.PUT_LINE (file,
        'EMPLOYEE: ' || emp_rec.last_name ||
        ' earns: ' || emp_rec.salary);
    olddeptno := emp_rec.department_id;
END LOOP;
UTL_FILE.PUT_LINE(file,'*** END OF REPORT ***');
UTL_FILE.FCLOSE (file);
EXCEPTION
    WHEN UTL_FILE.INVALID_FILEHANDLE THEN
        RAISE_APPLICATION_ERROR(-20001,'Invalid File.');
    WHEN UTL_FILE.WRITE_ERROR THEN
        RAISE_APPLICATION_ERROR (-20002, 'Unable to
write to file');
END sal_status;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

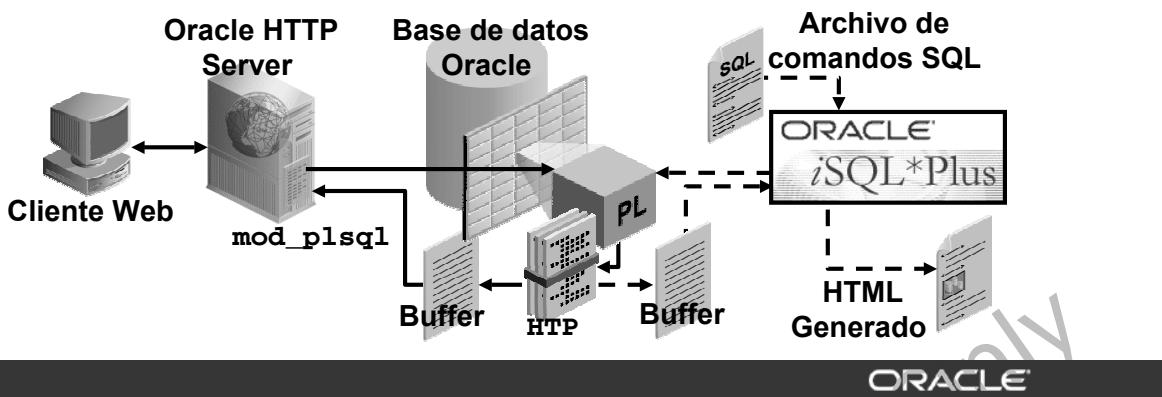
Uso de UTL_FILE: Ejemplo (continuación)

La salida para este informe en el archivo salreport.txt file es la siguiente:

```
SALARY REPORT: GENERATED ON 08-MAR-01
DEPARTMENT: 10
    EMPLOYEE: Whalen earns: 4400
DEPARTMENT: 20
    EMPLOYEE: Hartstein earns: 13000
    EMPLOYEE: Fay earns: 6000
DEPARTMENT: 30
    EMPLOYEE: Raphaely earns: 11000
    EMPLOYEE: Khoo earns: 3100
    ...
DEPARTMENT: 100
    EMPLOYEE: Greenberg earns: 12000
    ...
DEPARTMENT: 110
    EMPLOYEE: Higgins earns: 12000
    EMPLOYEE: Gietz earns: 8300
    EMPLOYEE: Grant earns: 7000
*** END OF REPORT ***
```

Generación de Páginas Web con el Paquete HTP

- Los procedimientos del paquete HTP generan etiquetas HTML.
- El paquete HTP se utiliza para generar documentos HTML de forma dinámica y se puede llamar desde:
 - Un explorador que utilice los servicios Oracle HTTP Server y gateway PL/SQL (`mod_plsql`)
 - Un archivo de comandos *iSQL*Plus* para mostrar la salida HTML



Copyright © 2004, Oracle. Todos los Derechos Reservados.

Generación de Páginas Web con el Paquete HTP

El paquete HTP contiene procedimientos que se utilizan para generar etiquetas HTML. Las etiquetas HTML que se generan normalmente delimitan los datos proporcionados como parámetros de los diferentes procedimientos. La transparencia muestra dos formas de utilizar el paquete HTP:

- Es muy probable que los servicios del gateway PL/SQL llamen a los procedimientos, a través del componente `mod_plsql` proporcionado con Oracle HTTP Server que forma parte de Oracle Application Server (representado por las líneas sólidas del gráfico).
- Asimismo (como muestran las líneas discontinuas del gráfico), *iSQL*Plus* puede llamar al procedimiento que puede mostrar la salida HTML generada, que se puede copiar y pegar en un archivo. Esta técnica se utiliza en este curso porque el software de Oracle Application Server no se instala como parte del entorno del curso.

Nota: Los procedimientos HTP extraen la información a un buffer de sesión contenido en el servidor de base de datos. En el contexto de Oracle HTTP Server, cuando termina el procedimiento, el componente `mod_plsql` recibe automáticamente el contenido del buffer, que después se devuelve al explorador como respuesta HTTP. En *SQL*Plus*, se debe ejecutar manualmente:

- Un comando `SET SERVEROUTPUT ON`
- El procedimiento para generar HTML en el buffer
- El procedimiento `OWA_UTIL.SHOWPAGE` para mostrar el contenido del buffer

Uso de los Procedimientos de Paquete HTP

- Generar una o más etiquetas HTML. Por ejemplo:

```
htp.bold('Hello');           -- <B>Hello</B>
htp.print('Hi <B>World</B>'); -- Hi <B>World</B>
```

- Utilizado para crear un documento HTML con formato correcto:

<pre>BEGIN htp.htmlOpen; -----> htp.headOpen; -----> htp.title('Welcome'); --> htp.headClose; -----> htp.bodyOpen; -----> htp.print('My home page'); htp.bodyClose; -----> htp.htmlClose; -----> END;</pre>	-- Generates: <pre><HTML> <HEAD> <TITLE>Welcome</TITLE> </HEAD> <BODY> My home page </BODY> </HTML></pre>
--	--

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de los Procedimientos de Paquete HTP

El paquete HTP está estructurado para proporcionar una asignación uno a uno de un procedimiento para etiquetas HTML estándar. Por ejemplo, para mostrar texto en negrita en una página Web, éste debe estar delimitado por el par de etiquetas HTML `` y ``. En el primer recuadro de código de la transparencia se muestra cómo generar la palabra Hello en negrita en HTML utilizando el procedimiento empaquetado `htp` equivalente, es decir, `HTP.BOLD`. El procedimiento `HTP.BOLD` acepta un parámetro y garantiza que esté delimitado por las etiquetas HTML apropiadas en la salida HTML que se genera.

El procedimiento `HTP.PRINT` copia el parámetro de texto en el buffer. El ejemplo de la transparencia muestra cómo el parámetro del procedimiento `HTP.PRINT` puede contener etiquetas HTML. Esta técnica se recomienda sólo si necesita utilizar etiquetas HTML que no se pueden generar utilizando el juego de procedimientos proporcionado en el paquete HTP.

El segundo ejemplo de la transparencia proporciona un bloque PL/SQL que genera la forma básica de un documento HTML. El ejemplo ilustra cómo cada procedimiento genera la correspondiente línea HTML en el recuadro de texto delimitado a la derecha.

La ventaja de utilizar el paquete HTP es que podrá crear documentos HTML con formato correcto, evitando así escribir las etiquetas HTML de forma manual con cada dato.

Nota: Para obtener más información sobre todos los procedimientos del paquete HTP, consulte el manual *PL/SQL Packages and Types Reference 10g*.

Creación de un Archivo HTML con iSQL*Plus

Para crear un archivo HTML con iSQL*Plus, realice los siguientes pasos:

1. Cree un archivo de comandos SQL con los siguientes comandos:

```
SET SERVEROUTPUT ON
ACCEPT procname PROMPT "Procedure: "
EXECUTE &procname
EXECUTE owa_util.showpage
UNDEFINE proc
```

2. Cargue y ejecute el archivo de comandos en iSQL*Plus y proporcione valores para las variables de sustitución.
3. Seleccione, copie y pegue el texto HTML que se genera en el explorador en un archivo HTML.
4. Abra el archivo HTML en un explorador.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un Archivo HTML con iSQL*Plus

El ejemplo de la transparencia muestra los pasos para generar HTML utilizando cualquier procedimiento y guardando la salida en un archivo HTML. Realice los siguientes pasos:

1. Active la salida del servidor con el comando `SET SERVEROUTPUT ON`. De lo contrario, obtendrá mensajes de excepción cuando ejecute los procedimientos que tengan llamadas al paquete HTP.
2. Ejecute el procedimiento que contiene llamadas al paquete HTP.
Nota: Esto *no* producirá ninguna salida, a menos que el procedimiento tenga llamadas al paquete DBMS_OUTPUT.
3. Ejecute el procedimiento OWA_UTIL.SHOWPAGE para mostrar el texto. Esta llamada realmente muestra el contenido HTML que se genera desde el buffer.

El comando ACCEPT solicita el nombre del procedimiento que se va a ejecutar. La llamada a OWA_UTIL.SHOWPAGE muestra las etiquetas HTML en la ventana del explorador.

A continuación podrá copiar y pegar las etiquetas HTML generadas desde la ventana del explorador al archivo HTML, normalmente con una extensión .htm o .html.

Nota: Si se utiliza SQL*Plus, se puede usar el comando SPOOL para dirigir la salida HTML directamente a un archivo HTML. El comando SPOOL no está soportado en iSQL*Plus; por lo tanto, la única opción es la técnica de copiar y pegar.

Uso de UTL_MAIL

El paquete UTL_MAIL:

- **Es una utilidad para gestionar correo electrónico que incluye funciones de correo electrónico utilizadas normalmente como anexos, copia, copia oculta y confirmación de recibo**
- **Necesita que el parámetro de inicialización de la base de datos SMTP_OUT_SERVER esté definido**
- **Proporciona los siguientes procedimientos:**
 - **SEND para mensajes sin anexos**
 - **SEND_ATTACH_RAW para mensajes con anexos binarios**
 - **SEND_ATTACH_VARCHAR2 para mensajes con anexos de texto**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de UTL_MAIL

El paquete UTL_MAIL es una utilidad para gestionar correo electrónico que incluye funciones de correo electrónico utilizadas normalmente como anexos, copia, copia oculta y confirmación de recibo.

El paquete UTL_MAIL no se instala por defecto debido a los requisitos de configuración de SMTP_OUT_SERVER y los problemas de seguridad que implica. Al instalar UTL_MAIL, debe tomar medidas para evitar que el puerto definido por SMTP_OUT_SERVER se vea desbordado por la transmisión de datos. Para instalar UTL_MAIL, conéctese como usuario DBA en iSQL*Plus y ejecute los siguientes archivos de comandos:

```
@$ORACLE_HOME/rdbms/admin/utlmail.sql  
@$ORACLE_HOME/rdbms/admin/prvtmail.plb
```

Debe definir el parámetro SMTP_OUT_SERVER en el archivo de inicialización de la base de datos init.ora file:

```
SMTP_OUT_SERVER=mystmpserver.mydomain.com
```

El parámetro SMTP_OUT_SERVER especifica el host y el puerto SMTP a los que UTL_MAIL entrega correo electrónico saliente. Se pueden especificar varios servidores separados por comas. Si el primer servidor de la lista no está disponible, UTL_MAIL intenta el segundo y así sucesivamente. Si SMTP_OUT_SERVER no está definido, se llama a un valor por defecto derivado de DB_DOMAIN, que es un parámetro de inicialización de la base de datos que especifica la ubicación lógica de la base de datos dentro de la estructura de la red. Por ejemplo:

```
db_domain=mydomain.com
```

Instalación y Uso de UTL_MAIL

- **Como SYSDBA, con iSQL*Plus:**
 - Defina SMTP_OUT_SERVER (necesita reiniciar DBMS).

```
ALTER SYSTEM SET SMTP_OUT_SERVER='smtp.server.com'  
SCOPE=SPFILE
```

- Instale el paquete UTL_MAIL.

```
@?/rdbms/admin/utlmail.sql  
@?/rdbms/admin/prvtmail.plb
```

- **Como desarrollador, llame al procedimiento UTL_MAIL:**

```
BEGIN  
    UTL_MAIL.SEND('otn@oracle.com', 'user@oracle.com',  
                  message => 'For latest downloads visit OTN',  
                  subject => 'OTN Newsletter');  
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Instalación y Uso de UTL_MAIL

La transparencia muestra cómo configurar el parámetro SMTP_OUT_SERVER en el nombre del host SMTP en la red y cómo instalar el paquete UTL_MAIL que no se instala por defecto. Si cambia SMTP_OUT_SERVER tendrá que reiniciar la instancia de la base de datos. Estas tareas las realiza un usuario con capacidades SYSDBA.

El último ejemplo de la transparencia muestra el modo más sencillo de enviar un mensaje de texto utilizando el procedimiento UTL_MAIL.SEND con al menos un asunto y un mensaje. Los dos primeros parámetros necesarios son los siguientes:

- La dirección de correo electrónico de sender (en este caso, otn@oracle.com)
- La dirección de correo electrónico de recipients (por ejemplo, user@oracle.com). El valor puede ser una lista de direcciones separadas por comas.

El procedimiento UTL_MAIL.SEND proporciona varios parámetros más, tales como cc, bcc y priority con valores por defecto si no se especifican. En el ejemplo, el parámetro message especifica el texto para el correo electrónico y el parámetro del asunto contiene el texto de la línea de asunto. Para enviar un mensaje con etiquetas HTML, agregue el parámetro mime_type (por ejemplo, mime_type=>'text/html').

Nota: Para obtener más información sobre todos los parámetros de procedimiento UTL_MAIL, consulte el manual *PL/SQL Packages and Types Reference 10g*.

Envío de Correo Electrónico con Anexos Binarios

Utilice el procedimiento UTL_MAIL.SEND_ATTACH_RAW:

```
CREATE OR REPLACE PROCEDURE send_mail_logo IS
BEGIN
    UTL_MAIL.SEND_ATTACH_RAW(
        sender => 'me@oracle.com',
        recipients => 'you@somewhere.net',
        message =>
            '<HTML><BODY>See attachment</BODY></HTML>',
        subject => 'Oracle Logo',
        mime_type => 'text/html'
        attachment => get_image('oracle.gif'),
        att_inline => true,
        att_mime_type => 'image/gif',
        att_filename => 'orologo.gif');
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Envío de Correo Electrónico con Anexos Binarios

La transparencia muestra un procedimiento denominado UTL_MAIL.SEND_ATTACH_RAW para enviar un mensaje de texto o HTML con un anexo binario. Además de los parámetros sender, recipients, message, subject y mime_type que proporcionan valores para la mayor parte del mensaje de correo electrónico, el procedimiento SEND_ATTACH_RAW tiene los siguientes parámetros resaltados:

- El parámetro attachment (necesario) acepta un tipo de dato RAW con un tamaño máximo de 32.767 caracteres binarios.
- El parámetro att_inline(opcional) es booleano (valor por defecto TRUE) para indicar que el anexo se puede ver con el cuerpo del mensaje.
- El parámetro att_mime_type (opcional) especifica el formato del anexo. Si no se ha proporcionado, se define en application/octet.
- El parámetro att_filename (opcional) le asigna cualquier nombre de archivo al anexo. Es NULL por defecto, en cuyo caso al nombre se le asigna un nombre por defecto.

La función get_image del ejemplo utiliza BFILE para leer los datos de la imagen. Si utiliza BFILE necesita crear un nombre de directorio lógico en la base de datos utilizando la sentencia CREATE DIRECTORY. Los detalles de cómo trabajar con BFILE se tratan en la lección titulada “Manipulación de Objetos Grandes”. El código para get_image se muestra en la página siguiente.

Envío de Correo Electrónico con Anexos Binarios (continuación)

La función `get_image` utiliza el paquete DBMS_LOB para leer un archivo binario del sistema operativo:

```
CREATE OR REPLACE FUNCTION get_image(
    filename VARCHAR2, dir VARCHAR2 := 'TEMP')
RETURN RAW IS
    image RAW(32767);
    file BFILE := BFILENAME(dir, filename);
BEGIN
    DBMS_LOB.FILEOPEN(file, DBMS_LOB.FILE_READONLY);
    image := DBMS_LOB.SUBSTR(file);
    DBMS_LOB.CLOSE(file);
    RETURN image;
END;
/
```

Para crear el directorio denominado TEMP, ejecute la siguiente sentencia de iSQL*Plus:

```
CREATE DIRECTORY temp AS 'e:\temp';
```

Nota: Necesita el privilegio del sistema CREATE ANY DIRECTORY para ejecutar esta sentencia.

Oracle Internal & OAI Use Only

Envío de Correo Electrónico con Anexos de Texto

Utilice el procedimiento

UTL_MAIL.SEND_ATTACH_VARCHAR2:

```
CREATE OR REPLACE PROCEDURE send_mail_file IS
BEGIN
    UTL_MAIL.SEND_ATTACH_VARCHAR2(
        sender => 'me@oracle.com',
        recipients => 'you@somewhere.net',
        message =>
            '<HTML><BODY>See attachment</BODY></HTML>',
        subject => 'Oracle Notes',
        mime_type => 'text/html'
        attachment => get_file('notes.txt'),
        att_inline => false,
        att_mime_type => 'text/plain',
        att_filename => 'notes.txt');
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Envío de Correo Electrónico con Anexos de Texto

La transparencia muestra un procedimiento que llama al procedimiento UTL_MAIL.SEND_ATTACH_VARCHAR2 para enviar un mensaje de texto o HTML con un anexo de texto. Además de los parámetros sender, recipients, message, subject y mime_type que proporcionan valores para la mayor parte del mensaje de correo electrónico, el procedimiento SEND_ATTACH_VARCHAR2 tiene los siguientes parámetros resaltados:

- El parámetro attachment (necesario) acepta un tipo de dato VARCHAR2 con un tamaño máximo de 32767 caracteres binarios.
- El parámetro att_inline(opcional) es booleano (valor por defecto TRUE) para indicar que el anexo se puede ver con el cuerpo del mensaje.
- El parámetro att_mime_type (opcional) especifica el formato del anexo. Si no se ha proporcionado, se define en application/octet.
- El parámetro att_filename (opcional) le asigna cualquier nombre de archivo al anexo. Es NULL por defecto, en cuyo caso al nombre se le asigna un nombre por defecto.

La función get_file del ejemplo utiliza BFILE para leer un archivo de texto de los directorios del sistema operativo para el valor del parámetro attachment, que se podría llenar simplemente desde una variable VARCHAR2. El código para get_file se muestra en la página siguiente.

Envío de Correo Electrónico con Anexos de Texto (continuación)

La función `get_file` utiliza el paquete DBMS_LOB para leer un archivo binario del sistema operativo y utiliza el paquete UTL_RAW para convertir los datos binarios RAW en datos de texto legibles en forma de un tipo de dato VARCHAR2:

```
CREATE OR REPLACE FUNCTION get_file(
    filename VARCHAR2, dir VARCHAR2 := 'TEMP')
RETURN VARCHAR2 IS
    contents VARCHAR2(32767);
    file BFILE := BFILENAME(dir, filename);
BEGIN
    DBMS_LOB.FILEOPEN(file, DBMS_LOB.FILE_READONLY);
    contents := UTL_RAW.CAST_TO_VARCHAR2(
        DBMS_LOB.SUBSTR(file));
    DBMS_LOB.CLOSE(file);
    RETURN contents;
END;
/
```

Nota: Asimismo, puede leer el contenido del archivo de texto en una variable VARCHAR2 utilizando la funcionalidad del paquete UTL_FILE.

El ejemplo anterior necesita que el directorio TEMP se cree de forma similar a la siguiente sentencia de *iSQL*Plus*:

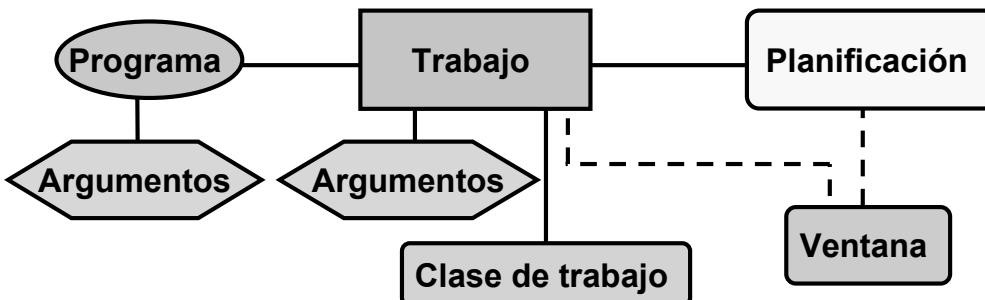
```
CREATE DIRECTORY temp AS 'e:\temp';
```

Nota: Se necesita el privilegio del sistema CREATE ANY DIRECTORY para ejecutar esta sentencia.

Paquete DBMS_SCHEDULER

El planificador de base de datos consta de varios componentes para poder ejecutar los trabajos. Utilice el paquete DBMS_SCHEDULER para crear cada trabajo con:

- **Un nombre de trabajo único**
- **Un programa (“qué” se ejecutará)**
- **Una planificación (“cuándo” se ejecutará)**



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Paquete DBMS_SCHEDULER

La base de datos Oracle 10g proporciona una recopilación de subprogramas en el paquete DBMS_SCHEDULER para simplificar la gestión y proporcionar un amplio juego de funcionalidades para tareas de planificación complejas. Colectivamente, a estos subprogramas se les denomina el planificador y se pueden llamar desde cualquier programa PL/SQL. El planificador permite a los administradores de la base de datos y a los desarrolladores de aplicaciones controlar cuándo y dónde van a tener lugar diferentes tareas. Asegurándose de que muchas tareas rutinarias de la base de datos suceden sin intervención manual, puede reducir los costes operativos, implementar rutinas más fiables y minimizar los errores humanos.

El diagrama muestra los siguientes componentes arquitectónicos del planificador:

- Un **trabajo** es la combinación de un programa y una planificación. Los argumentos que necesita el programa se pueden proporcionar con el programa o con el trabajo. Todos los nombres de trabajos siguen el formato [schema .] name. Al crear un trabajo, se especifica el nombre del trabajo, un programa, una planificación y (opcionalmente) las características del trabajo que se pueden proporcionar con una **clase de trabajo**.
- Un **programa** determina “qué” se ejecutará. Cada trabajo automático implica un ejecutable concreto, ya sea un bloque PL/SQL, un procedimiento almacenado, un ejecutable binario nativo o un archivo de comandos del shell. Un programa proporciona metadatos sobre un ejecutable concreto y puede necesitar una lista de argumentos.

Paquete DBMS_SCHEDULER (continuación)

- Una **planificación** especifica cuándo y cuántas veces se debe ejecutar un trabajo.
- Una **clase de trabajo** define una categoría de trabajos que comparten requisitos comunes de uso de recursos y otras características. Los trabajos no pueden pertenecer a varias clases de trabajo a la vez, sino sólo a una. Una clase de trabajo tiene los siguientes atributos:
 - Un nombre de **servicio** de la base de datos. Los trabajos de la clase de trabajo serán afines al servicio concreto que se especifique. Es decir, los trabajos se ejecutarán en las instancias que se encargan del servicio especificado.
 - Un **grupo de consumidores de recursos**, que clasifica un juego de sesiones de usuario con requisitos comunes de procesamiento de recursos. La sesión de usuario o la clase de trabajo pertenece en todo momento sólo a un grupo de consumidores de recursos. El grupo de consumidores de recursos con el que se asocia la clase de trabajo determina los recursos que se asignan a la clase de trabajo.
- Una **ventana** está representada por un intervalo de tiempo con un inicio y un fin claramente definidos y se utiliza para activar diferentes planes de recursos en momentos distintos.

La transparencia se centra en el componente del trabajo como entidad primaria. Sin embargo, un programa, planificación, ventana y clase de trabajo son componentes que se pueden crear como entidades individuales que se pueden asociar a un trabajo para que lo ejecute el planificador. Cuando se crea un trabajo, puede contener toda la información necesaria en línea, es decir, en la llamada que crea el trabajo. Asimismo, la creación de un trabajo puede hacer referencia a un programa o componente de la planificación definido previamente. Algunos ejemplos se presentarán en las páginas siguientes.

Para obtener más información sobre el planificador, consulte el curso de eStudy titulado *Oracle Database 10g: Configure and Manage Jobs with the Scheduler*.

Creación de un Trabajo

Un trabajo se puede crear de varias maneras utilizando una combinación de parámetros en línea `Programs` y `Schedules` con nombre. Para crear un trabajo con el procedimiento `CREATE_JOB`:

- Utilice la información en línea con el “qué” y la planificación especificados como parámetros
- Utilice un programa con nombre (guardado) y especifique la planificación en línea
- Especifique lo que se debe realizar en línea y utilice una planificación con nombre
- Utilice componentes de programa y planificación con nombre

Nota: Para crear un trabajo es necesario el privilegio del sistema `CREATE_JOB`.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un Trabajo

El componente que hace que algo se ejecute en un momento determinado se denomina **trabajo**. Utilice el procedimiento `DBMS_SCHEDULER.CREATE_JOB` del paquete `DBMS_SCHEDULER` para crear un trabajo, que está desactivado por defecto. Un trabajo se activa y se planifica si se activa explícitamente. Para crear un trabajo:

- Proporcione un nombre con el formato [schema.] name
- Necesita el privilegio `CREATE_JOB`

Nota: Un usuario con el privilegio `CREATE ANY JOB` puede crear un trabajo en cualquier esquema excepto en el esquema `SYS`. Para asociar un trabajo a una clase concreta se necesita el privilegio `EXECUTE` para esa clase.

Es decir, un trabajo se puede crear especificando todos los detalles del mismo: el programa que hay que ejecutar (qué) y su planificación (cuándo), en los argumentos del procedimiento `CREATE_JOB`. Asimismo, puede utilizar componentes de programa y planificación predefinidos. Si tiene un programa y una planificación con nombre, se pueden especificar o combinar con argumentos en línea para una máxima flexibilidad en el modo en que se crea el trabajo.

Se realiza una comprobación lógica simple en la información de la planificación (es decir, se comprueban los parámetros de fecha cuando se crea un trabajo). La base de datos comprueba si la fecha final aparece después de la fecha de inicio. Si la fecha de inicio hace referencia a un momento del pasado, se cambia a la fecha actual.

Creación de un Trabajo con Parámetros en Línea

Especifique el tipo de código, código, hora de inicio y la frecuencia del trabajo que han de ejecutarse en los argumentos del procedimiento CREATE_JOB.

A continuación se presenta un ejemplo que planifica un bloque PL/SQL cada hora:

```
BEGIN  
    DBMS_SCHEDULER.CREATE_JOB(  
        job_name => 'JOB_NAME',  
        job_type => 'PLSQL_BLOCK',  
        job_action => 'BEGIN ...; END;',  
        start_date => SYSTIMESTAMP,  
        repeat_interval=>'FREQUENCY=HOURLY; INTERVAL=1',  
        enabled => TRUE);  
END;  
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un Trabajo con Parámetros en Línea

Puede crear un trabajo para ejecutar un bloque PL/SQL, procedimiento almacenado o programa externo utilizando el procedimiento DBMS_SCHEDULER.CREATE_JOB. El procedimiento CREATE_JOB se puede utilizar directamente sin que tenga que crear los componentes de programa o planificación.

El ejemplo de la transparencia muestra cómo se pueden especificar todos los detalles del trabajo en línea. Los parámetros del procedimiento CREATE_JOB definen “qué” se tiene que ejecutar, la planificación y otros atributos del trabajo. Los siguientes parámetros definen lo que se debe ejecutar:

- El parámetro job_type puede tener tres valores:
 - PLSQL_BLOCK para cualquier bloque PL/SQL o sentencia SQL. Este tipo de trabajo no puede aceptar argumentos.
 - STORED_PROCEDURE para cualquier procedimiento autónomo o empaquetado almacenado. Los procedimientos pueden aceptar argumentos que se proporcionan con el trabajo.
 - EXECUTABLE para una aplicación ejecutable del sistema operativo de línea de comandos.
- La planificación se especifica utilizando los siguientes parámetros:
 - start_date acepta un registro de hora y repeat_interval especifica una cadena como calendario o expresión PL/SQL. Se puede especificar end_date.

Nota: Las expresiones de cadena que se especifican para repeat_interval se describirán más adelante. El ejemplo especifica que el trabajo se debe ejecutar cada hora.

Creación de un Trabajo Utilizando un Programa

- Utilice **CREATE_PROGRAM** para crear un programa:

```
BEGIN
  DBMS_SCHEDULER.CREATE_PROGRAM(
    program_name => 'PROG_NAME',
    program_type => 'PLSQL_BLOCK',
    program_action => 'BEGIN ...; END;');
END;
```

- Utilice el procedimiento sobrecargado **CREATE_JOB** con el parámetro **program_name**:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB('JOB_NAME',
    program_name => 'PROG_NAME',
    start_date => SYSTIMESTAMP,
    repeat_interval => 'FREQ=DAILY',
    enabled => TRUE);
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un Trabajo Utilizando un Programa

El procedimiento DBMS_SCHEDULER.CREATE_PROGRAM define un programa al que se debe asignar un nombre único. Al crear el programa por separado para el trabajo le permitirá:

- Definir la acción una vez y volver a utilizarla con varios trabajos
- Cambiar la planificación de un trabajo sin tener que volver a crear el bloque PL/SQL
- Cambiar el programa ejecutado sin cambiar todos los trabajos

La cadena de acción del programa especifica un procedimiento, nombre ejecutable o bloque PL/SQL dependiendo del valor del parámetro **program_type**, que puede ser:

- PLSQL_BLOCK para ejecutar un bloque anónimo o una sentencia SQL
- STORED PROCEDURE para ejecutar un procedimiento almacenado, como PL/SQL, Java o C
- EXECUTABLE para ejecutar los programas de línea de comandos del sistema operativo

El ejemplo que se muestra en la transparencia muestra cómo llamar a un bloque anónimo PL/SQL. También puede llamar a un procedimiento externo dentro de un programa como en el ejemplo siguiente:

```
DBMS_SCHEDULER.CREATE_PROGRAM(program_name => 'GET_DATE',
  program_action => '/usr/local/bin/date',
  program_type => 'EXECUTABLE');
```

Para crear un trabajo con un programa, especifique el nombre del programa en el argumento **program_name** en la llamada al procedimiento DBMS_SCHEDULER.CREATE_JOB como se muestra.

Creación de un Trabajo para un Programa con Argumentos

- Crear un programa:

```
DBMS_SCHEDULER.CREATE_PROGRAM(  
    program_name => 'PROG_NAME',  
    program_type => 'STORED PROCEDURE',  
    program_action => 'EMP_REPORT');
```

- Definir un argumento:

```
DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT(  
    program_name => 'PROG_NAME',  
    argument_name => 'DEPT_ID',  
    argument_position=> 1, argument_type=> 'NUMBER',  
    default_value => '50');
```

- Crear un trabajo especificando el número de argumentos:

```
DBMS_SCHEDULER.CREATE_JOB('JOB_NAME', program_name  
=> 'PROG_NAME', start_date => SYSTIMESTAMP,  
repeat_interval => 'FREQ=DAILY',  
number_of_arguments => 1, enabled => TRUE);
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un Trabajo para un Programa con Argumentos

Puede que algunos procedimientos PL/SQL o externos necesiten argumentos de entrada. Si utiliza el procedimiento DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT, puede definir un argumento para un programa existente. Entre los parámetros del procedimiento DEFINE_PROGRAM_ARGUMENT se incluyen:

- program_name especifica un programa existente que hay que modificar.
- argument_name especifica un nombre de argumento único para el programa.
- argument_position especifica la posición en la que el argumento se transfiere cuando se llama al programa.
- argument_type especifica el tipo de dato del valor del argumento que se transfiere al programa llamado.
- default_value especifica un valor por defecto que se proporciona al programa, si el trabajo que planifica el programa no proporciona un valor.

La transparencia muestra cómo crear un trabajo ejecutando un programa con un argumento. El valor por defecto del argumento del programa es 50. Para cambiar el valor del argumento del programa, utilice:

```
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE(  
    job_name => 'JOB_NAME',  
    argument_name => 'DEPT_ID', argument_value => '80');
```

Creación de un Trabajo Utilizando una Planificación

- Utilice **CREATE_SCHEDULE** para crear una planificación:

```
BEGIN
    DBMS_SCHEDULER.CREATE_SCHEDULE( 'SCHED_NAME' ,
        start_date => SYSTIMESTAMP,
        repeat_interval => 'FREQ=DAILY',
        end_date => SYSTIMESTAMP +15);
END;
```

- Utilice **CREATE_JOB** haciendo referencia a la planificación del parámetro **schedule_name**:

```
BEGIN
    DBMS_SCHEDULER.CREATE_JOB( 'JOB_NAME' ,
        schedule_name => 'SCHED_NAME',
        job_type => 'PLSQL_BLOCK',
        job_action => 'BEGIN ...; END;',
        enabled => TRUE);
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un Trabajo Utilizando una Planificación

Puede crear una planificación común que pueda aplicarse a diferentes trabajos sin tener que especificar los detalles de la misma cada vez. Los beneficios de crear una planificación son los siguientes:

- Se puede reutilizar y asignar a varios trabajos.
- Si se cambia la planificación afecta a todos los trabajos que utilizan la planificación.
Las planificaciones del trabajo se cambian una vez, no varias.

El grado de precisión de la planificación puede alcanzar sólo el segundo más cercano. Aunque el tipo de dato TIMESTAMP es más preciso, el planificador redondea todo aquello que sea más preciso al segundo más cercano.

Las horas de inicio y fin de una planificación se especifican mediante el tipo de dato TIMESTAMP. El valor `end_date` de una planificación guardada es la fecha de vencimiento de la planificación. La planificación del ejemplo es válida para 15 días después de utilizarla con el trabajo especificado.

El valor `repeat_interval` de una planificación guardada se debe crear con una expresión de calendario. Un valor NULL para `repeat_interval` especifica que el trabajo sólo se ejecuta una vez.

Nota: No puede utilizar expresiones PL/SQL para expresar el intervalo de repetición de una planificación guardada.

Definición del Intervalo de Repetición para un Trabajo

- Uso de una expresión de calendario:

```
repeat_interval=> 'FREQ=HOURLY; INTERVAL=4'  
repeat_interval=> 'FREQ=DAILY'  
repeat_interval=> 'FREQ=MINUTELY; INTERVAL=15'  
repeat_interval=> 'FREQ=YEARLY;  
BYMONTH=MAR, JUN, SEP, DEC;  
BYMONTHDAY=15'
```

- Uso de una expresión PL/SQL:

```
repeat_interval=> 'SYSDATE + 36/24'  
repeat_interval=> 'SYSDATE + 1'  
repeat_interval=> 'SYSDATE + 15/(24*60)'
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Definición del Intervalo de Repetición para un Trabajo

Al planificar intervalos de repetición para un trabajo, puede especificar una expresión PL/SQL (si está dentro de un argumento de trabajo) o una expresión de calendario.

Los ejemplos de la transparencia son los siguientes:

- FREQ=HOURLY ; INTERVAL=4 indica un intervalo de repetición cada cuatro horas.
- FREQ=DAILY indica un intervalo de repetición diario, al mismo tiempo que la fecha de inicio de la planificación.
- FREQ=MINUTELY ; INTERVAL=15 indica un intervalo de repetición cada 15 minutos.
- FREQ=YEARLY ; BYMONTH=MAR, JUN, SEP, DEC ; BYMONTHDAY=15 indica un intervalo de repetición anual el 15 de marzo, el 15 de junio, el 15 de septiembre y el 15 de diciembre.

Con una expresión de calendario, el momento de inicio siguiente para un trabajo se calcula mediante el intervalo de repetición y la fecha de inicio del trabajo.

Nota: Si no se especifica ningún intervalo de repetición (es decir, si se proporciona un valor NULL en el argumento), el trabajo sólo se ejecuta una vez en la fecha de inicio especificada.

Creación de un Trabajo Utilizando un Programa y una Planificación con Nombre

- Cree un programa con nombre denominado **PROG_NAME** utilizando el procedimiento **CREATE_PROGRAM**.
- Cree una planificación con nombre denominado **SCHED_NAME** utilizando el procedimiento **CREATE_SCHEDULE**.
- Cree un trabajo que haga referencia al programa y a la planificación con nombre:

```
BEGIN
    DBMS_SCHEDULER.CREATE_JOB(
        'JOB_NAME',
        program_name => 'PROG_NAME',
        schedule_name => 'SCHED_NAME',
        enabled => TRUE);
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un Trabajo Utilizando un Programa y una Planificación con Nombre

El ejemplo de la transparencia muestra la última forma de utilizar el procedimiento DBMS_SCHEDULER.CREATE_JOB. En este ejemplo, el programa (PROG_NAME) y la planificación (SCHED_NAME) con nombre se especifican en los respectivos parámetros en la llamada al procedimiento DBMS_SCHEDULER.CREATE_JOB.

Con este ejemplo, puede ver lo fácil que es crear trabajos utilizando un programa y una planificación predefinidos.

Algunos trabajos y planificaciones son demasiado complejos para tratarlos en este curso. Por ejemplo, puede crear ventanas para planes de intervalos periódicos y asociar un plan de recursos a una ventana. Un plan de recursos define atributos sobre los recursos necesarios durante el período definido por la ventana de ejecución.

Para obtener más información, consulte el curso de eStudy titulado *Oracle Database 10g: Configure and Manage Jobs with the Scheduler*.

Gestión de Trabajos

- **Ejecutar un trabajo:**

```
DBMS_SCHEDULER.RUN_JOB( 'SCHEMA.JOB_NAME' );
```

- **Parar un trabajo:**

```
DBMS_SCHEDULER.STOP_JOB( 'SCHEMA.JOB_NAME' );
```

- **Borrar un trabajo, aunque se esté ejecutando actualmente:**

```
DBMS_SCHEDULER.DROP_JOB( 'JOB_NAME' , TRUE );
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Gestión de Trabajos

Una vez creado un trabajo, podrá:

- Ejecutar el trabajo llamando al procedimiento RUN_JOB especificando el nombre del trabajo. El trabajo se ejecuta inmediatamente en la sesión actual.
- Pare el trabajo utilizando el procedimiento STOP_JOB. Si el trabajo se está ejecutando actualmente, se parará de inmediato. El procedimiento STOP_JOB tiene dos argumentos:
 - job_name: Nombre del trabajo que se va a parar.
 - force: Intenta terminar el trabajo de forma correcta. Si esto falla y force está definido en TRUE, se termina el esclavo del trabajo. (El valor por defecto es FALSE.) Para utilizar force, debe tener el privilegio del sistema MANAGE_SCHEDULER.
- Borre el trabajo con el procedimiento DROP_JOB. Este procedimiento tiene dos argumentos:
 - job_name: Nombre del trabajo que se va a borrar.
 - force: Si el trabajo se debe parar y borrar en caso de estar ejecutándose en esos momentos (el valor por defecto es FALSE).

Si se llama al procedimiento DROP_JOB y el trabajo especificado se está ejecutando en esos momentos, el comando fallará, a menos que la opción de forzado se haya definido en TRUE. Si la opción de forzado se definió en TRUE, se pararán las instancias del trabajo que se estén ejecutando y se borrará el trabajo.

Nota: Para ejecutar, parar o borrar un trabajo que pertenezca a otro usuario, necesita los privilegios ALTER de ese trabajo o el privilegio del sistema CREATE ANY JOB.

Vistas de Diccionario de Datos

- [DBA | ALL | USER]_SCHEDULER_JOBS
- [DBA | ALL | USER]_SCHEDULER_RUNNING_JOBS
- [DBA | ALL]_SCHEDULER_JOB_CLASSES
- [DBA | ALL | USER]_SCHEDULER_JOB_LOG
- [DBA | ALL | USER]_SCHEDULER_JOB_RUN_DETAILS
- [DBA | ALL | USER]_SCHEDULER_PROGRAMS

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Vistas de Diccionario de Datos

La vista DBA_SCHEDULER_JOB_LOG muestra todas las instancias de trabajo terminadas, tanto las correctas como las fallidas.

Para ver el estado de los trabajos, utilice la siguiente consulta:

```
SELECT job_name, program_name, job_type, state  
FROM USER_SCHEDULER_JOBS;
```

Utilice la siguiente consulta para determinar en qué instancia se está ejecutando un trabajo:

```
SELECT owner, job_name, running_instance,  
resource_consumer_group  
FROM DBA_SCHEDULER_RUNNING_JOBS;
```

Para determinar información sobre cómo se ejecutó un trabajo, utilice la siguiente consulta:

```
SELECT job_name, instance_id, req_start_date,  
actual_start_date, end_date, completion_status  
FROM ALL_SCHEDULER_JOB_LOG;
```

Para determinar el estado de los trabajos, utilice la siguiente consulta:

```
SELECT job_name, status, error#, run_duration, cpu_used  
FROM USER_SCHEDULER_JOB_RUN_DETAILS;
```

Resumen

En esta lección, debe haber aprendido lo siguiente:

- **Utilizar varios paquetes preinstalados que proporciona el servidor de Oracle**
- **Utilizar los siguientes paquetes:**
 - **DBMS_OUTPUT para almacenar en buffer y mostrar texto**
 - **UTL_FILE para escribir archivos de texto del sistema operativo**
 - **HTP para generar documentos HTML**
 - **UTL_MAIL para enviar mensajes con anexos**
 - **DBMS_SCHEDULER para automatizar el procesamiento**
- **Crear paquetes individualmente o utilizando el archivo de comandos catproc.sql**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen

Esta lección abarca un pequeño subjuego de paquetes que se proporciona con la base de datos Oracle. Ya ha utilizado ampliamente DBMS_OUTPUT para la depuración y para mostrar información generada con procedimientos en la pantalla de iSQL*Plus.

En esta lección, debe haber aprendido a utilizar las potentes funciones que ofrece la base de datos para crear archivos de texto en el sistema operativo utilizando UTL_FILE, generar informes HTML con el paquete htp, enviar correo electrónico con o sin anexos binarios o de texto utilizando el paquete UTL_MAIL y planificar PL/SQL y código externo para su ejecución con el paquete DBMS_SCHEDULER.

Nota: Para obtener más información sobre todos los paquetes y tipos PL/SQL, consulte el manual *PL/SQL Packages and Types Reference 10g*.

Práctica 5: Visión General

En esta práctica se abordan los siguientes temas:

- **Uso de UTL_FILE para generar un informe de texto**
- **Uso de HTP para generar un informe de página Web**
- **Uso de DBMS_SCHEDULER para automatizar el procesamiento de informes**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica 5: Visión General

En esta práctica, utilizará UTL_FILE para generar un informe de archivo de texto de los empleados de cada departamento. Creará un procedimiento para generar una versión HTML del informe de empleados y escribirá un archivo de comandos SQL para enviar los resultados a un archivo de texto. Utilizará DBMS_SCHEDULER para ejecutar el primer informe que crea un archivo de texto cada 5 minutos.

Práctica 5

1. Cree un procedimiento denominado EMPLOYEE_REPORT que genera un informe de empleados en un archivo del sistema operativo utilizando el paquete UTL_FILE. Este informe generará una lista de los empleados que han excedido el salario medio de su departamento.
 - a. El programa debe aceptar dos parámetros. El primero es el directorio de salida. El segundo es el nombre del archivo de texto escrito.

Nota: Utilice el valor de la ubicación del directorio UTL_FILE. Agregue una sección de manejo de excepciones para manejar los errores que se pueden encontrar al utilizar el paquete UTL_FILE.

- b. Llame al programa, utilizando el segundo parámetro con un nombre como sal_rptxx.txt en el que xx representa el número de usuario (por ejemplo, 01, 15, etc.). A continuación se muestra una salida de ejemplo del archivo de informe:

```
Employees who earn more than average salary:  
REPORT GENERATED ON 26-FEB-04  
Hartstein 20 $13,000.00  
Raphaely 30 $11,000.00  
Marvis 40 $6,500.00  
...  
*** END OF REPORT ***
```

Nota: Los datos muestran el apellido, el identificador de departamento y el salario del empleado. Pídale al instructor que le dé instrucciones sobre cómo obtener el archivo de informe del servidor utilizando la utilidad Putty PSFTP.

2. Cree un procedimiento nuevo denominado WEB_EMPLOYEE_REPORT que genere los mismos datos que EMPLOYEE_REPORT.
 - a. En primer lugar, ejecute SET SERVEROUTPUT ON y a continuación htp.print('hello') por último, ejecute OWA_UTIL.SHOWPAGE. Los mensajes de excepción generados se pueden ignorar.
 - b. Escriba el procedimiento WEB_EMPLOYEE_REPORT con el paquete HTP para generar un informe HTML de empleados con un salario mayor que la media de su departamento. Si conoce HTML, cree una tabla HTML, de lo contrario, límítese a crear líneas de datos.
- Indicación:** Copie la definición del cursor y el bucle FOR del procedimiento EMPLOYEE_REPORT para la estructura básica del informe Web.
- c. Ejecute el procedimiento con iSQL*Plus para generar los datos HTML en un buffer del servidor y ejecute el procedimiento OWA_UTIL.SHOWPAGE para mostrar el contenido del buffer. Recuerde que SERVEROUTPUT debe estar en ON antes de ejecutar el código.
 - d. Cree un archivo HTML denominado web_employee_report.htm que contenga el texto del resultado de salida que seleccione y copie desde la etiqueta de apertura <HTML> a la etiqueta de cierre </HTML>. Pegue el texto copiado en el archivo y guárdelo en el disco. Haga clic dos veces en el archivo para mostrar los resultados en el explorador por defecto.

Práctica 5 (continuación)

3. Su jefe desea ejecutar el informe de empleados con frecuencia. Cree un procedimiento que utilice el paquete DBMS_SCHEDULER para planificar el procedimiento EMPLOYEE_REPORT para su ejecución. Debe utilizar parámetros para especificar una frecuencia y un argumento opcional para especificar los minutos tras los que un trabajo planificado debe terminar.
 - a. Cree un procedimiento denominado SCHEDULE_REPORT que proporcione los dos parámetros siguientes:
 - interval para especificar una cadena que indique la frecuencia del trabajo planificado
 - minutes para especificar el tiempo total en minutos (el valor defecto es 10) para el trabajo planificado, tras los que estará terminado. El código divide la duración por la cantidad (24 x 60) cuando se agrega a la fecha y hora actuales para especificar la hora de terminación.

Cuando el procedimiento crea un trabajo con el nombre de EMPSAL_REPORT llamándolo DBMS_SCHEDULER.CREATE_JOB, el trabajo debe estar activado y planificado para que el bloque PL/SQL se inicie inmediatamente. Debe planificar un bloque anónimo para llamar al procedimiento EMPLOYEE_REPORT para que el nombre del archivo se pueda actualizar con una nueva hora cada vez que se ejecute el informe. A EMPLOYEE_REPORT se le da el nombre de directorio proporcionado por el instructor para la tarea 1 y el parámetro del nombre de archivo se especifica con el siguiente formato:

sal_rptxx_hh24-mi-ss.txt, en el que xx es el número de usuario asignado y hh24-mi-ss representa las horas, minutos y segundos.

Utilice la siguiente variable PL/SQL local para construir un bloque PL/SQL:

```
plsql_block VARCHAR2(200) :=  
'BEGIN' ||  
  ' EMPLOYEE_REPORT(''UTL_FILE'', ''||  
  '''sal_rptXX''' || to_char(sysdate, ''HH24-MI-S'') || '.txt'');' ||  
'END;';
```

Este código se proporciona para ayudarle ya que se trata de una cadena PL/SQL no trivial para construir. En el bloque PL/SQL, XX es el número de estudiante.

- b. Pruebe el procedimiento SCHEDULE_REPORT ejecutándolo con un parámetro que especifique una frecuencia de 2 minutos y una hora de terminación de 10 minutos desde el inicio.

Nota: Deberá conectarse al servidor de base de datos utilizando PSFTP para comprobar si los archivos se han creado.

- c. Durante el proceso y después del mismo, puede consultar job_name y las columnas activadas de la tabla USER_SCHEDULER_JOBS para comprobar si el trabajo aún existe.

Nota: Esta consulta no debería devolver ninguna fila cuando hayan pasado 10 minutos.

Oracle Internal & OAI Use Only



SQL Dinámico y Metadatos

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- **Describir el flujo de ejecución de sentencias SQL**
- **Construir y ejecutar sentencias SQL dinámicamente utilizando SQL dinámico nativo (es decir, con sentencias EXECUTE IMMEDIATE)**
- **Comparar SQL dinámico nativo con el enfoque del paquete DBMS_SQL**
- **Utilizar el paquete DBMS_METADATA para obtener metadatos del diccionario de datos como XML o DDL de creación que pueden utilizarse para recrear los objetos**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetivos

En esta lección, aprenderá a construir y ejecutar sentencias SQL de forma dinámica, es decir, en tiempo de ejecución utilizando sentencias SQL dinámicas nativas en PL/SQL. Comparará el SQL dinámico nativo con el paquete DBMS_SQL, que proporciona capacidades similares.

Aprenderá a utilizar el paquete DBMS_METADATA para recuperar metadatos del diccionario de la base de datos como XML o DDL de creación y a ejecutar XML para recrear el objeto.

Flujo de Ejecución de SQL

- **Todas las sentencias SQL pasan por varias fases:**
 - Analizar
 - Enlazar
 - Ejecutar
 - Recuperar
- **Puede que algunas de estas etapas no sean relevantes para todas las sentencias, por ejemplo, la fase de recuperación se aplica a las consultas.**

Nota: Para sentencias SQL embebidas (`SELECT`, `DML`, `COMMIT` y `ROLLBACK`), las fases de análisis y enlace se realizan en tiempo de compilación. Para sentencias SQL dinámicas, todas las fases se realizan en tiempo de ejecución.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Pasos para Procesar Sentencias SQL

Todas las sentencias SQL tienen que pasar por varias fases: Sin embargo, puede que algunas fases no sean relevantes para todas las sentencias. Las siguientes fases son clave:

- **Analizar:** Toda sentencia SQL debe analizarse. Analizar la sentencia implica comprobar la sintaxis de la sentencia y validarla, garantizando que todas las referencias a objetos sean correctas y que los privilegios de esos objetos existen.
- **Enlazar:** Después de analizar, puede que el servidor de Oracle necesite valores de o para la variable ligada en la sentencia. El proceso para obtener estos valores se denomina enlace de variables. Esta fase se puede saltar si la sentencia no contiene variables ligadas.
- **Ejecutar:** En este punto, el servidor de Oracle server tiene toda la información y recursos necesarios y la sentencia se ejecuta. Para sentencias que no sean de consulta, ésta es la última fase.
- **Recuperar:** En la fase de recuperación, que se aplica a las consultas, las filas se seleccionan y ordenan (si lo solicita la consulta) y cada recuperación sucesiva recupera otra fila del resultado hasta que la última fila se haya recuperado.

SQL Dinámico

Utilice SQL dinámico para crear una sentencia SQL cuya estructura puede cambiar en tiempo de ejecución. SQL Dinámico:

- **Se construye y almacena como una cadena de caracteres dentro de la aplicación**
- **Es una sentencia SQL con datos de columna variables o diferentes condiciones con o sin variables pendientes de asignación (variables ligadas)**
- **Activa sentencias de definición y control de datos o sentencias de control de sesión para que se escriban y ejecuten desde PL/SQL**
- **Se ejecutan con sentencias SQL dinámicas nativas o con el paquete DBMS_SQL**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

SQL Dinámico

Las sentencias SQL embebidas disponibles en PL/SQL se limitan a SELECT, INSERT, UPDATE, DELETE, COMMIT y ROLLBACK, todas las cuales se analizan en tiempo de compilación, es decir, tienen una estructura fija. Necesitará utilizar la funcionalidad de SQL dinámico si necesita:

- Que la estructura de una sentencia SQL se modifique en tiempo de ejecución
- Acceso a sentencias DDL y otras funcionalidades de SQL en PL/SQL

Para realizar este tipo de tareas en PL/SQL, debe construir sentencias SQL de forma dinámica en cadenas de caracteres y ejecutarlas utilizando cualquier de los siguientes métodos:

- Sentencias SQL dinámicas nativas con EXECUTE IMMEDIATE
- Paquete DBMS_SQL

El proceso de utilización de sentencias SQL que no están embebidas en el programa de origen, que estén construidas en cadenas y que se ejecuten en tiempo de ejecución se conoce como “SQL dinámico”. Las sentencias SQL se crean de forma dinámica en tiempo de ejecución y pueden acceder y utilizar variables PL/SQL. Por ejemplo, cree un procedimiento que utiliza SQL dinámico para operar en una tabla cuyo nombre no se conoce hasta el momento de ejecución o para ejecutar una sentencia de lenguaje de definición de datos (DDL) (como CREATE TABLE), una sentencia de control de datos (como GRANT) o una sentencia de control de sesión (como ALTER SESSION).

SQL Dinámico Nativo

- Proporciona soporte nativo para SQL dinámica directamente en el lenguaje PL/SQL
- Proporciona la capacidad de ejecutar sentencias SQL cuya estructura no se conoce hasta el tiempo de ejecución
- Lo soportan las siguientes sentencias PL/SQL:
 - EXECUTE IMMEDIATE
 - OPEN-FOR
 - FETCH
 - CLOSE

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

SQL Dinámico Nativo

En Oracle 8 y versiones anteriores, la única forma de implementar SQL en una aplicación PL/SQL era utilizando el paquete DBMS_SQL. En Oracle 8i y versiones posteriores, el entorno PL/SQL proporciona SQL dinámico nativo como alternativa.

El SQL dinámico nativo proporciona la capacidad de ejecutar dinámicamente sentencias SQL cuya estructura se construye en tiempo de ejecución. Las siguientes sentencias se han agregado o ampliado en PL/SQL para soportar SQL dinámico nativo:

- **EXECUTE IMMEDIATE:** Prepara una sentencia, la ejecuta, devuelve variables y anula la asignación de los recursos
- **OPEN-FOR:** Prepara y ejecuta una sentencia utilizando una variable de cursor
- **FETCH:** Recupera los resultados de una sentencia abierta utilizando la variable de cursor
- **CLOSE:** Cierra el cursor utilizado por la variable de cursor y anula la asignación de los recursos

Puede utilizar variables ligadas en los parámetros dinámicos de las sentencias EXECUTE IMMEDIATE y OPEN. El SQL dinámico nativo incluye las siguientes capacidades:

- Define una sentencia de SQL dinámica
- Enlaza instancias de cualquier tipo de dato SQL soportado en PL/SQL
- Maneja las variables ligadas IN, IN OUT y OUT que están enlazadas por posición, no por nombre

Uso de la Sentencia EXECUTE IMMEDIATE

Utilice la sentencia EXECUTE IMMEDIATE para SQL dinámico nativo o para bloques PL/SQL anónimos:

```
EXECUTE IMMEDIATE dynamic_string
  [INTO {define_variable
    [, define_variable] ... | record}]
  [USING [IN|OUT|IN OUT] bind_argument
    [, [IN|OUT|IN OUT] bind_argument] ... ];
```

- **INTO se utiliza para consultas de una sola fila y especifica las variables o los registros en los que los valores de columna se recuperan.**
- **USING se utiliza para contener todos los argumentos enlazados. El modo de parámetro por defecto es IN, si no se especifica.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de la Sentencia EXECUTE IMMEDIATE

La sentencia EXECUTE IMMEDIATE se puede utilizar para ejecutar sentencias SQL o bloques PL/SQL anónimos. Entre los elementos sintácticos se incluyen:

- *dynamic_string* es una expresión de cadena que representa una sentencia SQL dinámica (sin terminador) o un bloque PL/SQL (con terminador).
- *define_variable* es una variable PL/SQL que almacena el valor de la columna seleccionada.
- *record* es un registro definido por el usuario o %ROWTYPE que almacena la fila seleccionada.
- *bind_argument* es una expresión cuyo valor se transfiere a la sentencia SQL dinámica o al bloque PL/SQL.
- La cláusula INTO especifica las variables o el registro en el que los valores de columna se recuperan. Sólo se utiliza para consultas de una sola fila. Por cada valor recuperado por la consulta, debe de haber una variable o campo correspondiente, compatible en la cláusula INTO.
- La cláusula USING contiene todos los argumentos enlazados. El modo de parámetro por defecto es IN.

Puede utilizar literales numéricos, de cadena y de caracteres como argumentos enlazados, pero no puede utilizar literales booleanos (TRUE, FALSE y NULL).

Nota: Utilice OPEN-FOR, FETCH y CLOSE para una consulta de varias filas. La sintaxis mostrada en la transparencia no está completa ya que existe soporte para operaciones de procesamiento en bloque, que es un tema que no abarca este curso.

SQL Dinámico con una Sentencia DDL

- Creación de una tabla:

```
CREATE PROCEDURE create_table(
    table_name VARCHAR2, col_specs VARCHAR2) IS
BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE '||table_name||
        ' (' || col_specs || ')';
END;
/
```

- Ejemplo de llamada:

```
BEGIN
    create_table('EMPLOYEE_NAMES',
        'id NUMBER(4) PRIMARY KEY, name VARCHAR2(40)');
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

SQL Dinámico con una Sentencia DDL

Los ejemplos de código muestran la creación de un procedimiento `create_table` que acepta el nombre de tabla y las definiciones de columna (especificaciones) como parámetros.

La llamada muestra la creación de una tabla denominada `EMPLOYEE_NAMES` con dos columnas:

- Una columna de identificador con un tipo de dato `NUMBER` utilizado como clave primaria
- Una columna de nombre de hasta 40 caracteres para el nombre del empleado

Cualquier sentencia DDL se puede ejecutar utilizando la sintaxis que aparece en la transparencia, tanto si la sentencia se ha construido de forma dinámica como si se ha especificado como cadena de literal. Puede crear y ejecutar una sentencia almacenada en una variable de cadena PL/SQL como en el siguiente ejemplo:

```
CREATE PROCEDURE add_col(table_name VARCHAR2,
    col_spec VARCHAR2) IS
    stmt VARCHAR2(100) := 'ALTER TABLE ' || table_name ||
        ' ADD '||col_spec;
BEGIN
    EXECUTE IMMEDIATE stmt;
END;
/
```

Para agregar una nueva columna a una tabla, introduzca:

```
EXECUTE add_col('employee_names', 'salary number(8,2)')
```

SQL Dinámico con Sentencias DML

- Supresión de filas de cualquier tabla:

```
CREATE FUNCTION del_rows(table_name VARCHAR2)
RETURN NUMBER IS
BEGIN
  EXECUTE IMMEDIATE 'DELETE FROM '||table_name;
  RETURN SQL%ROWCOUNT;
END;
```

```
BEGIN DBMS_OUTPUT.PUT_LINE(
  del_rows('EMPLOYEE_NAMES') || ' rows deleted.');
END;
```

- Inserción de una fila en una tabla con dos columnas:

```
CREATE PROCEDURE add_row(table_name VARCHAR2,
  id NUMBER, name VARCHAR2) IS
BEGIN
  EXECUTE IMMEDIATE 'INSERT INTO '||table_name||
    ' VALUES (:1, :2)' USING id, name;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

SQL Dinámico con Sentencias DML

Los ejemplos de la transparencia demuestran lo siguiente:

- La función `del_rows` suprime filas de la tabla especificada y devuelve el número de filas suprimidas utilizando el atributo del cursor SQL implícito `%ROWCOUNT`. La ejecución de la función se muestra debajo del ejemplo para crear una función.
- El procedimiento `add_row` muestra cómo proporcionar valores de entrada para una sentencia SQL dinámica con la cláusula `USING`. Los nombres de las variables ligadas `:1` y `:2` no son importantes, pero el orden de los nombres de las variables (`id` y `name`) en la cláusula `USING` está asociado a las variables ligadas por posición, según el orden de su respectivo aspecto. Por lo tanto, la variable PL/SQL `id` se asigna a la variable pendiente de asignación `:1` y la variable `name` se asigna a la variable pendiente de asignación `:2`. Los nombres de las variables pendientes de asignación/ligadas pueden ser alfanuméricos pero deben estar precedidos por dos puntos.

Nota: La sentencia `EXECUTE IMMEDIATE` prepara (analiza) y ejecuta inmediatamente la sentencia SQL dinámica. Las sentencias SQL dinámicas siempre se analizan.

Además, tenga en cuenta que no se realiza ninguna operación `COMMIT` en ninguno de los ejemplos. Por lo tanto, las operaciones se pueden deshacer con una sentencia `ROLLBACK`.

SQL Dinámico con una Consulta de una Sola Fila

Ejemplo de consulta de una sola fila:

```
CREATE FUNCTION get_emp(emp_id NUMBER)
RETURN employees%ROWTYPE IS
    stmt VARCHAR2(200);
    emprec employees%ROWTYPE;
BEGIN
    stmt := 'SELECT * FROM employees ' ||
            'WHERE employee_id = :id';
    EXECUTE IMMEDIATE stmt INTO emprec USING emp_id;
    RETURN emprec;
END;
/
```

```
DECLARE
    emprec employees%ROWTYPE := get_emp(100);
BEGIN
    DBMS_OUTPUT.PUT_LINE('Emp: ' || emprec.last_name);
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

SQL Dinámico con una Consulta de una Sola Fila

El ejemplo de consulta de una sola fila muestra la función `get_emp` que recupera un registro EMPLOYEES en una variable especificada en la cláusula `INTO`. También muestra cómo proporcionar valores de entrada para la cláusula `WHERE`.

El bloque anónimo se utiliza para ejecutar la función `get_emp` y devolver el resultado en una variable EMPLOYEES de registro local.

El ejemplo podría mejorarse para proporcionar cláusulas `WHERE` alternativas dependiendo de los valores de parámetros de entrada, lo que hace que sea más adecuado para el procesamiento de SQL dinámico.

SQL Dinámico con una Consulta de Varias Filas

Utilizar los procesamientos OPEN-FOR, FETCH y CLOSE:

```
CREATE PROCEDURE list_employees(deptid NUMBER) IS
  TYPE emp_refcsr IS REF CURSOR;
  emp_cv emp_refcsr;
  emprec employees%ROWTYPE;
  stmt varchar2(200) := 'SELECT * FROM employees';
BEGIN
  IF deptid IS NULL THEN OPEN emp_cv FOR stmt;
  ELSE
    stmt := stmt || ' WHERE department_id = :id';
    OPEN emp_cv FOR stmt USING deptid;
  END IF;
  LOOP
    FETCH emp_cv INTO emprec;
    EXIT WHEN emp_cv%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(emprec.department_id|||
                           ' ' || emprec.last_name);
  END LOOP;
  CLOSE emp_cv;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

SQL Dinámico con una Consulta de Varias Filas

El ejemplo de la transparencia muestra cómo ejecutar una consulta de varias filas realizando los siguientes pasos de programación:

- Declarar un tipo REF CURSOR
- Declarar una variable de cursor basada en el tipo de nombre REF CURSOR que declare
- Ejecutar una sentencia OPEN-FOR que utiliza una variable de cursor
- Utilizar una sentencia FETCH que haga referencia a la variable de cursor hasta que todos los registros se hayan procesado
- Ejecutar la sentencia CLOSE utilizando la variable de cursor

Este proceso es igual que utilizar definiciones de cursor estático. Sin embargo, la sintaxis OPEN-FOR acepta un literal de cadena o variable que especifique la sentencia SELECT, que se puede construir de forma dinámica.

Nota: En la siguiente página se proporciona una breve introducción al tipo REF CURSOR y a las variables de cursor. Una alternativa es utilizar la sintaxis BULK COLLECT soportada por las sentencias de SQL dinámicas nativas (tema que no se trata en este curso).

Declaración de Variables de Cursor

- Declarar un tipo de cursor como REF CURSOR:

```
CREATE PROCEDURE process_data IS
    TYPE ref_ctype IS REF CURSOR; -- weak ref cursor
    TYPE emp_ref_ctype IS REF CURSOR -- strong
        RETURN employees%ROWTYPE;
    :
```

- Declarar una variable de cursor utilizando el tipo de cursor:

```
:
dept_csrvar ref_ctype;
emp_csrvar emp_ref_ctype;
BEGIN
    OPEN emp_csrvar FOR SELECT * FROM employees;
    OPEN dept_csrvar FOR SELECT * from departments;
    -- Then use as normal cursors
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Declaración de Variables de Cursor

Una variable de cursor es un identificador PL/SQL cuyo nombre de tipo se ha declarado como tipo REF CURSOR. Para crear una variable de cursor hay que seguir dos pasos:

- Declarar un nombre de tipo como REF CURSOR
- Declarar una variable PL/SQL utilizando el nombre de tipo declarado como tipo REF CURSOR

Los ejemplos de la transparencia crean dos tipos de cursores de referencia:

- El cursor ref_ctype es un cursor de referencia genérico, conocido como cursor de referencia débil. Un cursor de referencia débil puede estar asociado a cualquier consulta.
- El cursor emp_ref_ctype es un tipo de cursor de referencia fuerte que se debe asociar a una consulta de tipo compatible: la consulta debe devolver datos que sean compatibles con el tipo especificado después de la palabra clave RETURN (por ejemplo, un tipo de fila EMPLOYEES).

Una vez declarada una variable de cursor utilizando como referencia el nombre del tipo de cursor de referencia, la variable del cursor asociada a una consulta se abre utilizando la sintaxis OPEN-FOR mostrada. Las operaciones estándar FETCH, atributos de cursor y CLOSE utilizadas con cursores explícitos también se pueden aplicar con las variables de cursor. Para comparar las variables de cursor con cursores explícitos:

- Una variable de cursor se puede asociar a más de una consulta en tiempo de ejecución
- Un cursor explícito se asocia a una consulta en tiempo de compilación

Ejecución Dinámica de un Bloque PL/SQL

Ejecución dinámica de un bloque PL/SQL anónimo:

```
CREATE FUNCTION annual_sal(emp_id NUMBER)
RETURN NUMBER IS
  plsql varchar2(200) :=
    'DECLARE'|||
    '  emprec employees%ROWTYPE; '||
    'BEGIN'||
    '  emprec := get_emp(:empid); '||
    '  :res := emprec.salary * 12; '||
    'END;';
  result NUMBER;
BEGIN
  EXECUTE IMMEDIATE plsql
    USING IN emp_id, OUT result;
  RETURN result;
END;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(annual_sal(100))
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ejecución Dinámica de un Bloque PL/SQL

La función `annual_sal` construye de forma dinámica un bloque PL/SQL anónimo. El bloque PL/SQL contiene variables ligadas para:

- La entrada del identificador de empleado utilizando la variable pendiente de asignación `:empid`
- El resultado de salida que calcula el salario anual de los empleados utilizando la variable pendiente de asignación `:res`

Nota: Este ejemplo demuestra cómo utilizar la sintaxis de resultado OUT (en la cláusula USING de la sentencia EXECUTE IMMEDIATE) para obtener el resultado calculado por el bloque PL/SQL. Las variables de salida del procedimiento y los valores de retorno de la función se pueden obtener de forma similar de un bloque PL/SQL anónimo ejecutado de forma dinámica.

Uso de SQL Dinámico Nativo para Compilar Código PL/SQL

Compilar código PL/SQL con la sentencia ALTER:

- **ALTER PROCEDURE name COMPILE**
- **ALTER FUNCTION name COMPILE**
- **ALTER PACKAGE name COMPILE SPECIFICATION**
- **ALTER PACKAGE name COMPILE BODY**

```
CREATE PROCEDURE compile_plsql(name VARCHAR2,
    plsql_type VARCHAR2, options VARCHAR2 := NULL) IS
    stmt varchar2(200) := 'ALTER ' || plsql_type || 
        ' ' || name || ' COMPILE';
BEGIN
    IF options IS NOT NULL THEN
        stmt := stmt || ' ' || options;
    END IF;
    EXECUTE IMMEDIATE stmt;
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de SQL Dinámico Nativo para Compilar Código PL/SQL

El procedimiento `compile_plsql` del ejemplo se puede utilizar para compilar diferentes códigos PL/SQL utilizando la sentencia DDL ALTER. Se muestran cuatro formas básicas de la sentencia ALTER para compilar:

- Un procedimiento
- Una función
- Una especificación del paquete
- Un cuerpo del paquete

Nota: Si omite las palabras clave SPECIFICATION o BODY con la sentencia ALTER PACKAGE, tanto la especificación como el cuerpo se compilan.

A continuación se presentan ejemplos de llamadas al procedimiento en la transparencia para cada uno de los cuatro casos, respectivamente:

```
EXEC compile_plsql ('list_employees', 'procedure')
EXEC compile_plsql ('get_emp', 'function')
EXEC compile_plsql ('mypack', 'package', 'specification')
EXEC compile_plsql ('mypack', 'package', 'body')
```

Compilación con DEBUG activado para la función `get_emp`:

```
EXEC compile_plsql ('get_emp', 'function', 'debug')
```

Uso del Paquete DBMS_SQL

El paquete DBMS_SQL se utiliza para escribir SQL dinámico en procedimientos almacenados y para analizar sentencias DDL. Algunos de los procedimientos y funciones del paquete son:

- **OPEN_CURSOR**
- **PARSE**
- **BIND_VARIABLE**
- **EXECUTE**
- **FETCH_ROWS**
- **CLOSE_CURSOR**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso del Paquete DBMS_SQL

Si utiliza DBMS_SQL, puede escribir procedimientos almacenados y bloques PL/SQL anónimos que utilizan SQL dinámico, como ejecutar sentencias DDL en PL/SQL, por ejemplo, ejecutar una sentencia `DROP TABLE`. Las operaciones que proporciona este paquete las realiza el usuario actual, no el propietario del paquete `SYS`. El paquete DBMS_SQL proporciona los siguientes subprogramas para ejecutar SQL dinámico:

- OPEN_CURSOR para abrir un nuevo cursor y devolver un número de identificador de cursor.
- PARSE para analizar la sentencia SQL, es decir, comprueba la sintaxis de la sentencia y la asocia al cursor abierto. Las sentencias DDL se ejecutan inmediatamente al ser analizadas.
- BIND_VARIABLE para enlazar un valor determinado a una variable ligada identificada por su nombre en la sentencia que se está analizando. No es necesario si la sentencia no tiene variables ligadas.
- EXECUTE para ejecutar la sentencia SQL y devolver el número de filas procesadas.
- FETCH_ROWS para recuperar la siguiente fila para una consulta (se utiliza en un bucle para varias filas).
- CLOSE_CURSOR para cerrar el cursor especificado.

Nota: Si utiliza el paquete DBMS_SQL para ejecutar sentencias DDL se puede producir un interbloqueo. Por ejemplo, la razón más probable es que el paquete se está utilizando para borrar un procedimiento que aún está utilizando.

Uso de DBMS_SQL con una Sentencia DML

Ejemplo de supresión de filas:

```
CREATE OR REPLACE FUNCTION delete_all_rows
  (table_name VARCHAR2) RETURN NUMBER IS
  csr_id INTEGER;
  rows_del    NUMBER;
BEGIN
  csr_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQLPARSE(csr_id,
    'DELETE FROM '||table_name, DBMS_SQL.NATIVE);
  rows_del := DBMS_SQL.EXECUTE (csr_id);
  DBMS_SQL CLOSE_CURSOR (csr_id);
  RETURN rows_del;
END;
/
BEGIN DBMS_OUTPUT.PUT_LINE('Rows Deleted: ' || 
  delete_all_rows('employees'));
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de DBMS_SQL con una Sentencia DML

En la transparencia, el nombre de la tabla se transfiere a la función `delete_all_rows`. La función utiliza SQL dinámico para suprimir filas de la tabla especificada y devuelve un recuento que representa el número de filas que se han suprimido después la correcta ejecución de la sentencia.

Para procesar una sentencia DML de forma dinámica, realice los siguientes pasos:

1. Utilice `OPEN_CURSOR` para establecer un área en la memoria para procesar una sentencia SQL.
2. Utilice `PARSE` para establecer la validez de la sentencia SQL.
3. Utilice la función `EXECUTE` para ejecutar la sentencia SQL. Esta función devuelve el número de filas procesadas.
4. Utilice `CLOSE_CURSOR` para cerrar el cursor.

Los pasos para ejecutar una sentencia DDL son similares; pero el paso 3 es opcional porque la sentencia DDL se ejecuta de forma inmediata cuando `PARSE` se termina correctamente, es decir, la sintáctica y la semántica de la sentencia son correctas. Si utiliza la función `EXECUTE` con una sentencia DDL, no hace nada y devuelve un valor de 0 para el número de filas procesadas, ya que las sentencias DDL no procesan filas.

Uso de DBMS_SQL con una Sentencia DML con Parámetros

```
CREATE PROCEDURE insert_row (table_name VARCHAR2,
                            id VARCHAR2, name VARCHAR2, region NUMBER) IS
    csr_id      INTEGER;
    stmt        VARCHAR2(200);
    rows_added NUMBER;
BEGIN
    stmt := 'INSERT INTO '||table_name||
            ' VALUES (:cid, :cname, :rid)';
    csr_id := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQLPARSE(csr_id, stmt, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(csr_id, ':cid', id);
    DBMS_SQL.BIND_VARIABLE(csr_id, ':cname', name);
    DBMS_SQL.BIND_VARIABLE(csr_id, ':rid', region);
    rows_added := DBMS_SQL.EXECUTE(csr_id);
    DBMS_SQL CLOSE_CURSOR(csr_id);
    DBMS_OUTPUT.PUT_LINE(rows_added||' row added');
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de DBMS_SQL con una Sentencia DML con Parámetros

El ejemplo de la transparencia realiza la operación DML de insertar una fila en la tabla especificada. El ejemplo demuestra el paso adicional necesario para asociar valores a variables ligadas que existen en la sentencia SQL. Por ejemplo, una llamada al procedimiento anterior es:

```
EXECUTE insert_row('countries', 'ZA', 'South Africa', 4)
```

Cuando la sentencia se analiza, debe llamar al procedimiento DBMS_SQL.BIND_VARIABLE para asignar valores para cada variable ligada que exista en la sentencia. El enlace de valores se debe realizar antes de ejecutar el código. Para procesar una sentencia SELECT dinámicamente, realice los siguientes pasos después de abrir y antes de cerrar el cursor:

1. Ejecute DBMS_SQL.DEFINE_COLUMN para cada columna seleccionada.
2. Ejecute DBMS_SQL.BIND_VARIABLE para cada variable ligada en la consulta.
3. Para cada fila, haga lo siguiente:
 - a. Ejecute DBMS_SQL.FETCH_ROWS para recuperar una fila y devolver el número de filas recuperadas. Pare el procesamiento adicional cuando se devuelve un valor de cero.
 - b. Ejecute DBMS_SQL.COLUMN_VALUE para recuperar el valor de cada columna seleccionada en la variable PL/SQL para el procesamiento.

Aunque este proceso de codificación no es complejo, lleva más tiempo escribirlo y es proclive a errores comparado con el enfoque de SQL dinámico nativo.

Comparación de SQL Dinámico Nativo y el Paquete DBMS_SQL

SQL Dinámico Nativo:

- Es más fácil de utilizar que DBMS_SQL
- Necesita menos código que DBMS_SQL
- Mejora el rendimiento porque el intérprete de PL/SQL le proporciona soporte nativo
- Soporta todos los tipos soportados por SQL estático en PL/SQL, incluidos tipos definidos por el usuario
- Puede recuperar filas directamente en registros PL/SQL

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Comparación de SQL Dinámico Nativo y el Paquete DBMS_SQL

El SQL dinámico nativo proporciona las siguientes ventajas sobre el paquete DBMS_SQL.

Facilidad de uso: Ya que el SQL dinámico nativo se integra con SQL, puede utilizarlo del mismo modo que utiliza actualmente el SQL estático en código PL/SQL. El código es normalmente más compacto y legible comparado con el código escrito con el paquete DBMS_SQL.

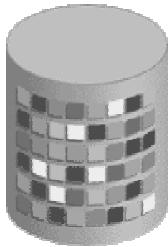
Mejora de rendimiento: El SQL dinámico nativo tiene un rendimiento significativamente mejor que el DBMS_SQL, en la mayoría de las circunstancias, debido al soporte nativo proporcionado por el intérprete PL/SQL. El enfoque DBMS_SQL utiliza API de procedimiento y se ve afectado por una alta llamada de procedimiento y una sobrecarga de copia de datos.

Soporte para tipos definidos por el usuario: El SQL dinámico nativo soporta todos los tipos soportados por el SQL estático en PL/SQL. Por lo tanto, el SQL dinámico nativo proporciona soporte para los tipos definidos por el usuario tales como objetos, recopilaciones y REF definidos por el usuario. El paquete DBMS_SQL no soporta estos tipos definidos por el usuario. Sin embargo, tiene soporte limitado para matrices.

Soporte para la recuperación en los registros: Con SQL dinámico nativo, las filas que resultan de una consulta se pueden recuperar directamente en los registros PL/SQL. El paquete DBMS_SQL no soporta la recuperación en las estructuras de registros.

Paquete DBMS_METADATA

El paquete DBMS_METADATA proporciona una utilidad centralizada para la extracción, manipulación y reenvío de los metadatos del diccionario.



ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Paquete DBMS_METADATA

Puede llamar al paquete DBMS_METADATA para recuperar metadatos del diccionario de la base de datos como XML o DDL de creación y ejecutar XML para recrear el objeto.

Puede utilizar DBMS_METADATA para extraer metadatos del diccionario, manipularlos (agregar columnas, cambiar tipos de dato de columna, etc.) y, a continuación, convertir los metadatos en lenguaje de definición de datos (DDL) para que el objeto pueda recrearse en la misma u otra base de datos. Antes había que hacer esto mediante programación con problemas que aparecían en cada nueva versión.

La funcionalidad DBMS_METADATA se utiliza para la sustitución de exportación/importación de Oracle 10g , comúnmente denominada “Pump de Datos”.

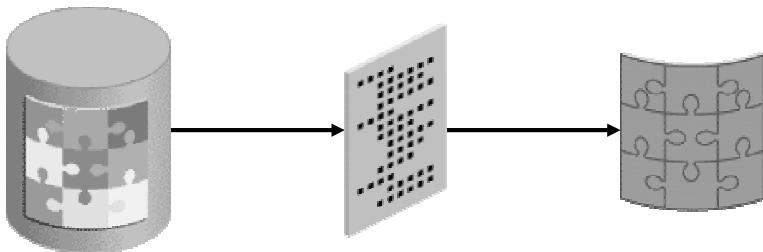
Este paquete se introdujo en Oracle9i y se ha mejorado en la base de datos Oracle 10g.

Nota: Para obtener más información sobre el paquete DBMS_DATAPUMP, consulte el curso de eStudy titulado *Oracle Database 10g: Reduce Management - Tools and Utilities*.

API de Metadatos

El procesamiento implica los siguientes pasos:

- 1. La recuperación de los metadatos de un objeto como XML.**
- 2. La transformación de XML de varias maneras (incluida su transformación en SQL DDL).**
- 3. La ejecución de XML para recrear el objeto.**



ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

API de Metadatos

Cada entidad de la base de datos adopta el modelo de un objeto que pertenece a un tipo de objeto. Por ejemplo, la tabla EMPLOYEES es un objeto; el tipo de objeto es TABLE. Cuando recupera los metadatos de un objeto, debe especificar el tipo de objeto.

Cada tipo de objeto se implementa con tres entidades:

- Un tipo definido por el usuario (UDT) cuyos atributos constan de todos los metadatos para los objetos de ese tipo. La representación XML de un objeto es una traducción de una instancia de tipo en XML, con los nombres de las etiquetas derivados de los nombres de atributos del tipo. (En el caso de las tablas, se necesitan varios UDTs para representar las diferentes variedades del tipo de objeto.)
- Una vista de objeto del UDT que rellena las instancias del tipo de objeto.
- Un archivo de comandos XSL (Lenguaje de Hoja de Estilo Extensible) que convierte la representación XML de un objeto en SQL DDL.

Subprogramas en DBMS_METADATA

Nombre	Descripción
OPEN	Especifica el tipo de objeto que se debe recuperar, la versión de sus metadatos y el modelo de objeto. El valor de retorno es un manejador de contexto opaco para el juego de objetos.
SET_FILTER	Especifica las restricciones en los objetos que se deben recuperar tales como el nombre o el esquema del objeto
SET_COUNT	Especifica el número máximo de objetos que se deben recuperar en una sola llamada <code>FETCH_xxx</code>
GET_QUERY	Devuelve el texto de las consultas que utilizará <code>FETCH_xxx</code>
SET_PARSE_ITEM	Activa el análisis de la salida y especifica el atributo de un objeto para que se analice y devuelva
ADD_TRANSFORM	Especifica una transformación que <code>FETCH_xxx</code> aplica a la representación XML de los objetos recuperados
SET_TRANSFORM_PARAM, SET_REMAP_PARAM	Especifica los parámetros de la hoja de estilo XSLT identificada por <code>transform_handle</code>
FETCH_xxx	Devuelve metadatos para los objetos que cumplen los criterios establecidos por OPEN, SET_FILTER
CLOSE	Invalide el manejador devuelto por OPEN y limpia el estado asociado

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Subprogramas en DBMS_METADATA

La tabla proporciona una visión general de los procedimientos y las funciones disponibles en el paquete DBMS_METADATA. Para recuperar metadatos, puede especificar:

- El tipo de objeto recuperado, ya sea un tipo de objeto (tabla, índice, procedimiento) o una recopilación heterogénea de tipos de objeto que forman una unidad lógica (como la exportación de una base de datos y la exportación de esquemas)
- Criterios de selección (propietario, nombre, etc.)
- Atributos “parse items” de objetos que deben ser analizados y devueltos por separado
- Transformaciones en la salida, implementadas por los archivos de comandos XSLT

El paquete proporciona dos tipos de interfaces de recuperación para dos tipos de uso:

- Para la utilización programática: OPEN, SET_FILTER, SET_COUNT, GET_QUERY, SET_PARSE_ITEM, ADD_TRANSFORM, SET_TRANSFORM_PARAM, SET_REMAP_PARAM (nuevo en la base de datos Oracle 10g), FETCH_xxx y CLOSE. Éstos permiten criterios de selección flexible y extracción de un flujo de objetos.
- Para la utilización en consultas SQL y para la exploración ad-hoc: Las interfaces GET_xxx (GET_XML y GET_DDL) devuelven metadatos para un solo objeto con nombre. Las interfaces GET_DEPENDENT_xxx y GET_GRANTED_xxx devuelven metadatos para uno o más objetos dependientes u otorgados. Ninguno de estos API soportan tipos de objetos heterogéneos.

Subprogramas `FETCH_xxx`

Nombre	Descripción
<code>FETCH_XML</code>	Esta función devuelve los metadatos XML para un objeto como <code>XMLType</code> .
<code>FETCH_DDL</code>	Esta función devuelve DDL (ya sea para crear o para borrar el objeto) en una tabla anidada predefinida.
<code>FETCH_CLOB</code>	Esta función devuelve los objetos, transformados o no, como <code>CLOB</code> .
<code>FETCH_XML_CLOB</code>	Este procedimiento devuelve metadatos XML para estos objetos como <code>CLOB</code> en un parámetro <code>IN OUT NOCOPY</code> para evitar costosas copias LOB.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Subprogramas `FETCH_xxx`

Estas funciones y procedimientos devuelven metadatos para los objetos que cumplen los criterios establecidos por la llamada a la función `OPEN` que ha devuelto el manejador y las llamadas posteriores a `SET_FILTER`, `SET_COUNT`, `ADD_TRANSFORM`, etc. Cada llamada a `FETCH_xxx` devuelve el número de objetos especificado por `SET_COUNT` (o un número inferior, si menos objetos permanecen en el cursor actual) hasta que todos los objetos se hayan devuelto.

Procedimiento SET_FILTER

- **Sintaxis:**

```
PROCEDURE set_filter
( handle IN NUMBER,
  name   IN VARCHAR2,
  value  IN VARCHAR2 | BOOLEAN | NUMBER,
  object_type_path VARCHAR2
);
```

- **Ejemplo:**

```
...
DBMS_METADATA.SET_FILTER (handle, 'NAME', 'HR');
...
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Procedimiento SET_FILTER

El procedimiento SET_FILTER se utiliza para identificar las restricciones de los objetos que se deben recuperar. Por ejemplo, puede especificar las restricciones en un objeto o esquema que se está recuperando. Este procedimiento está sobrecargado con los parámetros que tienen los siguientes significados:

- handle es el manejador devuelto desde la función OPEN.
- name es el nombre del filtro. Para cada filtro, el tipo de objeto se aplica a su nombre, tipo de dato (texto o booleano) y significado o efecto (incluido su valor por defecto, si hay uno).
- value es el valor del filtro. Puede ser un valor de texto, booleano o numérico.
- object_type_path es un nombre de ruta de acceso que designa los tipos de objeto a los que se aplica el filtro. Por defecto, el filtro se aplica al tipo de objeto del manejador OPEN.

Si utiliza un filtro de expresión, éste se colocará a la derecha de una comparación SQL y del valor con el que se compara. El valor debe contener paréntesis y comillas donde corresponda. Un valor de filtro se combina con un atributo de objeto particular para producir una condición WHERE en la consulta que recupera los objetos.

Filtros

Hay más de 70 filtros organizados por categorías de tipos de objetos tales como:

- **Objetos con nombre**
- **Tablas**
- **Objetos dependientes de tablas**
- **Índice**
- **Objetos dependientes**
- **Objetos otorgados**
- **Datos de tabla**
- **Estadísticas de índice**
- **Restricciones**
- **Todos los tipos de objetos**
- **Exportación de la base de datos**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Filtros

Hay más de 70 filtros que puede especificar cuando utiliza el procedimiento SET_FILTER. Estos filtros están organizados en categorías de tipos de objetos. Algunas de las nuevas categorías de tipos de objetos en la base de datos Oracle 10g se muestran en la transparencia.

Al utilizar el procedimiento SET_FILTER, especifica el nombre del filtro y su valor correspondiente.

Por ejemplo, puede utilizar el filtro SCHEMA con un valor para identificar el esquema cuyos objetos se han seleccionado. A continuación, utilice una segunda llamada al procedimiento SET_FILTER y utilice un filtro denominado INCLUDE_USER que contenga un tipo de dato booleano para su valor. Si está definido en TRUE, se recuperan los objetos que contienen información privilegiada sobre el usuario.

```
DBMS_METADATA.SET_FILTER(handle, SCHEMA, 'HR') ;
DBMS_METADATA.SET_FILTER(handle, INCLUDE_USER, TRUE) ;
```

Cada llamada a SET_FILTER hace que una condición WHERE se agregue a la consulta subyacente que recupera el juego de objetos. Las condiciones WHERE se combinan utilizando un operador AND, para que pueda utilizar varias llamadas SET_FILTER para refinar el juego de objetos que se debe devolver.

Ejemplos de Definición de Filtros

Para configurar el filtro para recuperar los objetos del esquema HR excluidos los tipos de objetos de las funciones, procedimientos y paquetes además de cualquier vista que contenga PAYROLL al inicio del nombre de la vista:

```
DBMS_METADATA.SET_FILTER(handle, 'SCHEMA_EXPR',
  'IN (''PAYROLL'', ''HR''));
DBMS_METADATA.SET_FILTER(handle, 'EXCLUDE_PATH_EXPR',
  '=''''FUNCTION'''');
DBMS_METADATA.SET_FILTER(handle, 'EXCLUDE_PATH_EXPR',
  '=''''PROCEDURE'''');
DBMS_METADATA.SET_FILTER(handle, 'EXCLUDE_PATH_EXPR',
  '=''''PACKAGE'''');
DBMS_METADATA.SET_FILTER(handle, 'EXCLUDE_NAME_EXPR',
  'LIKE ''PAYROLL%'''', 'VIEW');
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ejemplos de Definición de Filtros

El ejemplo que se muestra en la transparencia llama al procedimiento SET_FILTER varias veces para crear una condición WHERE que identifique qué tipos de objetos se deben recuperar. Primero, los objetos del esquema HR y PAYROLL se identifican como tipos de objetos que se deben recuperar. Después, el procedimiento SET_FILTER identifica ciertos tipos de objetos (funciones, procedimientos y paquetes) y nombres de objetos de vista que se deben excluir.

Uso Programático: Ejemplo 1

```
CREATE PROCEDURE example_one IS
    h      NUMBER; th1  NUMBER; th2  NUMBER;
    doc    sys.ku$_ddls; ←①
BEGIN
    h := DBMS_METADATA.OPEN('SCHEMA_EXPORT'); ←②
    DBMS_METADATA.SET_FILTER(h, 'SCHEMA', 'HR'); ←③
    th1 := DBMS_METADATA.ADD_TRANSFORM(h, ←④
        'MODIFY', NULL, 'TABLE');
    DBMS_METADATA.SET_REMAP_PARAM(th1, ←⑤
        'REMAP_TABLESPACE', 'SYSTEM', 'TBS1');
    th2 := DBMS_METADATA.ADD_TRANSFORM(h, 'DDL'); ←⑥
    DBMS_METADATA.SET_TRANSFORM_PARAM(th2,
        'SQLTERMINATOR', TRUE);
    DBMS_METADATA.SET_TRANSFORM_PARAM(th2,
        'REF_CONSTRAINTS', FALSE, 'TABLE');
    LOOP
        doc := DBMS_METADATA.FETCH_DDL(h); ←⑦
        EXIT WHEN doc IS NULL;
    END LOOP;
    DBMS_METADATA CLOSE(h); ←⑧
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso Programático: Ejemplo 1

En este ejemplo, todos los objetos se recuperan desde el esquema HR como DDL de creación.

La transformación MODIFY se utiliza para cambiar los tablespaces para las tablas.

1. El paquete DBMS_METADATA tiene varios tipos predefinidos pertenecientes a SYS. El tipo sys.ku\$_ddls se define en el paquete DBMS_METADATA. Es un tipo de tabla que contiene tipos CLOB de datos.
2. Puede utilizar la función OPEN para especificar el tipo de objeto que se debe recuperar, la versión de sus metadatos y el modelo de objeto. Devuelve un manejador de contexto para el juego de objetos.
En este ejemplo, 'SCHEMA_EXPORT' es el tipo de objeto e indica todos los objetos de metadatos en un esquema. Hay 85 tipos predefinidos de objetos para el modelo que puede especificar para este parámetro. Tanto los parámetros de la versión de metadatos como del modelo del objeto no se identifican en este ejemplo. La versión del parámetro de metadatos está definida por defecto en 'COMPATIBLE'. También puede especificar 'LATEST' o una versión de base de datos concreta. Actualmente, el parámetro model sólo soporta el modelo Oracle en la base de datos Oracle 10g. Éste es el valor por defecto.
3. El procedimiento SET_FILTER identifica las restricciones en los objetos que se deben recuperar.

Uso Programático: Ejemplo 1 (continuación)

4. La función ADD_TRANSFORM especifica una transformación que FETCH_xxx aplica a la representación XML de los objetos recuperados. Puede obtener más de una transformación. En el ejemplo, suceden dos transformaciones, una por cada una de las variables de programa th1 y th2. La función ADD_TRANSFORM acepta cuatro parámetros y devuelve un número que representa el manejador opaco a la transformación. Los parámetros son el manejador devuelto desde la sentencia OPEN, el nombre de la transformación (DDL, DROP o MODIFY), el nombre de la codificación que es el nombre del juego de caracteres NLS en el que la hoja de estilo apuntada por el nombre está codificada y el tipo de objeto. Si el tipo de objeto está omitido, la transformación se aplica a todos los objetos; de otro modo se aplica sólo al tipo de objeto especificado.
La primera transformación mostrada en el código del programa es el manejador devuelto desde la función OPEN. La segunda transformación que se muestra en el código tiene dos valores de parámetros especificados. El primer parámetro es el manejador identificado desde la función OPEN. El segundo valor de parámetro es DDL, lo que significa que el documento se transforma en DDL que crea el objeto. La salida de esta transformación no es un documento XML. Los parámetros tercero y cuarto no se especifican. Ambos toman los valores por defecto para los parámetros de la codificación y el tipo de objeto.
5. El procedimiento SET_REMAP_PARAM identifica los parámetros con la hoja de estilo XSLT identificada por el manejador de la transformación, que es el primer parámetro transferido al procedimiento. En el ejemplo, el valor del segundo parámetro 'REMAP_TABLESPACE' significa que a los tablespaces de los objetos se les cambia el nombre de un valor anterior a un valor nuevo. En la función ADD_TRANSFORM, las opciones son DDL, DROP o MODIFY. Para cada uno de estos valores, SET_REMAP_PARAM identifica el nombre del parámetro. REMAP_TABLESPACE significa que a los objetos del documento se les cambiará el nombre de los tablespaces de un valor anterior a un valor nuevo. Los parámetros tercero y cuarto identifican el valor anterior y el valor nuevo. En este ejemplo, el nombre anterior del tablespace es SYSTEM y el nombre nuevo del tablespace es TBS1.
6. SET_TRANSFORM_PARAM funciona de forma similar para SET_REMAP_PARAM. En el código mostrado, la primera llamada a SET_TRANSFORM_PARAM identifica los parámetros para la variable th2. Los valores de los parámetros SQLTERMINATOR y TRUE hacen que el terminador SQL(; o /) se agregue a cada sentencia DDL.
La segunda llamada a SET_TRANSFORM_PARAM identifica más características para la variable th2. REF_CONSTRAINTS, FALSE, TABLE significa que las restricciones referenciales de las tablas no se copian en el documento.
7. La función FETCH_DDL devuelve metadatos para los objetos que cumplen los criterios establecidos por las subrutinas OPEN, SET_FILTER, ADD_TRANSFORM, SET_REMAP_PARAM y SET_TRANSFORM_PARAM.
8. La función CLOSE function invalida el manejador devuelto por la función OPEN y limpia el estado asociado. Utilice esta función para terminar el flujo de objetos establecido por la función OPEN.

Uso Programático: Ejemplo 2

```
CREATE FUNCTION get_table_md RETURN CLOB IS
  h    NUMBER; -- returned by 'OPEN'
  th   NUMBER; -- returned by 'ADD_TRANSFORM'
  doc  CLOB;
BEGIN
  -- specify the OBJECT TYPE
  h := DBMS_METADATA.OPEN('TABLE');
  -- use FILTERS to specify the objects desired
  DBMS_METADATA.SET_FILTER(h, 'SCHEMA', 'HR');
  DBMS_METADATA.SET_FILTER(h, 'NAME', 'EMPLOYEES');
  -- request to be TRANSFORMED into creation DDL
  th := DBMS_METADATA.ADD_TRANSFORM(h, 'DDL');
  -- FETCH the object
  doc := DBMS_METADATA.FETCH_CLOB(h);
  -- release resources
  DBMS_METADATA CLOSE(h);
  RETURN doc;
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso Programático: Ejemplo 2

Este ejemplo devuelve los metadatos para la tabla EMPLOYEES. El resultado es:

```
set pagesize 0
set long 1000000
SELECT get_table_md FROM dual;

CREATE TABLE "HR"."EMPLOYEES"
  (  "EMPLOYEE_ID" NUMBER(6,0),
     "FIRST_NAME" VARCHAR2(20),
     "LAST_NAME" VARCHAR2(25) CONSTRAINT
"EMP_LAST_NAME_NN"
        NOT NULL ENABLE,
     "e-mail" VARCHAR2(25) CONSTRAINT "EMP_e-mail_NN"
        NOT NULL ENABLE,
     "PHONE_NUMBER" VARCHAR2(20),
     "HIRE_DATE" DATE CONSTRAINT "EMP_HIRE_DATE_NN"
        NOT NULL ENABLE,
     "JOB_ID" VARCHAR2(10) CONSTRAINT "EMP_JOB_NN"
        NOT NULL ENABLE,
     "SALARY" NUMBER(8,2),
  ...

```

Uso Programático: Ejemplo 2 (continuación)

```
"COMMISSION_PCT" NUMBER(2,2),
    "MANAGER_ID" NUMBER(6,0),
    "DEPARTMENT_ID" NUMBER(4,0),
    CONSTRAINT "EMP_SALARY_MIN" CHECK (salary > 0)
ENABLE,
    CONSTRAINT "EMP_e-mail_UK" UNIQUE ("e-mail")
USING INDEX PCTFREE 10 INITTRANS 2 MAXTRANS 255
STORAGE(INITIAL 65536 NEXT 65536 MINEXTENTS 1
MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT)
TABLESPACE "EXAMPLE" ENABLE,
    CONSTRAINT "EMP_EMP_ID_PK" PRIMARY KEY
("EMPLOYEE_ID")
USING INDEX PCTFREE 10 INITTRANS 2 MAXTRANS 255
STORAGE(INITIAL 65536 NEXT 65536 MINEXTENTS 1
MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT)
TABLESPACE "EXAMPLE" ENABLE,
    CONSTRAINT "EMP_DEPT_FK" FOREIGN KEY
("DEPARTMENT_ID")
    REFERENCES "HR"."DEPARTMENTS" ("DEPARTMENT_ID")
ENABLE,
    CONSTRAINT "EMP_JOB_FK" FOREIGN KEY ("JOB_ID")
    REFERENCES "HR"."JOBS" ("JOB_ID") ENABLE,
    CONSTRAINT "EMP_MANAGER_FK" FOREIGN KEY
("MANAGER_ID")
    REFERENCES "HR"."EMPLOYEES" ("EMPLOYEE_ID")
ENABLE
) PCTFREE 0 PCTUSED 40 INITTRANS 1 MAXTRANS 255 C
OMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 65536 MINEXTENTS 1
MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT)
TABLESPACE "EXAMPLE"
Puede conseguir el mismo efecto con la interfaz de
exploracion:
SELECT dbms_metadata.get_ddl
('TABLE','EMPLOYEES','HR')
FROM dual;
```

API de Exploración

Nombre	Descripción
GET_XXX	Las funciones GET_XML y GET_DDL devuelven metadatos para un sólo objeto con nombre.
GET_DEPENDENT_XXX	Esta función devuelve metadatos para un objeto dependiente.
GET_GRANTED_XXX	Esta función devuelve metadatos para un objeto otorgado.
Donde xxx es:	DDL O XML

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

API de Exploración

Las API de exploración están diseñadas para su uso en las consultas SQL y en las exploraciones ad-hoc. Estas funciones permiten recuperar metadatos para objetos con una sola llamada. Encapsulan llamadas a OPEN, SET_FILTER, etcétera. Qué función que utiliza depende de las características del tipo de objeto y de si desea XML o DDL.

Para algunos tipos de objetos, se puede utilizar más de una función. Puede utilizar GET_XXX para recuperar un nombre de índice o GET_DEPENDENT_XXX para recuperar el mismo índice especificando la tabla en la que está definido.

GET_XXX devuelve un sólo nombre de objeto.

Para GET_DEPENDENT_XXX y GET_GRANTED_XXX, puede que un número arbitrario de objetos otorgados o dependientes coincidan con los criterios de entrada. Puede especificar un recuento de objetos al recuperarlos.

Si llama a estas funciones desde iSQL*Plus, deberá utilizar los comandos SET LONG y SET PAGESIZE para recuperar la salida completa e ininterrumpida.

```
SET LONG 2000000  
SET PAGESIZE 300
```

API de Exploración: Ejemplos

1. Obtenga representación XML de HR.EMPLOYEES:

```
SELECT DBMS_METADATA.GET_XML
      ('TABLE', 'EMPLOYEES', 'HR')
FROM   dual;
```

2. Recupere el DDL para todos los objetos otorgados en HR.EMPLOYEES:

```
SELECT DBMS_METADATA.GET_DEPENDENT_DDL
      ('OBJECT_GRANT', 'EMPLOYEES', 'HR')
FROM   dual;
```

3. Recupere el DDL para todos los permisos del sistema otorgados a HR:

```
SELECT DBMS_METADATA.GET_GRANTED_DDL
      ('SYSTEM_GRANT', 'HR')
FROM   dual;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

API de Exploración: Ejemplos

- Los resultados de la recuperación de representación XML de HR.EMPLOYEES son:

```
DBMS_METADATA.GET_XML('TABLE','EMPLOYEES','HR')
-----
<?xml version="1.0"?>
<ROWSET>
  <ROW>
    <TABLE_T>
      <VERS_MAJOR>1</VERS_MAJOR>
```

- Los resultados de la recuperación de DDL para todos los objetos otorgados en HR.EMPLOYEES son:

```
DBMS_METADATA.GET_DEPENDENT_DDL
  ('OBJECT_GRANT', 'EMPLOYEES', 'HR')
-----
GRANT SELECT ON "HR"."EMPLOYEES" TO "OE"
GRANT REFERENCES ON "HR"."EMPLOY"
```

- Los resultados de la recuperación de DDL para todos los permisos del sistema otorgados a HR son:

```
DBMS_METADATA.GET_GRANTED_DDL('SYSTEM_GRANT','HR')
-----
GRANT UNLIMITED TABLESPACE TO "HR"
```

API de Exploración: Ejemplos

```
BEGIN
    DBMS_METADATA.SET_TRANSFORM_PARAM(
        DBMS_METADATA.SESSION_TRANSFORM,
        'STORAGE', false);
END;
/
SELECT DBMS_METADATA.GET_DDL('TABLE', u.table_name)
FROM user_all_tables u
WHERE u.nested = 'NO'
AND (u.iot_type IS NULL OR u.iot_type = 'IOT');

BEGIN
    DBMS_METADATA.SET_TRANSFORM_PARAM(
        DBMS_METADATA.SESSION_TRANSFORM, 'DEFAULT');
END;
/
```

- 1
- 2
- 3

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

API de Exploración: Ejemplos (continuación)

El ejemplo de la transparencia muestra cómo recuperar DDL de creación para todas las tablas “completas” en el esquema actual, filtrando las tablas anidadas y los segmentos de desbordamiento. Los pasos que aparecen en la transparencia son los siguientes:

1. La función SET_TRANSFORM_PARAM especifica que las cláusulas de almacenamiento no se deben devolver en SQL DDL. La función SESSION_TRANSFORM se interpreta para que signifique “para la sesión actual”.
2. Utilice la función GET_DDL para recuperar DDL en todas las tablas no anidadas y no IOT.

```
CREATE TABLE "HR"."COUNTRIES"
( "COUNTRY_ID" CHAR(2)
    CONSTRAINT "COUNTRY_ID_NN" NOT NULL ENABLE,
    "COUNTRY_NAME" VARCHAR2(40),
    "REGION_ID" NUMBER,
    CONSTRAINT "COUNTRY_C_ID_PK"
    PRIMARY KEY ("COUNTRY_ID") ENABLE,
    CONSTRAINT "COUNTR_REG_FK" FOREIGN KEY
    ...
```

3. Restablezca los parámetros a nivel de sesión a los valores establecidos por defecto.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- **Explicar el flujo de ejecución de sentencias SQL**
- **Crear sentencias SQL dinámicamente y ejecutarlas utilizando sentencias SQL dinámicas nativas o el paquete DBMS_SQL**
- **Reconocer las ventajas de uso de SQL dinámico nativo en comparación con el paquete DBMS_SQL**
- **Utilizar subprogramas DBMS_METADATA para obtener metadatos del diccionario de datos mediante programación**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen

En esta lección, ha aprendido cómo crear dinámicamente cualquier sentencia SQL y ejecutarla utilizando sentencias SQL dinámicas nativas. La ejecución dinámica de código SQL y PL/SQL amplía las capacidades de PL/SQL más allá de operaciones de consulta y transacción. En versiones anteriores de la base de datos, se podían obtener resultados parecidos con el paquete DBMS_SQL.

En esta lección se han explorado algunas diferencias y se ha realizado una comparación entre la utilización de SQL dinámico nativo y del paquete DBMS_SQL. Si está utilizando Oracle8i o versiones posteriores, debería utilizar SQL dinámico nativo para los nuevos proyectos.

En la lección también se describió la utilización del paquete DBMS_METADATA para recuperar metadatos del diccionario de la base de datos con los resultados que se han presentado en XML o formato DDL de creación. Los datos XML resultantes se pueden utilizar para recrear el objeto.

Práctica 6: Visión General

En esta práctica se abordan los siguientes temas:

- **Creación de un paquete que utiliza SQL dinámico nativo para crear o borrar una tabla y para llenar, modificar y suprimir filas de una tabla**
- **Creación de un paquete que compile el código PL/SQL en el esquema**
- **Uso de DBMS_METADATA para mostrar la sentencia para regenerar un subprograma PL/SQL**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica 6: Visión General

En esta práctica, escribirá código para realizar las siguientes tareas:

- Creación de un paquete que utiliza SQL dinámico nativo para crear o borrar una tabla y para llenar, modificar y suprimir filas de una tabla.
- Creación de un paquete que compile código PL/SQL en el esquema, ya sea todo el código PL/SQL o aquel que tenga un estado INVALID en la tabla USER_OBJECTS.
- Uso de DBMS_METADATA para regenerar código PL/SQL para cualquier procedimiento que tenga en el esquema.

Práctica 6

1. Cree un paquete denominado TABLE_PKG que utilice SQL dinámico nativo para crear o borrar una tabla y para llenar, modificar y suprimir filas de la tabla.

- a. Cree una especificación del paquete siguiendo estos procedimientos:

```
PROCEDURE make(table_name VARCHAR2, col_specs VARCHAR2)
PROCEDURE add_row(table_name VARCHAR2, values VARCHAR2,
                  cols VARCHAR2 := NULL)
PROCEDURE upd_row(table_name VARCHAR2, values VARCHAR2,
                  conditions VARCHAR2 := NULL)
PROCEDURE del_row(table_name VARCHAR2,
                  conditions VARCHAR2 := NULL);
PROCEDURE remove(table_name VARCHAR2)
```

Asegúrese de que los subprogramas gestionan parámetros por defecto opcionales con valores NULL.

- b. Cree el cuerpo del paquete que acepte los parámetros y construya dinámicamente las sentencias SQL adecuadas que se ejecutan utilizando SQL dinámico nativo, excepto por el procedimiento remove que se debería escribir utilizando el paquete DBMS_SQL.

- c. Ejecute el procedimiento MAKE del paquete para crear la siguiente tabla:

```
make('my_contacts', 'id number(4), name varchar2(40)');
```

- d. Describa la estructura de la tabla MY_CONTACTS.

- e. Ejecute el procedimiento empaquetado ADD_ROW para agregar las siguientes filas:

```
add_row('my_contacts', '1, ''Geoff Gallus'', 'id, name');
add_row('my_contacts', '2, ''Nancy'', 'id, name');
add_row('my_contacts', '3, ''Sunitha Patel'', 'id, name');
add_row('my_contacts', '4, ''Valli Pataballa'', 'id, name');
```

- f. Consulte el contenido de la tabla MY_CONTACTS.

- g. Ejecute el procedimiento empaquetado DEL_ROW para suprimir un contacto con ID valor 1.

- h. Ejecute el procedimiento UPD_ROW con los siguientes datos de fila:

```
upd_row('my_contacts', 'name=' 'Nancy Greenberg', 'id=2');
```

- i. Seleccione los datos de la tabla MY_CONTACTS de nuevo para ver los cambios.

- j. Borre la tabla utilizando el procedimiento remove y describa la tabla MY_CONTACTS.

2. Cree un paquete COMPILE_PKG que compile el código PL/SQL en el esquema.

- a. En la especificación, cree un procedimiento empaquetado denominado MAKE que acepte el nombre de una unidad de programa PL/SQL para compilarla.

- b. En el cuerpo, el procedimiento MAKE debería llamar a una función privada denominada GET_TYPE para determinar el tipo de objeto PL/SQL del diccionario de datos y devolver el nombre del tipo (utilice PACKAGE para un paquete con cuerpo) si existe el objeto; de lo contrario, devolverá NULL. Si el objeto existe, MAKE lo compila dinámicamente con la sentencia ALTER.

- c. Utilice el procedimiento COMPILE_PKG.MAKE para compilar el procedimiento EMPLOYEE_REPORT el paquete EMP_PKG y un objeto no existente denominado EMP_DATA.

Práctica 6 (continuación)

3. Agregue un procedimiento a COMPILE_PKG que utilice DBMS_METADATA para obtener una sentencia DDL que pueda regenerar un subprograma PL/SQL con nombre y escriba la sentencia DDL en un archivo utilizando el paquete UTL_FILE.
 - a. En la especificación del paquete, cree un procedimiento denominado REGENERATE que acepte el nombre de un componente PL/SQL para regenerarse. Declare una variable pública VARCHAR2 denominada `dir` inicializada con valor de alias de directorio '`UTL_FILE`'. Compile la especificación.
 - b. En el cuerpo del paquete, implemente el procedimiento REGENERATE para que utilice la función `GET_TYPE` para determinar el tipo de objeto PL/SQL del nombre proporcionado. Si el objeto existe, obtenga el DDL utilizado para crear el componente con el procedimiento `DBMS_METADATA.GET_DDL`, que se debe proporcionar con el nombre del objeto en mayúscula. Guarde la sentencia DDL en un archivo utilizando el procedimiento `UTL_FILE.PUT`. Escriba el archivo en la ruta de acceso del directorio almacenada en la variable pública denominada `dir` (de la especificación). Construya un nombre de archivo (en minúsculas) concatenando la función `USER`, un subrayado y el nombre de objeto con una extensión `.sql`. Por ejemplo: `ora1_myobject.sql`. Compile el cuerpo.
 - c. Ejecute el procedimiento `COMPILE_PKG.REGENERATE` utilizando el nombre de `TABLE_PKG` creado en la primera tarea de esta práctica.
 - d. Utilice Putty FTP para obtener el archivo generado del servidor en el directorio local. Edite el archivo para insertar un carácter de terminación / al final de una sentencia `CREATE` (si es necesario). Corte y pegue los resultados en el buffer de *iSQL*Plus* y ejecute la sentencia.

Oracle Internal & OAI Use Only

Oracle Internal & OAI Use Only

Consideraciones de Diseño para Código PL/SQL

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- Utilizar especificaciones de paquete para crear excepciones y constantes estándar
- Escribir y llamar a subprogramas locales
- Definir la directiva AUTHID para controlar los privilegios en tiempo de ejecución de un subprograma
- Ejecutar subprogramas para realizar transacciones autónomas
- Utilizar el enlace en bloque y la cláusula RETURNING con DML
- Transferir parámetros por referencia mediante una indicación NOCOPY
- Utilizar la indicación PARALLEL ENABLE para la optimización

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetivos

En esta lección aprenderá a utilizar las especificaciones de paquete para estandarizar los nombres para excepciones y valores de constantes. Aprenderá a crear subprogramas en la sección Declaration de cualquier bloque PL/SQL para su uso de forma local en el bloque. Se describe la directiva de compilador AUTHID para mostrar cómo puede gestionar privilegios en tiempo de ejecución del código PL/SQL y crear transacciones independientes mediante la directiva AUTONOMOUS TRANSACTION para subprogramas.

En esta lección también se abordan algunas consideraciones sobre rendimiento que se puede aplicar a las aplicaciones PL/SQL como, por ejemplo, operaciones de enlace en bloque con una única sentencia SQL, la cláusula RETURNING y las indicaciones NOCOPY y PARALLEL ENABLE.

Estandarización de Constantes y Excepciones

Las constantes y las excepciones se suelen implementar mediante un paquete sin cuerpo (es decir, en una especificación del paquete).

- **La estandarización ayuda a:**
 - Desarrollar programas que sean consistentes
 - Fomentar un grado superior de reutilización de código
 - Facilitar el mantenimiento del código
 - Implementar los estándares de la compañía en aplicaciones completas
- **Empezar con la estandarización de:**
 - Nombres de excepciones
 - Definiciones de constantes

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Estandarización de Constantes y Excepciones

Cuando varios desarrolladores escriben sus propios manejadores de excepciones en una aplicación, pueden haber inconsistencias en el manejo de situaciones de error. A menos que se adhieran a determinados estándares, la situación puede resultar confusa debido a los diferentes enfoques seguidos en el manejo del mismo error o debido a la visualización de mensajes de error en conflicto que confunden a los usuarios. Para evitar esto, puede:

- Implementar estándares de la compañía que utilicen un enfoque consistente con respecto al manejo de errores en la aplicación completa.
- Crear manejadores de excepciones genéricos predefinidos que produzcan consistencia en la aplicación.
- Escribir y llamar a programas que produzcan mensajes de error consistentes.

Todos los buenos entornos de programación fomentan los estándares de nomenclatura y de codificación. En PL/SQL, una buena forma para empezar a implementar estándares de nomenclatura y codificación es con las constantes y las excepciones más utilizadas que se producen en el dominio de aplicación.

La construcción de la especificación del paquete PL/SQL es un excelente componente para dar soporte a la estandarización, porque todos los identificadores declarados en la especificación del paquete son públicos. Son visibles para los subprogramas que desarrolla el propietario del paquete y todo el código con derechos EXECUTE para la especificación del paquete.

Estandarización de Excepciones

Crear un paquete de manejo de errores estandarizado que incluya todas las excepciones con nombre y definidas por el programador que se utilizarán en la aplicación.

```
CREATE OR REPLACE PACKAGE error_pkg IS
    fk_err      EXCEPTION;
    seq_nbr_err EXCEPTION;
    PRAGMA EXCEPTION_INIT (fk_err, -2292);
    PRAGMA EXCEPTION_INIT (seq_nbr_err, -2277);
    ...
END error_pkg;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Estandarización de Excepciones

En el ejemplo de la transparencia, el paquete `error_pkg` es un paquete de excepción estandarizado. Declara un juego de identificadores de excepción definidos por el programador. Puesto que muchas de las excepciones predefinidas de la base de datos Oracle no tienen nombres identificadores, el paquete de ejemplo mostrado utiliza la directiva `PRAGMA EXCEPTION_INIT` para asociar los nombres de excepciones seleccionados a un número de error de la base de datos Oracle. De esta forma, puede hacer referencia a las excepciones de una forma estándar en las aplicaciones. Por ejemplo:

```
BEGIN
    DELETE FROM departments
    WHERE department_id = deptno;
    ...
EXCEPTION
    WHEN error_pkg.fk_err THEN
    ...
    WHEN OTHERS THEN
    ...
END;
/
```

Estandarización del Manejo de Excepciones

Puede escribir un subprograma para el manejo de excepciones comunes para:

- Mostrar errores basados en los valores SQLCODE y SQLERRM para las excepciones
- Realizar un seguimiento de errores de tiempo de ejecución fácilmente mediante parámetros en el código para identificar:
 - El procedimiento en el que se produjo el error
 - La ubicación (número de línea) del error
 - RAISE_APPLICATION_ERROR con capacidades de rastreo de pila y con el tercer argumento definido en TRUE

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Estandarización del Manejo de Excepciones

El manejo de excepciones estandarizado se puede implementar como un subprograma autónomo o como un subprograma agregado al paquete que define las excepciones estándar. Considere la creación de un paquete con lo siguiente:

- Cada una de las excepciones con nombre que se utilizará en la aplicación.
- Todas las excepciones definidas por el programador sin nombre que se utilizan en la aplicación. Se trata de los números de error del –20000 al –20999.
- Un programa para llamar a RAISE_APPLICATION_ERROR basado en excepciones del paquete.
- Un programa para mostrar un error basado en los valores de SQLCODE y SQLERRM.
- Objetos adicionales, como tablas de log de errores, y programas para acceder a las tablas.

Una práctica común consiste en utilizar parámetros que identifiquen el nombre del procedimiento y la ubicación en la que se ha producido el error. De esta forma, puede realizar un seguimiento de los errores de tiempo de ejecución más fácilmente. Una alternativa es utilizar el procedimiento incorporado RAISE_APPLICATION_ERROR para realizar un rastreo de pila de las excepciones que se pueden utilizar para un seguimiento de la secuencia de llamada causante del error. Para ello, defina el tercer argumento opcional en TRUE. Por ejemplo:

```
RAISE_APPLICATION_ERROR(-20001, 'My first error', TRUE);
```

Esto resulta importante cuando se produce más de una excepción de esta forma.

Estandarización de Constantes

Para los programas que utilizan variables locales cuyos valores no deben cambiar:

- **Convertir las variables en constantes para reducir el mantenimiento y la depuración**
- **Crear una especificación del paquete central y colocar todas las constantes en ella**

```
CREATE OR REPLACE PACKAGE constant_pkg IS
    c_order_received CONSTANT VARCHAR(2) := 'OR';
    c_order_shipped   CONSTANT VARCHAR(2) := 'OS';
    c_min_sal         CONSTANT NUMBER(3)  := 900;
    ...
END constant_pkg;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Estandarización de Constantes

Por definición, el valor de una variable cambia, mientras que el de una constante no puede hacerlo. Si tiene programas que utilizan variables locales cuyos valores no deben cambiar y no lo hacen, convierta dichas variables en constantes. Esto puede ayudar en el mantenimiento y la depuración del código.

Puede crear un único paquete compartido con todas las constantes. Esto facilita enormemente el mantenimiento y el cambio de las constantes. Este procedimiento o paquete se puede cargar en el inicio del sistema para un mejor rendimiento.

En el ejemplo de la transparencia se muestra el paquete `constant_pkg` que contiene algunas constantes. Haga referencia a cualquiera de las constantes del paquete en la aplicación cuando sea necesario. Por ejemplo:

```
BEGIN
    UPDATE employees
        SET salary = salary + 200
        WHERE salary <= constant_pkg.c_min_sal;
    ...
END;
/
```

Subprogramas Locales

- **Un subprograma local es un argumento PROCEDURE o FUNCTION definido en la sección de declaraciones.**

```
CREATE PROCEDURE employee_sal(id NUMBER) IS
    emp employees%ROWTYPE;
    FUNCTION tax(salary VARCHAR2) RETURN NUMBER IS
        BEGIN
            RETURN salary * 0.825;
        END tax;
    BEGIN
        SELECT * INTO emp
        FROM EMPLOYEES WHERE employee_id = id;
        DBMS_OUTPUT.PUT_LINE('Tax: ' || tax(emp.salary));
    END;
```

- **El subprograma local se debe definir al final de la sección de declaraciones.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Subprogramas Locales

Los subprogramas locales pueden afectar al diseño de arriba abajo. Reducen el tamaño de un módulo al eliminar código redundante. Ésta es una de las principales razones de la creación de un subprograma local. Si un módulo necesita la misma rutina varias veces, pero sólo este módulo la necesita, defínalala como un subprograma local.

Puede definir un bloque PL/SQL con nombre en la sección de declaraciones del programa, procedimiento, función o bloque PL/SQL anónimo, siempre que se declare al final de la sección Declaration. Los subprogramas locales poseen estas características:

- Sólo puede acceder a ellos el bloque en el que están definidos.
- Se compilan como parte de sus bloques delimitadores.

Las ventajas de los subprogramas locales son las siguientes:

- Reducción del código repetitivo
- Legibilidad mejorada del código y facilidad de mantenimiento
- Menos administración porque existe un programa para el mantenimiento en lugar de dos

El concepto es simple. En la transparencia se ilustra esto con un ejemplo básico del cálculo del impuesto sobre la renta del salario de un empleado.

Derechos del Responsable de la Definición frente a Derechos del Invocador

Derechos del responsable de la definición:

- Utilizado en versiones anteriores a Oracle8i.
- Los programas se ejecutan con los privilegios del usuario creador.
- El usuario no necesita privilegios en los objetos subyacentes a los que accede el procedimiento.
El usuario necesita privilegios sólo para ejecutar un procedimiento.

Derechos del invocador:

- Introducido por primera vez en Oracle8i.
- Los programas se ejecutan con los privilegios del usuario que realiza la llamada.
- El usuario necesita privilegios en los objetos subyacentes a los que accede el procedimiento.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Derechos del Responsable de la Definición frente a Derechos del Invocador

Modelo de Derechos del Responsable de la Definición

En las versiones anteriores a Oracle8i, todos los programas se ejecutaban con los privilegios del usuario que creó el subprograma. Esto se denomina modelo de derechos del responsable de la definición y tiene las siguientes características:

- Concede al emisor de llamada del programa el privilegio para ejecutar el procedimiento, pero no privilegios en los objetos subyacentes a los que accede dicho procedimiento.
- Necesita que el propietario tenga todos los privilegios de objeto necesarios para los objetos a los que hace referencia el procedimiento.

Por ejemplo, si el usuario Scott crea un subprograma PL/SQL `get_employees` que posteriormente llama Sarah, el procedimiento `get_employees` se ejecuta con los privilegios del responsable de la definición Scott.

Modelo de Derechos del Invocador

En el modelo de derechos del invocador, introducido por primera vez en Oracle8i, los programas se ejecutan con los privilegios del usuario que realiza la llamada. Un usuario de un procedimiento ejecutado con los derechos del invocador necesita privilegios en los objetos subyacentes a los que el procedimiento hace referencia.

Por ejemplo, si Sarah llama al subprograma PL/SQL `get_employees` de Scott, el procedimiento `get_employees` se ejecuta con los privilegios del invocador Sarah.

Especificación de Derechos del Invocador

Definir AUTHID en CURRENT_USER:

```
CREATE OR REPLACE PROCEDURE add_dept(
    id NUMBER, name VARCHAR2) AUTHID CURRENT_USER IS
BEGIN
    INSERT INTO departments
    VALUES (id, name, NULL, NULL);
END;
```

Cuando se utiliza con funciones, procedimientos o paquetes autónomos:

- Los nombres utilizados en consultas, DML, SQL dinámico nativo y el paquete DBMS_SQL se resuelven en el esquema del invocador
- Las llamadas a otros paquetes, funciones y procedimientos se resuelven en el esquema del responsable de la definición

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Especificación de Derechos del Invocador

Definición de Derechos del Invocador

A continuación se muestra la sintaxis de definición de los derechos del invocador para distintas construcciones de subprogramas PL/SQL:

```
CREATE FUNCTION name RETURN type AUTHID CURRENT_USER IS...
CREATE PROCEDURE name AUTHID CURRENT_USER IS...
CREATE PACKAGE name AUTHID CURRENT_USER IS...
CREATE TYPE name AUTHID CURRENT_USER IS OBJECT...
```

En estas sentencias, defina AUTHID en DEFINER, o bien no lo utilice para obtener un comportamiento por defecto.

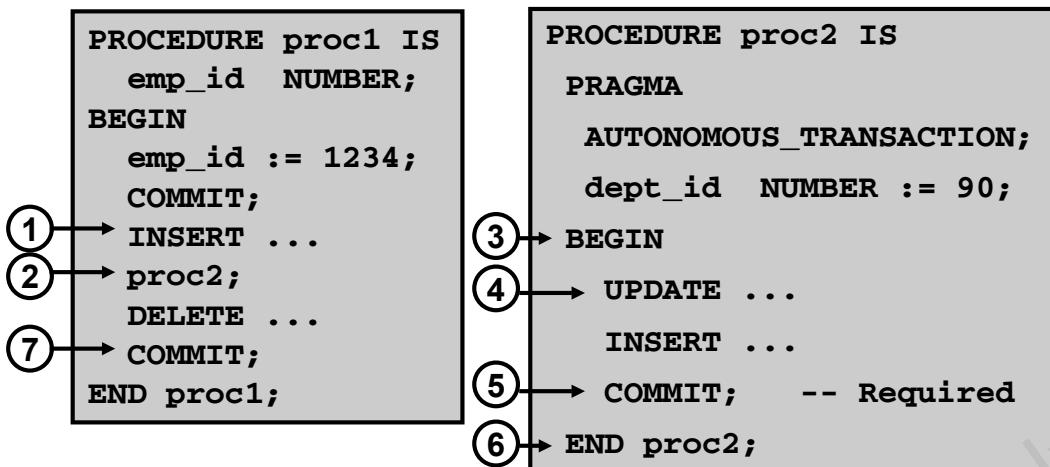
Resolución de Nombres

Para un procedimiento con derechos del responsable de la definición, todas las referencias externas se resuelven en el esquema del responsable de la definición. Para un procedimiento con derechos del invocador, la resolución de referencias externas depende del tipo de sentencia en la que aparezcan:

- Los nombres utilizados en consultas, sentencias DML, SQL dinámico y DBMS_SQL se resuelven en el esquema del invocador.
- Todas las demás sentencias, como las llamadas a otros paquetes, funciones y procedimientos, se resuelven en el esquema del responsable de la definición.

Transacciones Autónomas

- Son transacciones independientes iniciadas por otra transacción principal.
- Se especifican con PRAGMA AUTONOMOUS_TRANSACTION.



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Transacciones Autónomas

Una transacción es una serie de sentencias que constituye una unidad lógica de trabajo que termina o falla como una unidad integrada. A menudo, una transacción inicia otra que puede necesitar funcionar fuera del ámbito de la transacción que la inició. Es decir, en una transacción existente, puede que una transacción independiente necesaria deba confirmar o realizar un rollback de los cambios sin afectar al resultado de la transacción inicial. Por ejemplo, en una transacción de compra de stock, se debe confirmar la información del cliente independientemente de si se termina la compra global de stock. O bien, al ejecutar la misma transacción, desea registrar mensajes en una tabla incluso si se realiza un rollback de la transacción global.

A partir de Oracle8i, las transacciones autónomas se agregaron para posibilitar la creación de una transacción independiente. Una transacción autónoma (AT) es una transacción independiente iniciada por otra principal (MT). En la transparencia se muestra el comportamiento de una AT:

1. Empieza la transacción principal.
2. Se llama a un procedimiento proc2 para iniciar la transacción autónoma.
3. Se suspende la transacción principal.
4. Empieza la operación de la transacción autónoma.
5. La transacción autónoma termina en una operación de confirmación o de rollback.
6. Se reanuda la transacción principal.
7. Termina la transacción principal.

Funciones de las Transacciones Autónomas

Las transacciones autónomas:

- **Son independientes de la transacción principal**
- **Suspenden la transacción que realiza la llamada hasta que terminan**
- **No son transacciones anidadas**
- **No se realiza un rollback de ellas si se hace de la transacción principal**
- **Permiten que los cambios sean visibles para otras transacciones tras una confirmación**
- **Están delimitadas (iniciadas y finalizadas) por subprogramas individuales y no por un bloque PL/SQL anónimo o anidado**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Funciones de las Transacciones Autónomas

Las transacciones autónomas presentan las siguientes funciones:

- Aunque se llaman en una transacción, las transacciones autónomas son independientes de ella. Es decir, no son transacciones anidadas.
- Si se realiza un rollback de la transacción principal, no se hace de las transacciones autónomas.
- Los cambios realizados por una transacción autónoma son visibles para otras transacciones cuando se confirma dicha transacción autónoma.
- Con su función similar a una pila, sólo se puede acceder a la transacción “superior” en un momento determinado. Al finalizar, se sale de la transacción autónoma y se reanuda la transacción que realizó la llamada.
- No existen límites, a excepción de los límites de recursos, en el número de transacciones autónomas que se pueden llamar de forma recurrente.
- Es necesario confirmar explícitamente las transacciones autónomas o realizar explícitamente un rollback de las mismas; de lo contrario, se produce un error al intentar volver del bloque autónomo.
- No puede utilizar PRAGMA para marcar todos los subprogramas de un paquete como autónomos. Sólo las rutinas individuales se pueden marcar como autónomas.
- No puede marcar un bloque PL/SQL anónimo o anidado como autónomo.

Uso de Transacciones Autónomas

Ejemplo:

```
PROCEDURE bank_trans(cardnbr NUMBER, loc NUMBER) IS
BEGIN
    log_usage (cardnbr, loc);
    INSERT INTO txn VALUES (9001, 1000,...);
END bank_trans;
```

```
PROCEDURE log_usage (card_id NUMBER, loc NUMBER)
IS
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO usage
    VALUES (card_id, loc);
    COMMIT;
END log_usage;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de Transacciones Autónomas

Para definir transacciones autónomas, utilice PRAGMA AUTONOMOUS_TRANSACTION. PRAGMA indica al compilador PL/SQL que marque una rutina como autónoma (independiente). En este contexto, el término “rutina” incluye bloques PL/SQL anónimos de nivel superior (no anidados); funciones y procedimientos locales, autónomos y empaquetados; métodos de un tipo de objeto SQL y disparadores de base de datos. Puede codificar PRAGMA en cualquier lugar de la sección de declaraciones de una rutina. Sin embargo, por motivos de legibilidad, es mejor colocarla en la parte superior de la sección Declaration.

En el ejemplo de la transparencia, se realiza un seguimiento de dónde se utiliza la tarjeta de crédito, independientemente de si la transacción se realiza correctamente. A continuación se presentan las ventajas de las transacciones autónomas:

- Una vez iniciada, una transacción autónoma es totalmente independiente. No comparte bloqueos, recursos ni dependencias de confirmación con la transacción principal, por lo que puede registrar eventos, aumentar los contadores de reintentos, etc., incluso si se realiza un rollback de la transacción principal.
- Reviste mayor importancia que las transacciones autónomas ayudan a crear componentes de software modulares y reutilizables. Por ejemplo, los procedimientos almacenados puede iniciar y terminar transacciones autónomas por su cuenta. Una aplicación que realice una llamada debe conocer las operaciones autónomas de un procedimiento y el procedimiento tiene que conocer el contexto de la transacción de la aplicación. Esto convierte a las transacciones autónomas en menos proclives a errores que las transacciones normales y en más fáciles de utilizar.

Cláusula RETURNING

La cláusula RETURNING:

- Mejora el rendimiento al devolver valores de columna con sentencias INSERT, UPDATE y DELETE
- Elimina la necesidad de una sentencia SELECT

```
CREATE PROCEDURE update_salary(emp_id NUMBER) IS
    name      employees.last_name%TYPE;
    new_sal   employees.salary%TYPE;
BEGIN
    UPDATE employees
        SET salary = salary * 1.1
    WHERE employee_id = emp_id
    RETURNING last_name, salary INTO name, new_sal;
END update_salary;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

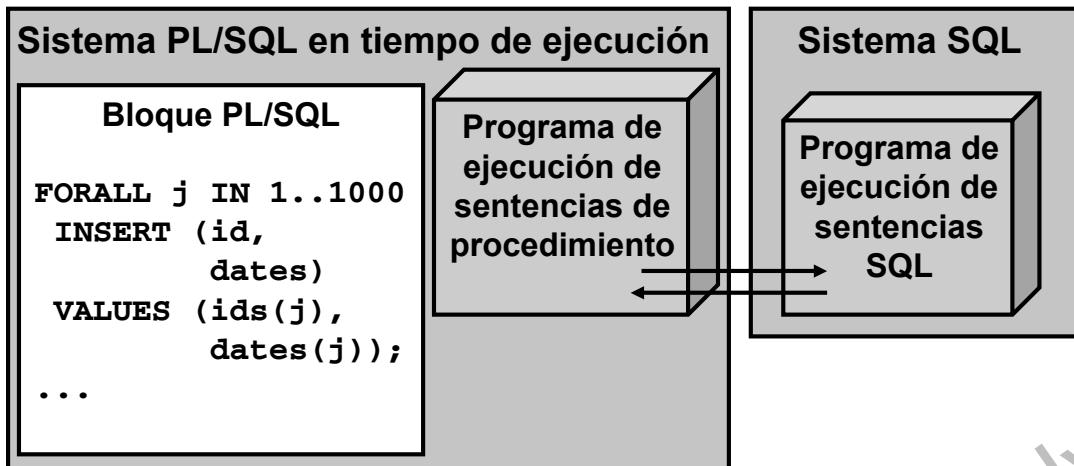
Cláusula RETURNING

Con frecuencia, las aplicaciones necesitan información sobre la fila afectada por una operación SQL (por ejemplo, para generar un informe o para realizar una acción posterior). Las sentencias INSERT, UPDATE y DELETE pueden incluir una cláusula RETURNING, que devuelve valores de columna de la fila afectada en variables PL/SQL o del host. Esto elimina la necesidad de utilizar SELECT para seleccionar la fila después de INSERT o UPDATE o antes de DELETE. En consecuencia, se necesitan menos recorridos de ida y vuelta en la red, menos tiempo de CPU del servidor, menos cursor y menos memoria del servidor.

En el ejemplo de la transparencia se muestra cómo actualizar el salario de un empleado y al mismo tiempo recuperar el apellido y el nuevo salario del empleado en una variable PL/SQL local.

Enlace en Bloque

Enlaza todas las matrices de valores en una única operación, en lugar de utilizar un bucle para realizar una operación **FETCH, **INSERT**, **UPDATE** y **DELETE** varias veces**



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Enlace en Bloque

El servidor de Oracle utiliza dos sistemas para ejecutar subprogramas y bloques PL/SQL:

- El sistema PL/SQL en tiempo de ejecución, que ejecuta sentencias de procedimiento pero transfiere las sentencias SQL al sistema SQL.
- El sistema SQL, que analiza y ejecuta la sentencia SQL y, en algunos casos, devuelve datos al sistema PL/SQL.

Durante la ejecución, cada una de las sentencias SQL provoca un intercambio de contexto entre los dos sistemas, lo que afecta al rendimiento para cantidades excesivas de procesamiento SQL. Esto es típico de aplicaciones que tienen una sentencia SQL en un bucle que utiliza valores de elementos de recopilación indexados. Las recopilaciones incluyen las tablas anidadas, las matrices variables, las tablas de índice y las matrices del host.

El rendimiento se puede mejorar de manera considerable al minimizar el número de intercambios de contexto mediante el enlace en bloque. El enlace en bloque hace que una recopilación completa se enlace en una llamada, un intercambio de contexto, al sistema SQL. Es decir, un proceso de enlace en bloque transfiere la recopilación completa de valores entre los dos sistemas en un solo intercambio de contexto, comparado con provocar un intercambio de contexto para cada elemento de recopilación en una iteración de un bucle. Cuantas más filas se vean afectadas por una sentencia SQL, mayor será la mejora del rendimiento con el enlace en bloque.

Uso del Enlace en Bloque

Palabras clave para soportar el enlace en bloque:

- La palabra clave **FORALL** indica al sistema PL/SQL que debe realizar un enlace en bloque de las recopilaciones de entrada antes de enviarlas al sistema SQL.

```
FORALL index IN lower_bound .. upper_bound
  [SAVE EXCEPTIONS]
  sql_statement;
```

- La palabra clave **BULK COLLECT** indica al sistema SQL que debe realizar un enlace en bloque de las recopilaciones de salida antes de devolverlas al sistema PL/SQL.

```
... BULK COLLECT INTO
  collection_name [,collection_name] ...
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso del Enlace en Bloque

Utilice los enlaces en bloque para mejorar el rendimiento de:

- Sentencias DML que hacen referencia a elementos de recopilación
- Sentencias SELECT que hacen referencia a elementos de recopilación
- Bucles FOR de cursor que hacen referencia a recopilaciones y la cláusula RETURNING INTO

Palabras Clave para Soportar el Enlace en Bloque

La palabra clave FORALL indica al sistema PL/SQL que debe realizar un enlace en bloque de las recopilaciones de entrada antes de enviarlas al sistema SQL. Aunque la sentencia FORALL contiene un esquema de iteración, no es un bucle FOR.

La palabra clave BULK COLLECT indica al sistema SQL que debe realizar un enlace en bloque de las recopilaciones de salida, antes de devolverlas al sistema PL/SQL. Esto permite enlazar ubicaciones en las que SQL puede devolver los valores recuperados en bloque. Por lo tanto, puede utilizar estas palabras clave en las cláusulas SELECT INTO, FETCH INTO y RETURNING INTO.

La palabra clave SAVE EXCEPTIONS es opcional. Sin embargo, si algunas de las operaciones DML se realizan correctamente y otras fallan, deseará realizar un seguimiento de aquellas que fallen o crear un informe de ellas. Con la palabra clave SAVE EXCEPTIONS, se hará que las operaciones fallidas se almacenen en un atributo de cursor denominado %BULK_EXCEPTIONS, que es una recopilación de registros que indican el número de iteración DML en bloque y el correspondiente código de error.

Enlace en Bloque FORALL: Ejemplo

```
CREATE PROCEDURE raise_salary(percent NUMBER) IS
    TYPE numlist IS TABLE OF NUMBER
        INDEX BY BINARY_INTEGER;
    id  numlist;
BEGIN
    id(1) := 100; id(2) := 102;
    id(3) := 106; id(3) := 110;
    -- bulk-bind the PL/SQL table
    FORALL i IN id.FIRST .. id.LAST
        UPDATE employees
            SET salary = (1 + percent/100) * salary
            WHERE manager_id = id(i);
END;
/
```

```
EXECUTE raise_salary(10)
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Enlace en Bloque FORALL: Ejemplo

En el ejemplo de la transparencia, el bloque PL/SQL aumenta el salario de los empleados cuyo identificador de director es 100, 102, 104 o 110. Utiliza la palabra clave FORALL para enlazar en bloque la recopilación. Sin el enlace en bloque, el bloque PL/SQL enviaría una sentencia SQL al sistema SQL para cada empleado actualizado. Si hay muchos empleados que actualizar, el gran número de intercambios de contexto entre el sistema PL/SQL y el sistema SQL puede afectar al rendimiento. Sin embargo, la palabra clave FORALL enlaza en bloque la recopilación para mejorar el rendimiento.

Nota: Ya no es necesaria una construcción en bucle al utilizar esta función.

Para gestionar excepciones y para que se termine el enlace en bloque a pesar de los errores, agregue la palabra clave SAVE EXCEPTIONS a la sentencia FORALL después de los límites y antes de la sentencia DML. Utilice el nuevo atributo de cursor %BULK_EXCEPTIONS, que almacena una recopilación de registros con dos campos:

- %BULK_EXCEPTIONS(i).ERROR_INDEX contiene la "iteración" de la sentencia durante la que se produjo la excepción.
- %BULK_EXCEPTIONS(i).ERROR_CODE contiene el código de error correspondiente.

Los valores almacenados en %BULK_EXCEPTIONS hacen referencia a la sentencia FORALL ejecutada más recientemente. Sus archivos de comandos secundarios varían de 1 a %BULK_EXCEPTIONS.COUNT.

Enlace en Bloque FORALL: Ejemplo (continuación)

Atributo de Cursor Adicional para Operaciones DML

Otro atributo de cursor agregado para soportar operaciones en bloque es %BULK_ROWCOUNT. El atributo %BULK_ROWCOUNT es una estructura compuesta diseñada para su uso con la sentencia FORALL. Este atributo actúa como una tabla de índice. Su i^{o} elemento almacena el número de filas procesadas por la i^{a} ejecución de una sentencia UPDATE o DELETE. Si la i^{a} ejecución no afecta a ninguna fila, %BULK_ROWCOUNT(i) devuelve cero.

Por ejemplo:

```
CREATE TABLE num_table (n NUMBER);
DECLARE
    TYPE NumList IS TABLE OF NUMBER
        INDEX BY BINARY_INTEGER;
    nums NumList;
BEGIN
    nums(1) := 1;
    nums(2) := 3;
    nums(3) := 5;
    nums(4) := 7;
    nums(5) := 11;
    FORALL i IN nums.FIRST .. nums.LAST
        INSERT INTO num_table (n) VALUES (nums(i));
    FOR i IN nums.FIRST .. nums.LAST
        LOOP
            dbms_output.put_line('Inserted ' ||
                SQL%BULK_ROWCOUNT(i) || ' row(s)' ||
                ' on iteration ' || i);
        END LOOP;
    END;
/
DROP TABLE num_table;
```

Este ejemplo produce los siguientes resultados:

```
Inserted 1 row(s) on iteration 1
Inserted 1 row(s) on iteration 2
Inserted 1 row(s) on iteration 3
Inserted 1 row(s) on iteration 4
Inserted 1 row(s) on iteration 5

PL/SQL procedure successfully completed.
```

Uso de BULK COLLECT INTO con Consultas

La sentencia SELECT se ha mejorado para dar soporte a la sintaxis BULK COLLECT INTO. Por ejemplo:

```
CREATE PROCEDURE get_departments(loc NUMBER) IS
  TYPE dept_tabtype IS
    TABLE OF departments%ROWTYPE;
  depts dept_tabtype;
BEGIN
  SELECT * BULK COLLECT INTO depts
  FROM departments
  WHERE location_id = loc;
  FOR I IN 1 .. depts.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(depts(i).department_name
      || ' ' || depts(i).department_name);
  END LOOP;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de BULK COLLECT INTO con Consultas

En la base de datos Oracle 10g, al utilizar una sentencia SELECT en PL/SQL, puede usar la sintaxis de recopilación en bloque que se muestra en el ejemplo de la transparencia. De esta forma, puede obtener rápidamente un juego de filas sin usar un mecanismo de cursor.

El ejemplo lee todas las filas de departamentos para una región concreta en una tabla PL/SQL, cuyo contenido se muestra con el bucle FOR que sigue a la sentencia SELECT.

Uso de BULK COLLECT INTO con Cursos

La sentencia **FETCH** se ha mejorado para dar soporte a la sintaxis **BULK COLLECT INTO**. Por ejemplo:

```
CREATE PROCEDURE get_departments(loc NUMBER) IS
  CURSOR dept_csr IS SELECT * FROM departments
    WHERE location_id = loc;
  TYPE dept_tabtype IS TABLE OF dept_csr%ROWTYPE;
  depts dept_tabtype;
BEGIN
  OPEN dept_csr;
  FETCH dept_csr BULK COLLECT INTO depts;
  CLOSE dept_csr;
  FOR i IN 1 .. depts.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(depts(i).department_name
      || ' ' || depts(i).department_name);
  END LOOP;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de BULK COLLECT INTO con Cursos

En la base de datos Oracle 10g, al utilizar cursos en PL/SQL, puede usar un formato de la sentencia **FETCH** que soporte la sintaxis de recopilación en bloque que se muestra en el ejemplo de la transparencia.

En este ejemplo se muestra cómo se puede utilizar **BULK COLLECT INTO** con cursos.

También puede agregar una cláusula **LIMIT** para controlar el número de filas recuperadas en cada operación. El ejemplo de código de la transparencia se puede modificar de la siguiente forma:

```
CREATE PROCEDURE get_departments(loc NUMBER,
  nrows NUMBER) IS
  CURSOR dept_csr IS SELECT * FROM departments
    WHERE location_id = loc;
  TYPE dept_tabtype IS TABLE OF dept_csr%ROWTYPE;
  depts dept_tabtype;
BEGIN
  OPEN dept_csr;
  FETCH dept_csr BULK COLLECT INTO depts LIMIT nrows;
  CLOSE dept_csr;
  DBMS_OUTPUT.PUT_LINE(depts.COUNT || ' rows read');
END;
```

Uso de BULK COLLECT INTO con una Cláusula RETURNING

Ejemplo:

```
CREATE PROCEDURE raise_salary(rate NUMBER) IS
    TYPE emplist IS TABLE OF NUMBER;
    TYPE numlist IS TABLE OF employees.salary%TYPE
        INDEX BY BINARY_INTEGER;
    emp_ids emplist := emplist(100,101,102,104);
    new_sals numlist;
BEGIN
    FORALL i IN emp_ids.FIRST .. emp_ids.LAST
        UPDATE employees
            SET commission_pct = rate * salary
        WHERE employee_id = emp_ids(i)
        RETURNING salary BULK COLLECT INTO new_sals;
    FOR i IN 1 .. new_sals.COUNT LOOP ...
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de BULK COLLECT INTO con una Cláusula RETURNING

Se pueden utilizar enlaces en bloque para mejorar el rendimiento de los bucles FOR que hacen referencia a recopilaciones y devuelven DML. Si tiene, o planea tener, código PL/SQL que haga esto, puede utilizar la palabra clave FORALL junto con las palabras clave RETURNING y BULK COLLECT INTO para mejorar el rendimiento.

En el ejemplo que aparece en la transparencia, la información salary se recupera de la tabla EMPLOYEES y se recopila en la matriz new_sals. La recopilación new_sals se devuelve en bloque al sistema PL/SQL.

En el ejemplo de la transparencia se muestra un bucle FOR incompleto que se utiliza para iterar con los datos de nuevo salario recibidos de la operación UPDATE y procesar los resultados.

Uso de la Indicación NOCOPY

La indicación NOCOPY:

- Es una solicitud al compilador PL/SQL para transferir parámetros OUT e IN OUT por referencia en lugar de por valor
- Mejora el rendimiento mediante la reducción de la sobrecarga al transferir parámetros

```
DECLARE
    TYPE emptabtype IS TABLE OF employees%ROWTYPE;
    emp_tab emptabtype;
    PROCEDURE populate(tab IN OUT NOCOPY emptabtype)
    IS BEGIN ... END;
BEGIN
    populate(emp_tab);
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de la Indicación NOCOPY

Tenga en cuenta que los subprogramas PL/SQL soportan tres modos de transferencia de parámetros: IN, OUT e IN OUT. Por defecto:

- El parámetro IN se transfiere por referencia. Un puntero al parámetro real IN se transfiere al parámetro formal correspondiente. Por lo tanto ambos parámetros hacen referencia a la misma ubicación de la memoria, que contiene el valor del parámetro real.
- Los parámetros OUT e IN OUT se transfieren por valor. El valor del parámetro real OUT o IN OUT se copia en el parámetro formal correspondiente. A continuación, si se sale normalmente del subprograma, los valores asignados a los parámetros formales OUT e IN OUT se copian en los parámetros reales correspondientes.

Al utilizar NOCOPY con parámetros OUT o IN OUT que representen grandes estructuras de datos (como recopilaciones, registros e instancias de tipos de objetos), todo ello copiando con los parámetros OUT e IN OUT, se ralentiza la ejecución y se consume memoria. Para evitar esta sobrecarga, puede especificar la indicación NOCOPY, que permite al compilador PL/SQL transferir parámetros OUT e IN OUT por referencia.

En la transparencia se muestra un ejemplo de declaración de un parámetro IN OUT con la indicación NOCOPY.

Efectos de la Indicación NOCOPY

- **Si se sale del subprograma con una excepción no tratada:**
 - No puede confiar en los valores de los parámetros reales transferidos a un parámetro NOCOPY
 - No se realiza un “rollback” de las modificaciones incompletas
- **El protocolo de llamada a procedimiento remoto (RPC) permite transferir parámetros sólo por valor.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Efectos de la Indicación NOCOPY

Para proporcionar un mejor rendimiento, la indicación NOCOPY permite intercambiar semántica de excepción bien definida. Su uso afecta al manejo de excepciones de la siguiente forma:

- Puesto que NOCOPY es una indicación y no una directiva, el compilador puede transferir parámetros NOCOPY a un subprograma por valor o por referencia. Por lo tanto, si se sale de un subprograma con una excepción no tratada, no puede confiar en los valores de los parámetros reales NOCOPY.
- Por defecto, si se sale de un subprograma con una excepción no tratada, los valores asignados a sus parámetros formales OUT e IN OUT no se copian en los correspondientes parámetros reales y parece que se realiza un rollback de los cambios. Sin embargo, al especificar NOCOPY, las asignaciones a los parámetros formales inmediatamente afectan también a los parámetros reales. Por lo tanto, si se sale del subprograma con una excepción no tratada, no se realiza un “rollback” de los cambios (posiblemente no terminados).
- Actualmente, el protocolo RPC permite transferir parámetros sólo por valor. Así, la semántica de excepción puede cambiar sin notificación al realizar la partición de aplicaciones. Por ejemplo, si mueve un procedimiento local con parámetros NOCOPY a una ubicación remota, dichos parámetros ya no se transfieren por referencia.

La Indicación NOCOPY Se Puede Ignorar

La indicación NOCOPY no tiene ningún efecto si:

- **El parámetro real:**
 - Es un elemento de una tabla de índice
 - Está restringido (por ejemplo, por escala o NOT NULL)
 - Y el parámetro formal son registros, en los que uno o ambos se declararon mediante %ROWTYPE o %TYPE, y las restricciones en los campos correspondientes de los registros difieren
 - Necesita una conversión implícita del tipo de dato
- **El subprograma está relacionado con una llamada a procedimiento remoto o externo**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

La Indicación NOCOPY Se Puede Ignorar

En los siguientes casos, el compilador PL/SQL ignora la indicación NOCOPY y utiliza el método de transferencia de parámetros por valor (sin generar errores):

- El parámetro real es un elemento de una tabla de índice. Esta restricción no se aplica a tablas de índice completas.
- El parámetro real está restringido (por escala o NOT NULL). Esta restricción no se extiende a elementos o atributos restringidos. Asimismo, no se aplica a cadenas de caracteres restringidas por tamaño.
- Los parámetros real y formal son registros; uno o ambos se declararon mediante %ROWTYPE o %TYPE, y las restricciones en los campos correspondientes de los registros difieren.
- Los parámetros real y formal son registros; el parámetro real se declaró (implícitamente) como índice de un bucle FOR de cursor, y las restricciones en los campos correspondientes de los registros difieren.
- La transferencia del parámetro real necesita una conversión implícita de tipo de dato.
- El subprograma está relacionado con una llamada a procedimiento remoto o externo.

Indicación PARALLEL_ENABLE

La indicación PARALLEL_ENABLE:

- Se puede utilizar en funciones como indicación de optimización

```
CREATE OR REPLACE FUNCTION f2 (p1 NUMBER)
  RETURN NUMBER PARALLEL_ENABLE IS
BEGIN
  RETURN p1 * 2;
END f2;
```

- Indica que una función se puede utilizar en una consulta con paralelismo o en una sentencia DML con paralelismo

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Indicación PARALLEL_ENABLE

La palabra clave PARALLEL_ENABLE se puede utilizar en la sintaxis para declarar una función. Se trata de una indicación de optimización que indica que la función se puede utilizar en una consulta con paralelismo o en una sentencia DML con paralelismo. La función de ejecución en paralelo de Oracle divide el trabajo de ejecución de una sentencia SQL entre varios procesos. Las funciones llamadas desde una sentencia SQL que se ejecuta en paralelo pueden tener una copia independiente que se ejecuta en cada uno de estos procesos, donde se llama a cada copia sólo para el subjuego de filas manejadas por el proceso.

Para las sentencias DML, en las versiones anteriores a Oracle8i, la optimización del paralelismo buscaba si una función se anotaba como que tenía los cuatro RND\$, WND\$, RNPS y WNPS especificados en una declaración PRAGMA RESTRICT_REFERENCES; estas funciones marcadas como no de lectura ni de escritura en las variables del paquete o la base de datos se podían ejecutar en paralelo. De nuevo, estas funciones definidas con una sentencia CREATE FUNCTION tenían su código implícitamente examinado para determinar si realmente eran lo suficientemente puras; la ejecución con paralelismo podía producirse incluso aunque PRAGMA no se pudiera especificar en estas funciones.

La palabra clave PARALLEL_ENABLE se coloca después del tipo de valor de retorno de la declaración de la función, como se muestra en el ejemplo de la transparencia.

Nota: La función no debe utilizar estado de la sesión, como variables del paquete, porque puede que dichas variables no se compartan entre los servidores de ejecución en paralelo.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- **Crear constantes y excepciones estandarizadas mediante paquetes**
- **Desarrollar y llamar a subprogramas locales**
- **Controlar los privilegios en tiempo de ejecución de un subprograma mediante la definición de la directiva AUTHID**
- **Ejecutar transacciones autónomas**
- **Utilizar la cláusula RETURNING con sentencias DML y enlazar en bloque recopilaciones con las cláusulas FORALL y BULK COLLECT INTO**
- **Transferir parámetros por referencia mediante una indicación NOCOPY**
- **Permitir la optimización con indicaciones PARALLEL_ENABLE**



Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen

La lección permite comprender la gestión del código PL/SQL mediante la definición de constantes y excepciones en una especificación de paquete. Esto permite un alto grado de reutilización y estandarización de códigos.

Los subprogramas locales se pueden utilizar para simplificar y basar en módulos un bloque de código en el que la funcionalidad del subprograma se utilice de forma repetida en el bloque local.

Los privilegios de seguridad en tiempo de ejecución de un subprograma se pueden modificar mediante derechos del responsable de la definición o del invocador.

Las transacciones autónomas se pueden ejecutar sin que afecten a una transacción principal existente.

Debe comprender cómo obtener mejoras del rendimiento mediante la indicación NOCOPY, el enlace en bloque y las cláusulas RETURNING de las sentencias SQL, y la indicación PARALLEL_ENABLE para la optimización de las funciones.

Práctica 7: Visión General

En esta práctica se abordan los siguientes temas:

- **Creación de un paquete que utilice operaciones de recuperación en bloque**
- **Creación de un subprograma local para realizar una transacción autónoma para auditar una operación de negocio**
- **Prueba de la funcionalidad AUTHID**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica 7: Visión General

En esta práctica, creará un paquete que realice una recuperación en bloque de empleados de un departamento concreto. Los datos se almacenarán en una tabla PL/SQL del paquete. También proporcionará un procedimiento para visualizar el contenido de la tabla.

Agregará un procedimiento `add_employee` al paquete que inserte nuevos empleados. El procedimiento utilizará un subprograma autónomo local para escribir un registro log cada vez que se llame al procedimiento `add_employee`, tanto si agrega correctamente un registro como si no.

Por último, hará que el paquete utilice AUTHID de CURRENT_USER y probará el comportamiento con otro estudiante. Probará primero el código con derechos del responsable de la definición y, a continuación, con derechos del invocador.

Práctica 7

1. Actualice EMP_PKG con un nuevo procedimiento para consultar los empleados de un departamento especificado.
 - a. En la especificación, declare un procedimiento `get_employees`, con el parámetro denominado `dept_id` basado en el tipo de columna `employees.department_id`. Defina un tipo de índice PL/SQL como TABLE OF `EMPLOYEES%ROWTYPE`.
 - b. En el cuerpo del paquete, defina una variable privada denominada `emp_table` basada en el tipo definido en la especificación para almacenar los registros de los empleados. Implemente el procedimiento `get_employees` para recuperar en bloque los datos en la tabla.
 - c. Cree un nuevo procedimiento en la especificación y el cuerpo, denominado `show_employees`, que no toma argumentos y muestra el contenido de la variable de tabla PL/SQL privada (si existen datos).
Indicación: Utilice el procedimiento `print_employee`.
 - d. Llame al procedimiento `emp_pkg.get_employees` para el departamento 30 y, a continuación, llame a `emp_pkg.show_employees`. Repítalo para el departamento 60.
2. El director desea mantener un log cada vez que se llama al procedimiento `add_employee` en el paquete para insertar un nuevo empleado en la tabla EMPLOYEES.
 - a. En primer lugar, cargue y ejecute el archivo de comandos `E:\labs\PLPU\labs\lab_07_02_a.sql` para crear una tabla de log denominada `LOG_NEWEMP` y una secuencia denominada `log_newemp_seq`.
 - b. En el cuerpo del paquete, modifique el procedimiento `add_employee`, que realiza la operación real `INSERT`, para tener un procedimiento local denominado `audit_newemp`. El procedimiento `audit_newemp` debe utilizar una transacción autónoma para insertar un registro log en la tabla `LOG_NEWEMP`. Almacene el USER, la hora actual y el nombre del nuevo empleado en la fila de la tabla de log. Utilice `log_newemp_seq` para definir la columna `entry_id`.
Nota: Recuerde realizar una operación `COMMIT` en un procedimiento con una transacción autónoma.
 - c. Modifique el procedimiento `add_employee` para llamar a `audit_emp` antes de que realice la operación de inserción.
 - d. Llame al procedimiento `add_employee` para los siguientes nuevos empleados: Max Smart del departamento 20 y Clark Kent del departamento 10.
¿Qué sucede?
 - e. Consulte los dos registros EMPLOYEES que se han agregado y los registros de la tabla `LOG_NEWEMP`. ¿Cuántos registros log están presentes?
 - f. Ejecute una sentencia `ROLLBACK` para deshacer las operaciones de inserción que no se han confirmado. Utilice las mismas consultas del Ejercicio 2e: la primera para comprobar si las filas para los empleados Smart y Kent se han eliminado y la segunda para comprobar los registros log de la tabla `LOG_NEWEMP`. ¿Cuántos registros log están presentes? ¿Por qué?

Práctica 7 (continuación)

Si tiene tiempo, realice el siguiente ejercicio:

3. Modifique el paquete EMP_PKG para utilizar AUTHID de CURRENT_USER y pruebe el comportamiento con otro estudiante.

Nota: Verifique si existe en esta práctica la tabla LOG_NEWEMP del Ejercicio 2.

- a. Otorgue el privilegio EXECUTE a otro estudiante en el paquete EMP_PKG.
- b. Pídale a otro estudiante que llame al procedimiento add_employee para insertar al empleado Jaco Pastorius en el departamento 10. Recuerde anteponer el nombre del propietario del paquete al nombre del paquete. La llamada funcionará con los derechos del responsable de la definición.
- c. Ahora, ejecute una consulta de los empleados del departamento 10. ¿En la tabla de empleados de qué usuario se ha insertado el nuevo registro?
- d. Modifique la especificación del paquete EMP_PKG para utilizar AUTHID CURRENT_USER. Compile el cuerpo de EMP_PKG.
- e. Pídale al mismo estudiante que vuelva a ejecutar el procedimiento add_employee, para agregar al empleado Joe Zawinal en el departamento 10.
- f. Consulte a los empleados del departamento 10. ¿En qué tabla se agregó el nuevo empleado?
- g. Escriba una consulta para mostrar los registros que se han agregado en las tablas LOG_NEWEMP. Pídale al otro estudiante que consulte su propia copia de la tabla.

Oracle Internal & OAI Use Only

Gestión de Dependencias

8

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- **Realizar un seguimiento de las dependencias de procedimiento**
- **Predecir el efecto del cambio de un objeto de base de datos en funciones y procedimientos almacenados**
- **Gestionar dependencias de procedimiento**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetivos

Esta lección ofrece una introducción a las dependencias de objetos y a la recompilación implícita y explícita de objetos no válidos.

Descripción de las Dependencias

Objetos dependientes

Tabla
Vista
Disparador de base de datos
Procedimiento
Función
Cuerpo del paquete
Especificación del paquete
Objeto definido por el usuario y tipos de recopilaciones

Objetos a los que se hace referencia

Función
Especificación del paquete
Procedimiento
Secuencia
Sinónimo
Tabla
Vista
Objeto definido por el usuario y tipos de recopilaciones

Copyright © 2004, Oracle. Todos los Derechos Reservados.

ORACLE

Objetos Dependientes y a los que se Hace Referencia

Algunos objetos hacen referencia a otros objetos como parte de sus definiciones. Por ejemplo, un procedimiento almacenado puede contener una sentencia SELECT que seleccione columnas de una tabla. Por este motivo, el procedimiento almacenado se denomina objeto dependiente, mientras que la tabla es el objeto al que se hace referencia.

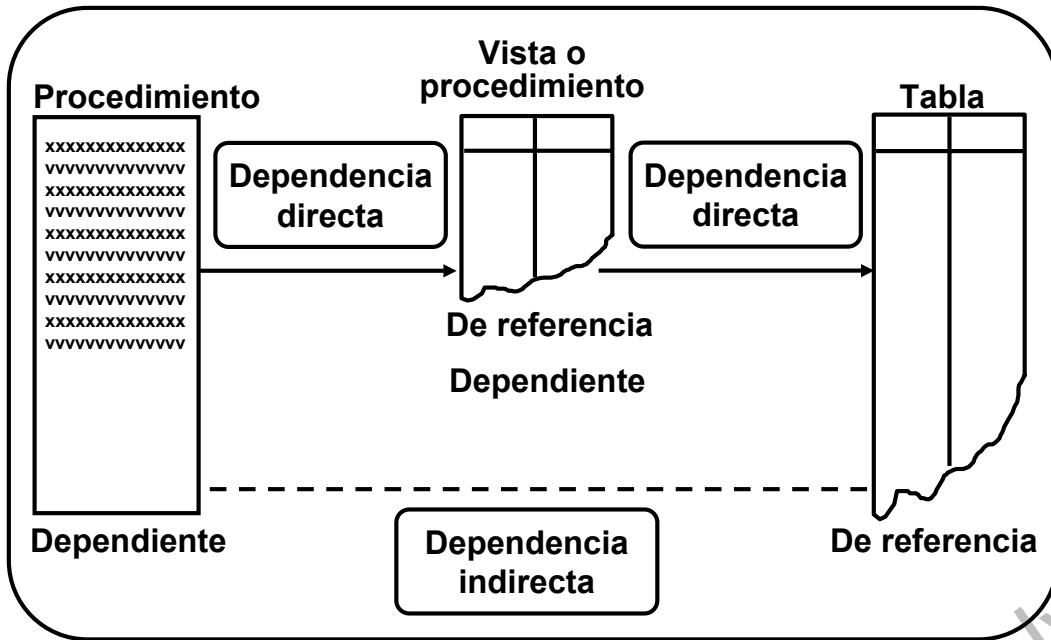
Problemas de la Dependencia

Si modifica la definición de un objeto al que se hace referencia, los objetos dependientes pueden o no seguir funcionando correctamente. Por ejemplo, si se cambia la definición de la tabla, el procedimiento puede o no seguir funcionando sin ningún error.

El servidor de Oracle registra de forma automática dependencias entre objetos. Para gestionar las dependencias, todos los objetos de esquema tienen un estado (válido o no válido) que se registra en el diccionario de datos. Puede ver el estado en la vista del diccionario de datos USER_OBJECTS.

Estado	Significado
VALID	El objeto de esquema se ha compilado y se puede utilizar inmediatamente cuando se haga referencia al mismo.
INVALID	El objeto de esquema se debe compilar antes de que se pueda utilizar.

Dependencias



ORACLE

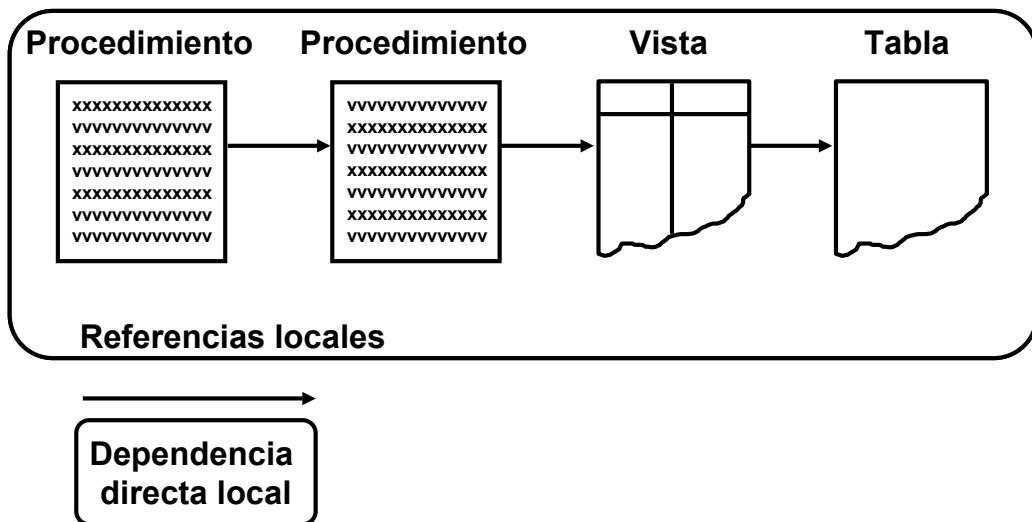
Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetos Dependientes y a los que se Hace Referencia (continuación)

Un procedimiento o una función puede hacer referencia directa o indirectamente (mediante una función, procedimiento, vista intermedia o función o procedimiento empaquetado) a los siguientes objetos:

- Tablas
- Vistas
- Secuencias
- Procedimientos
- Funciones
- Funciones o procedimientos empaquetados

Dependencias Locales



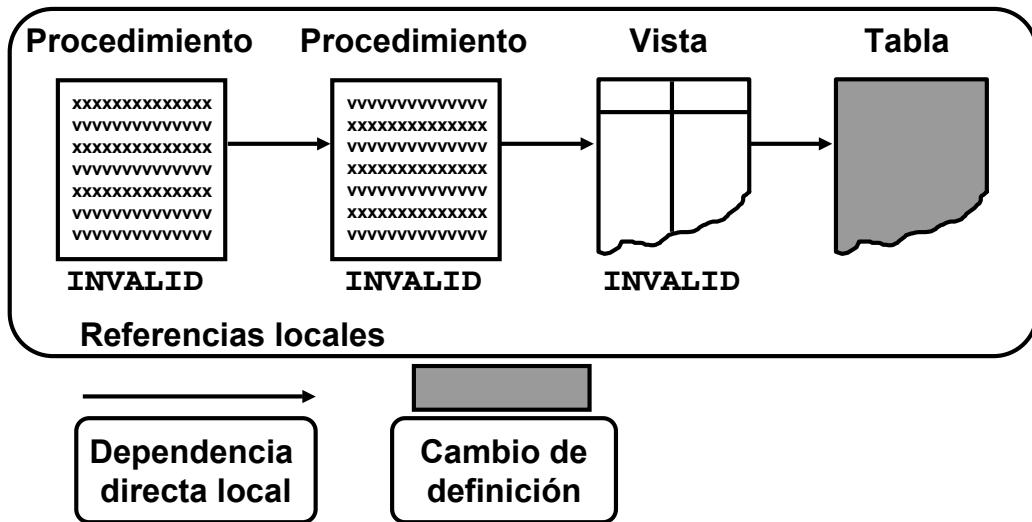
ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Gestión de Dependencias Locales

En el caso de dependencias locales, los objetos están en el mismo nodo de la misma base de datos. El servidor de Oracle gestiona de forma automática todas las dependencias locales con la tabla interna “dependiente” de la base de datos. Cuando se modifica un objeto al que se hace referencia, los objetos dependientes se invalidan. La próxima vez que se llame a un objeto invalidado, el servidor de Oracle lo recompilará de forma automática.

Dependencias Locales



El servidor Oracle recompila implícitamente cualquier objeto `INVALID` la próxima vez que sea llamado.

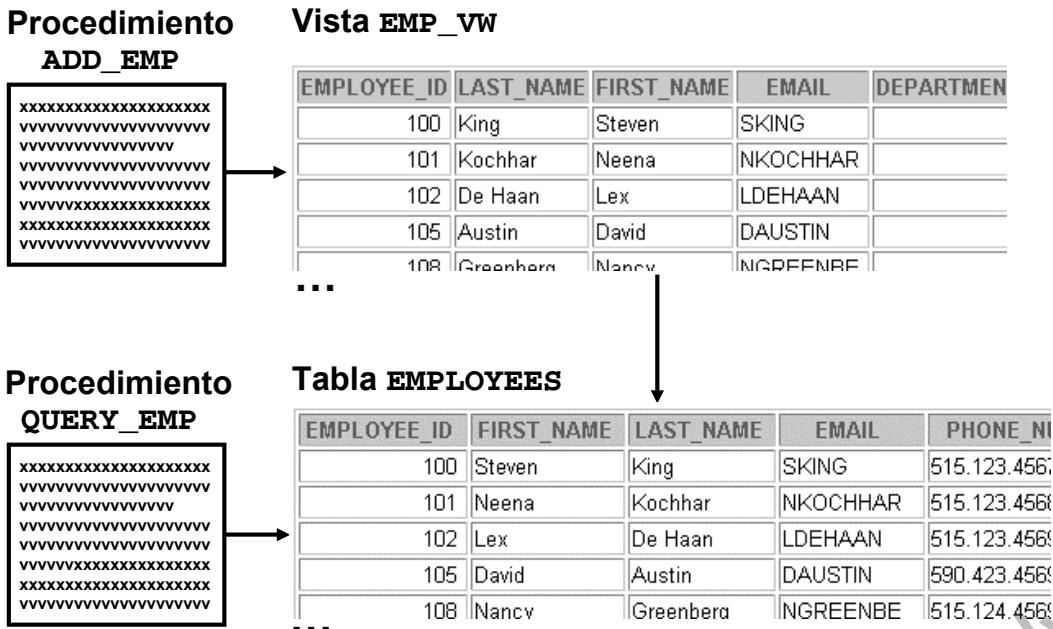
ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Gestión de Dependencias Locales (continuación)

Si se modifica la estructura de la tabla en la que se basa una vista, cuando describa la vista con el comando *iSQL*Plus DESCRIBE*, obtendrá un mensaje de error que indicará que el objeto no es válido para su descripción. Esto se debe a que no se trata de un comando SQL y, en este punto, la vista no es válida porque se ha cambiado la estructura de su tabla base. Si consulta la vista ahora, ésta se recompilará de forma automática y podrá ver el resultado si la recompilación se ha realizado correctamente.

Supuesto de Dependencias Locales



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ejemplo

El procedimiento QUERY_EMP hace referencia directa a la tabla EMPLOYEES. El procedimiento ADD_EMP actualiza la tabla EMPLOYEES indirectamente mediante la vista EMP_VW.

En cada uno de los siguientes casos, ¿se invalidará el procedimiento ADD_EMP y se recompilará correctamente?

1. Se ha modificado la lógica interna del procedimiento QUERY_EMP.
2. Se ha agregado una nueva columna a la tabla EMPLOYEES.
3. Se ha borrado la vista EMP_VW.

Visualización de Dependencias Directas mediante USER_DEPENDENCIES

```
SELECT name, type, referenced_name, referenced_type  
FROM user_dependencies  
WHERE referenced_name IN ('EMPLOYEES', 'EMP_VW');
```

NAME	TYPE	REFERENCED_NAME	REFERENCED_T
EMP_DETAILS_VIEW	VIEW	EMPLOYEES	TABLE
...			
EMP_VW	VIEW	EMPLOYEES	TABLE
...			
QUERY_EMP	PROCEDURE	EMPLOYEES	TABLE
ADD_EMP	PROCEDURE	EMP_VW	VIEW

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Visualización de Dependencias Directas mediante USER_DEPENDENCIES

Determine los objetos de base de datos que se van a recompilar de forma manual, mostrando las dependencias directas de la vista del diccionario de datos USER_DEPENDENCIES.

Examine las vistas ALL_DEPENDENCIES y DBA_DEPENDENCIES, que contienen la columna adicional OWNER, que hace referencia al propietario del objeto.

Columna	Descripción de la Columna
NAME	Nombre del objeto dependiente
TYPE	Tipo del objeto dependiente (PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER o VIEW)
REFERENCED_OWNER	Esquema del objeto al que se hace referencia
REFERENCED_NAME	Nombre del objeto al que se hace referencia
REFERENCED_TYPE	Tipo del objeto al que se hace referencia
REFERENCED_LINK_NAME	Enlace de base de datos que se utiliza para acceder al objeto al que se hace referencia

Visualización de Dependencias Directas e Indirectas

1. Ejecute el archivo de comandos `utldtree.sql` para crear los objetos que permiten mostrar las dependencias directas e indirectas.
2. Ejecute el procedimiento `DEPTREE_FILL`.

```
EXECUTE deptree_fill('TABLE', 'SCOTT', 'EMPLOYEES')
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Visualización de Dependencias Directas e Indirectas mediante Vistas Proporcionadas por Oracle

Puede visualizar las dependencias directas e indirectas desde las vistas adicionales del usuario denominadas DEPTREE y IDEPTREE que proporciona Oracle.

Ejemplo

1. Asegúrese de que se ha ejecutado el archivo de comandos `utldtree.sql`. Este archivo de comandos está ubicado en la carpeta `$ORACLE_HOME/rdbms/admin`. (Este archivo de comandos está en la carpeta `lab` de los archivos de clase.)
2. Rellene la tabla `DEPTREE_TEMPTAB` con información para un objeto determinado al que se hace referencia, llamando al procedimiento `DEPTREE_FILL`. Hay tres parámetros para este procedimiento:

<i>object_type</i>	Type of the referenced object
<i>object_owner</i>	Schema of the referenced object
<i>object_name</i>	Name of the referenced object

Visualización de Dependencias

Vista DEPTREE

```
SELECT nested_level, type, name  
FROM deptree  
ORDER BY seq#;
```

NESTED_LEVEL	TYPE	NAME
0	TABLE	EMPLOYEES
1	VIEW	EMP_DETAILS_VIEW
...		
1	TRIGGER	CHECK_SALARY
1	VIEW	EMP_VW
2	PROCEDURE	ADD_EMP
1	PACKAGE	MGR_CONSTRAINTS_PKG
2	TRIGGER	CHECK_PRES_TITLE
...		

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Visualización de Dependencias

Ejemplo

Puede visualizar una representación tabular de todos los objetos dependientes consultando la vista DEPTREE.

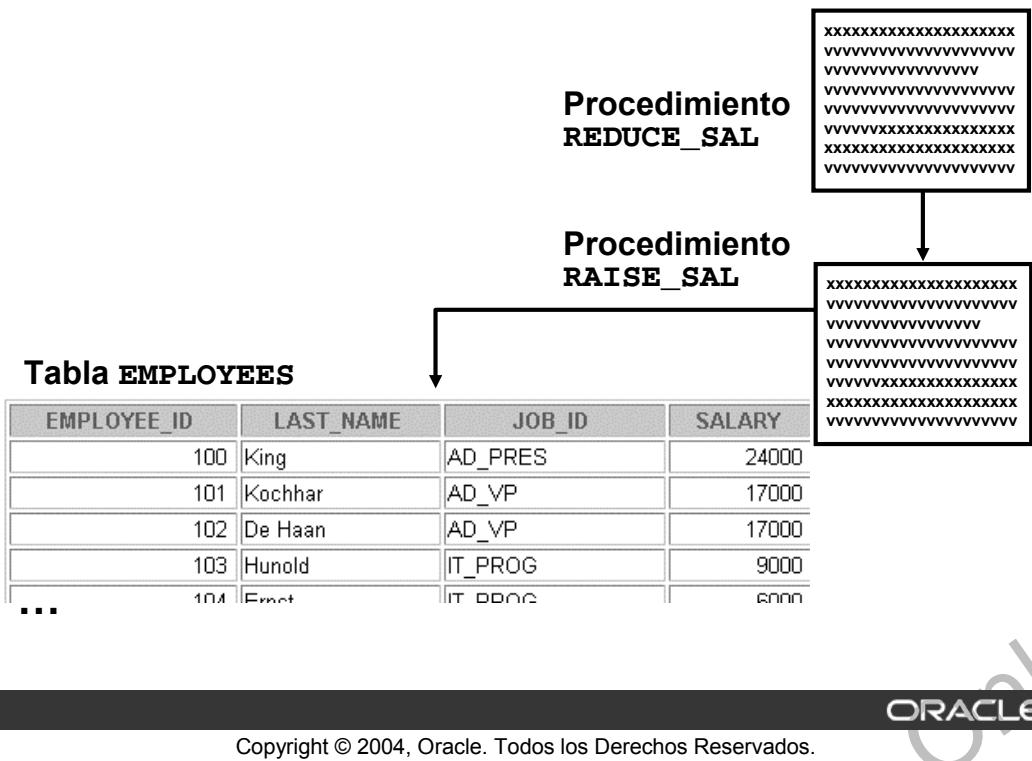
Realice una representación sangrada de la misma información consultando la vista IDEPTREE, que consta de una sola columna denominada DEPENDENCIES.

Por ejemplo,

```
SELECT *  
FROM ideptree;
```

proporciona una sola columna de salida sangrada de las dependencias en una estructura jerárquica.

Otro Supuesto de Dependencias Locales



Otro Supuesto de Dependencias Locales

Ejemplo 1

Prediga el efecto que produce un cambio en la definición de un procedimiento en la recompilación de un procedimiento dependiente.

Si el procedimiento RAISE_SAL actualiza la tabla EMPLOYEES directamente y el procedimiento REDUCE_SAL actualiza la tabla EMPLOYEES indirectamente mediante RAISE_SAL,

en los siguientes casos, ¿se recompilará el procedimiento REDUCE_SAL correctamente?

1. Se ha modificado la lógica interna del procedimiento RAISE_SAL.
2. Se ha eliminado uno de los parámetros formales para el procedimiento RAISE_SAL.

Supuesto de Dependencias Locales de Nomenclatura

Procedimiento
QUERY_EMP

xxxxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvvvv
vvvvvvvvvvvvvvv
vvvvvvvvvvvvvvv
vvvvvvvvvvvvvvv
vvvvvvvvvvvvvvv
vvvvxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx
vvvvvvvvvvvvvvv

Sinónimo público **EMPLOYEES**

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
104	Evert	IT_PROG	6000



Tabla
EMPLOYEES

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
104	Evert	IT_PROG	6000

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Supuesto de Dependencias Locales de Nomenclatura

Ejemplo 2

Tenga en cuenta el sutil caso en el que la creación de una tabla, vista o sinónimo puede invalidar de forma inesperada un objeto dependiente porque interfiera con la jerarquía del servidor de Oracle para la resolución de las referencias de nombre.

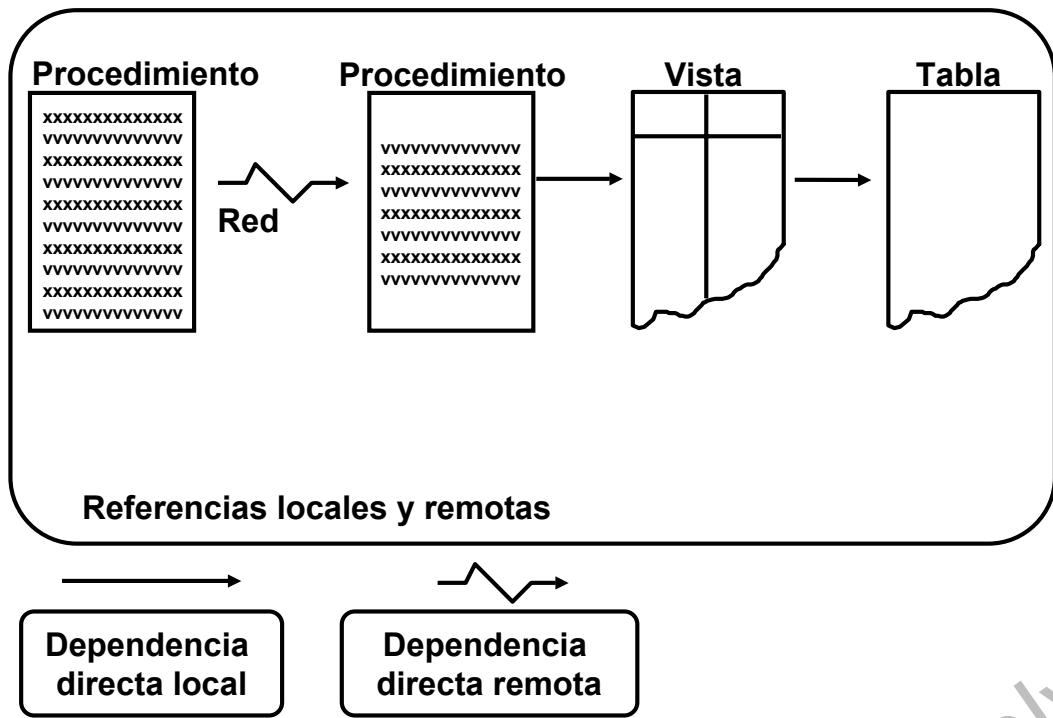
Prevea el efecto que tiene el nombre de un objeto nuevo sobre un procedimiento dependiente.

Si el procedimiento **QUERY_EMP** hizo referencia en un principio a un sinónimo público denominado **EMPLOYEES**, pero acaba de crear una tabla nueva denominada **EMPLOYEES** en su propio esquema, ¿invalida este cambio al procedimiento? ¿A cuál de los dos objetos **EMPLOYEES** hace referencia **QUERY_EMP** cuando se recompila el procedimiento?

Si borra la tabla privada **EMPLOYEES**, ¿se invalida el procedimiento? ¿Qué ocurre cuando se recompila el procedimiento?

Puede realizar un seguimiento de las dependencias de seguridad en la vista del diccionario de datos **USER_TAB_PRIVS**.

Descripción de las Dependencias Remotas



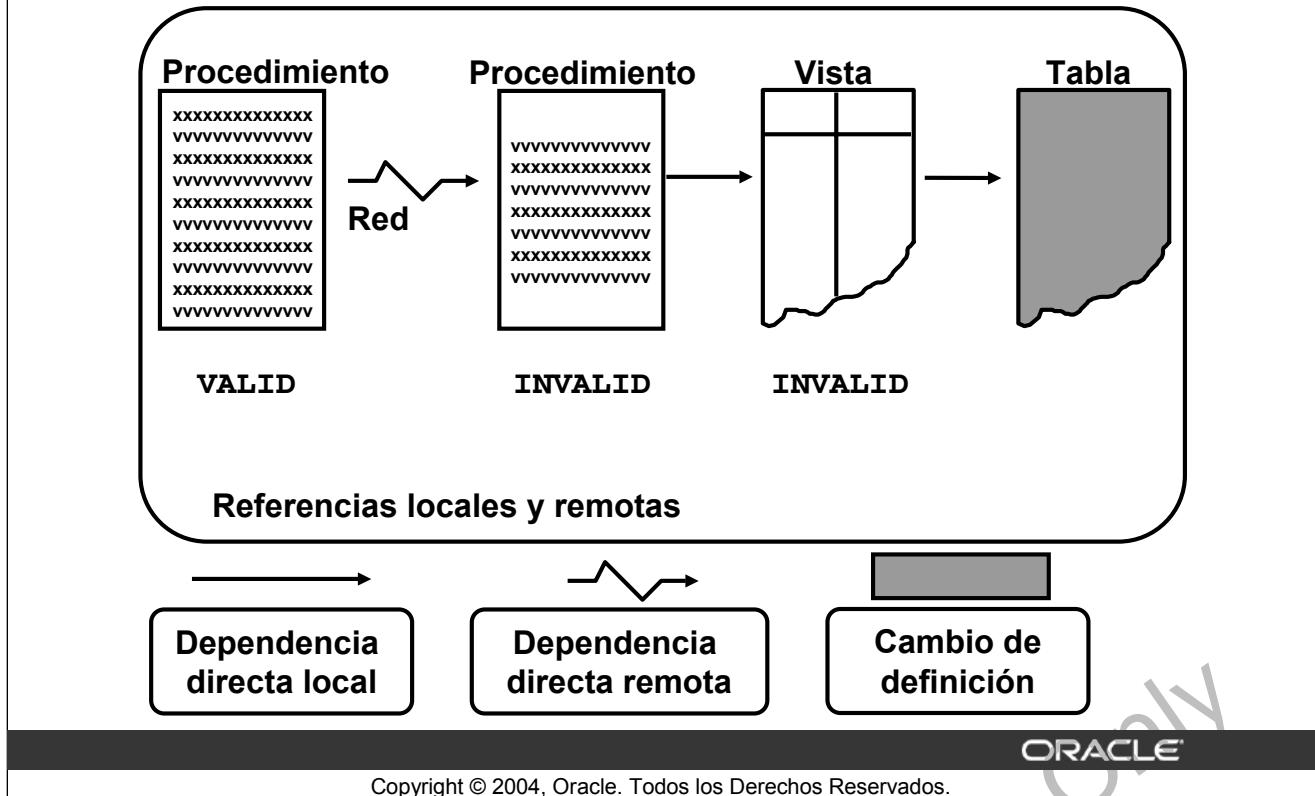
ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Descripción de las Dependencias Remotas

En el caso de dependencias remotas, los objetos están en nodos independientes. El servidor de Oracle no gestiona dependencias entre objetos remotos de esquema distintas de las dependencias de procedimiento local a procedimiento remoto (incluidas las funciones, los paquetes y los disparadores). El procedimiento local almacenado y todos sus objetos dependientes se invalidan pero no se recompilan de forma automática cuando se llaman por primera vez.

Descripción de las Dependencias Remotas



Descripción de las Dependencias Remotas (continuación)

Recompilación de Objetos Dependientes: Locales y Remotos

- Verifique la recompilación explícita correcta de los procedimientos remotos dependientes y la recompilación implícita de los procedimientos locales dependientes, comprobando el estado de estos procedimientos en la vista USER_OBJECTS.
- Si falla una recompilación implícita automática de los procedimientos locales dependientes, el estado seguirá siendo no válido y el servidor de Oracle emitirá un error de tiempo de ejecución. Por lo tanto, para evitar la interrupción de la producción, se recomienda recompilar los objetos dependientes locales de forma manual, en lugar de confiar en un mecanismo automático.

Conceptos de Dependencias Remotas

Las dependencias remotas se rigen por el modo que elija el usuario:

- **Comprobación de TIMESTAMP**
- **Comprobación de SIGNATURE**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Conceptos de Dependencias Remotas

Comprobación de **TIMESTAMP**

Cada unidad de programa PL/SQL incluye un registro de hora que se define cuando se crea o recompila. Siempre que modifique una unidad de programa PL/SQL o un objeto de esquema relevante, todas sus unidades de programa dependientes se marcarán como no válidas y se deberán recompilar antes de que se puedan ejecutar. La comparación de registro de hora real se produce cuando una sentencia del cuerpo de un procedimiento local llama a un procedimiento remoto.

Comprobación de **SIGNATURE**

El registro de hora y la firma se registran para cada unidad de programa PL/SQL. La firma de una construcción PL/SQL contiene información acerca de lo siguiente:

- El nombre de la construcción (procedimiento, función o paquete)
- Los tipos base de los parámetros de la construcción
- Los modos de los parámetros (IN, OUT o IN OUT)
- El número de los parámetros

El registro de hora registrado de la unidad de programa de llamada se compara con el registro de hora actual de la unidad llamada del programa remoto. Si los registros de hora coinciden, se continúa con la llamada. Si no coinciden, la capa de llamada a procedimiento remoto (RPC) realizará una comparación sencilla de la firma para determinar si la llamada es segura o no. Si la firma no se ha cambiado de forma incompatible, la ejecución continuará. De lo contrario, aparecerá un error.

Parámetro **REMOTE_DEPENDENCIES_MODE**

Definición de REMOTE_DEPENDENCIES_MODE:

- **Como parámetro init.ora**
REMOTE_DEPENDENCIES_MODE = value
- **A nivel de sistema**
ALTER SYSTEM SET
REMOTE_DEPENDENCIES_MODE = value
- **A nivel de sesión**
ALTER SESSION SET
REMOTE_DEPENDENCIES_MODE = value

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Parámetro **REMOTE_DEPENDENCIES_MODE**

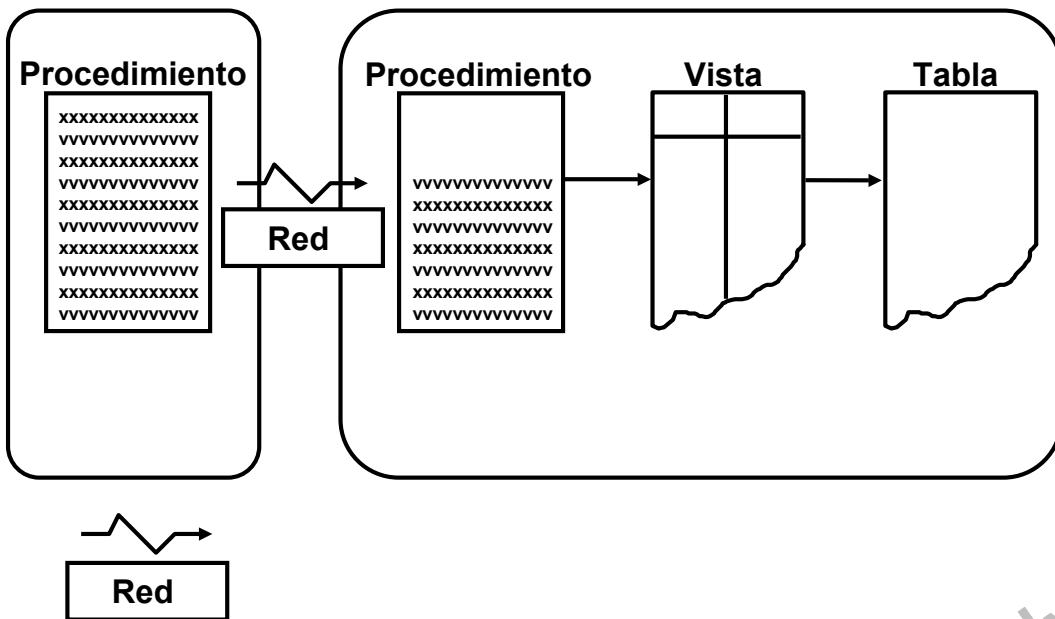
Definición de REMOTE_DEPENDENCIES_MODE

value **TIMESTAMP**
 SIGNATURE

Especifique el valor del parámetro **REMOTE_DEPENDENCIES_MODE** utilizando uno de los tres métodos que se describen en la transparencia.

Nota: La dirección de llamada determina el modelo de dependencia.

Dependencias Remotas y Modo de Registro de Hora



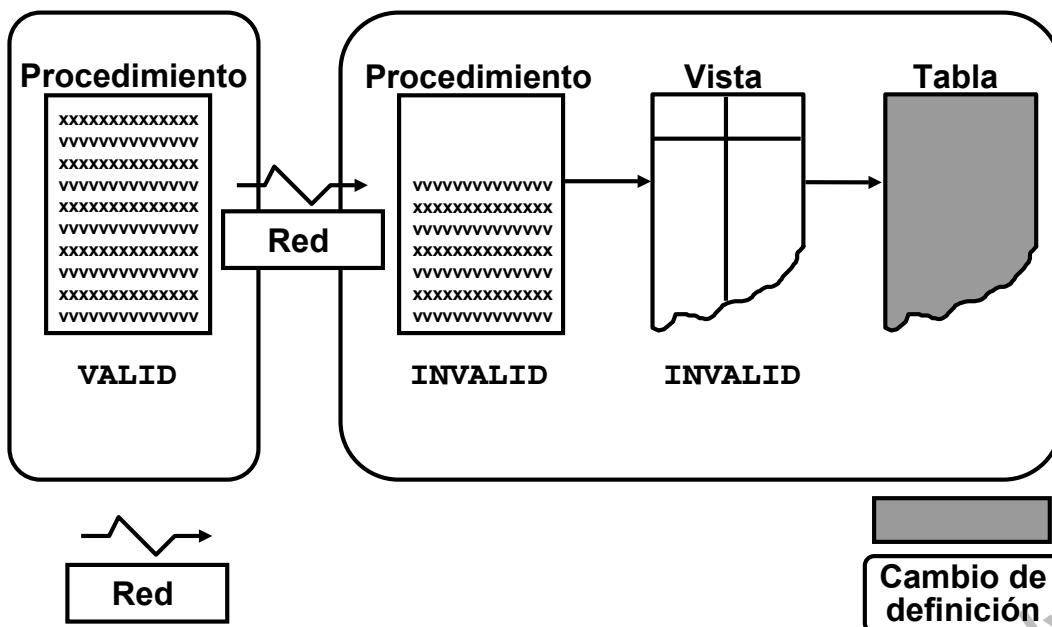
ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso del Modo de Registro de Hora para la Recompilación Automática de Objetos Locales y Remotos

Si los registros de hora se utilizan para manejar dependencias entre unidades de programa PL/SQL, siempre que modifique una unidad de programa o un objeto de esquema relevante, todas sus unidades dependientes se marcarán como no válidas y se deberán recompilar antes de que se puedan ejecutar.

Dependencias Remotas y Modo de Registro de Hora



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

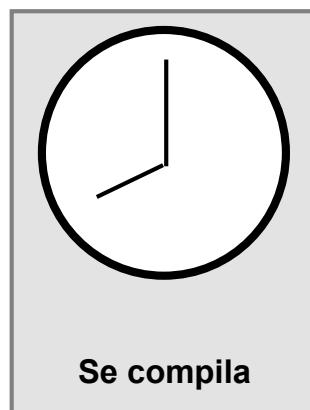
Uso del Modo de Registro de Hora para la Recompilación Automática de Objetos Locales y Remotos (continuación)

En el ejemplo de la transparencia, la definición de la tabla cambia. Por lo tanto, todas sus unidades dependientes se marcarán como no válidas y se deberán recompilar antes de que se puedan ejecutar.

- Cuando cambien los objetos remotos, se recomienda recompilar los objetos dependientes locales de forma manual para evitar la interrupción de la producción.
- El mecanismo de dependencia remota es distinto del mecanismo de dependencia local automática, tratado anteriormente. La primera vez que un subprograma local llame a un subprograma remoto recompilado, obtendrá un error de ejecución y se invalidará el subprograma local. La segunda vez que esto ocurra, tendrá lugar la recompilación automática implícita.

El Procedimiento Remoto B se Compila a las 8:00 a.m.

Procedimiento remoto B



Válido

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Procedimientos Locales que Hacen Referencia a Procedimientos Remotos

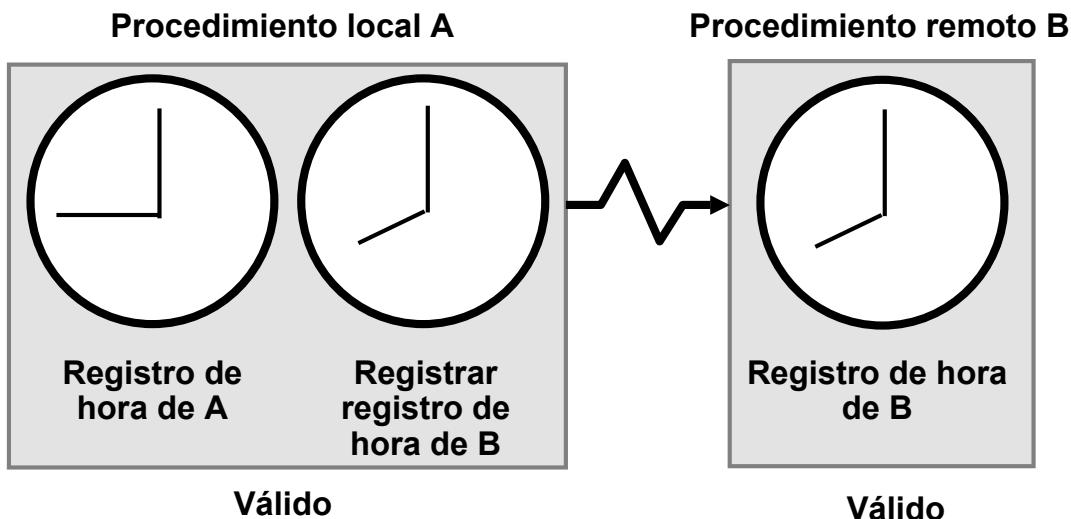
El servidor de Oracle invalidará un procedimiento local que haga referencia a un procedimiento remoto, si el procedimiento remoto se recompila después de compilar el procedimiento local.

Mecanismo de Dependencia Remota Automática

Cuando se compila un procedimiento, el servidor de Oracle registra el registro de hora de dicha compilación en el código P del procedimiento.

En la transparencia, cuando el procedimiento remoto B se compile correctamente a las 8:00 a.m., este tiempo se registrará como el registro de hora.

El Procedimiento Local A se Compila a las 9:00 a.m.



ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

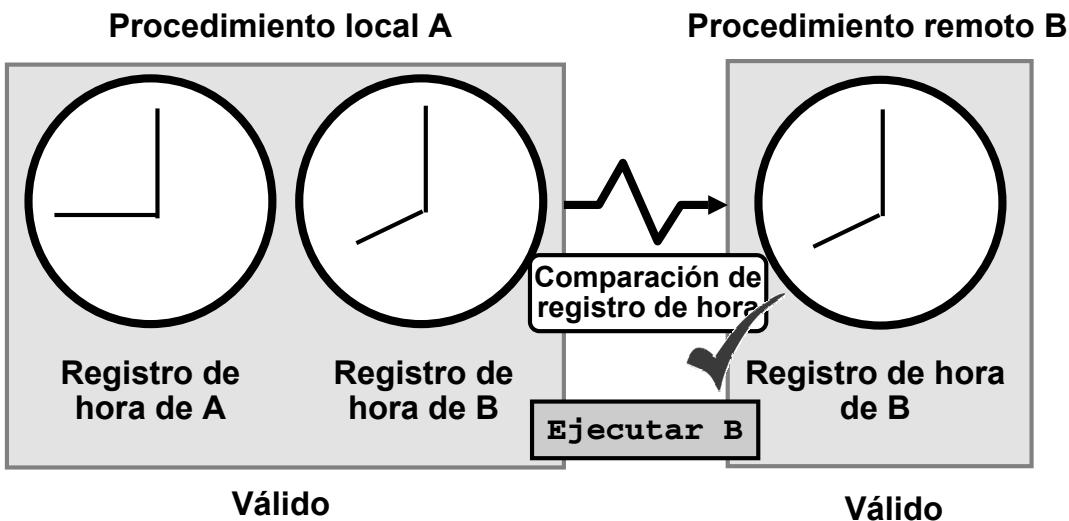
Procedimientos Locales que Hacen Referencia a Procedimientos Remotos (continuación)

Mecanismo de la Dependencia Remota Automática (continuación)

Cuando un procedimiento local que hace referencia a un procedimiento remoto se compila, el servidor de Oracle registrará también el registro de hora del procedimiento remoto del código P del procedimiento local.

En la transparencia, el procedimiento local A (que depende del procedimiento remoto B) se compila a las 9:00 a.m. Los registros de hora del procedimiento A y del procedimiento remoto B se registrarán en el código P del procedimiento A.

Ejecución del Procedimiento A



ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Dependencia Remota Automática

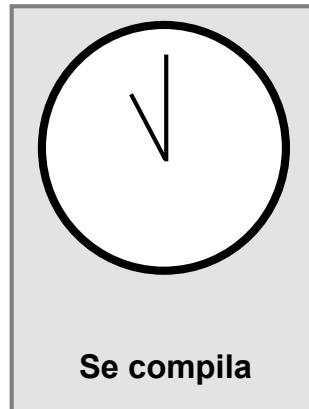
Cuando se llama al procedimiento local, el servidor de Oracle compara en tiempo de ejecución los dos registros de hora del procedimiento remoto al que se hace referencia.

Si los registros de hora son iguales (indicando que el procedimiento remoto no se ha recompilado), el servidor de Oracle ejecutará el procedimiento local.

En el ejemplo de la transparencia, el registro de hora registrado con el código P del procedimiento remoto B es igual al registrado con el procedimiento local A. Por lo tanto, el procedimiento local A es válido.

Procedimiento Remoto B Recompilado a las 11:00 a.m.

Procedimiento remoto B



Válido

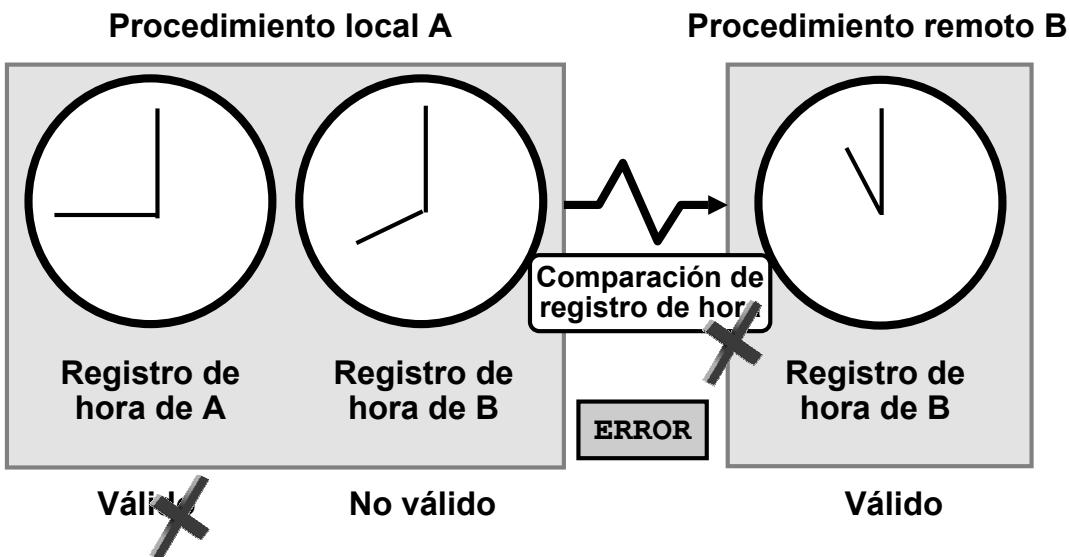
ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Procedimientos Locales que Hacen Referencia a Procedimientos Remotos

Si el procedimiento remoto B se ha recompilado correctamente a las 11:00 a.m, el nuevo registro de hora se registra con el código P.

Ejecución del Procedimiento A



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Dependencia Remota Automática

Si los registros de hora no son iguales (indicando que el procedimiento remoto se ha recompilado), el servidor de Oracle invalidará el procedimiento local y devolverá un error de tiempo de ejecución. Si el procedimiento local (que aparece ahora etiquetado como no válido) se llama una segunda vez, el servidor de Oracle lo recompilará antes de ejecutarlo, de acuerdo con el mecanismo de dependencia local automática.

Nota: Si un procedimiento local devuelve un error de tiempo de ejecución la primera vez que se llama (indicando que el registro de hora del procedimiento remoto ha cambiado), debe desarrollar una estrategia para volver a llamar al procedimiento local.

En el ejemplo de la transparencia, el procedimiento remoto se ha recompilado a las 11:00 a.m. y quedará registrado como el registro de hora en el código P. El código P del procedimiento local A sigue teniendo las 8:00 a.m. como registro de hora para el procedimiento remoto B. Puesto que el registro de hora registrado con el código P del procedimiento local A es distinto del registrado con el procedimiento remoto B, el procedimiento local se marcará como no válido. Cuando el procedimiento local se llame por segunda vez, se podrá compilar correctamente y marcar como válido.

Una desventaja del modo de registro de hora es que resulta innecesariamente restrictivo. La recompilación de los objetos dependientes en la red se suele realizar cuando no es estrictamente necesaria, dando lugar a una degradación del rendimiento.

Modo de Firma

- **La firma de un procedimiento consta de:**
 - El nombre del procedimiento
 - Los tipos de dato de los parámetros
 - Los modos de los parámetros
- **La firma del procedimiento remoto se guarda en el procedimiento local.**
- **Cuando se ejecuta un procedimiento dependiente, se compara la firma del procedimiento remoto al que se hace referencia.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Firmas

Para aliviar algunos de los problemas con el modelo de dependencia de sólo registro de hora, puede utilizar el modelo de firma. De este modo, se puede recompilar el procedimiento remoto sin que afecte a los procedimientos locales. Esto es importante si se distribuye la base de datos.

La firma de un subprograma contiene la siguiente información:

- El nombre del subprograma
- Los tipos de dato de los parámetros
- Los modos de los parámetros
- El número de los parámetros
- El tipo de dato del valor de retorno para una función

Si un programa remoto se cambia y se recompila pero no se cambia la firma, el procedimiento local podrá ejecutar el procedimiento remoto. Con el método de registro de hora, se habría producido un error porque los registros de hora no habrían coincidido.

Recompilación de una Unidad de Programa PL/SQL

Recompilación:

- **Se maneja automáticamente a través de la recompilación implícita en tiempo de ejecución**
- **Se maneja a través de la recompilación explícita con la sentencia ALTER**

```
ALTER PROCEDURE [SCHEMA.]procedure_name COMPILE;
```

```
ALTER FUNCTION [SCHEMA.]function_name COMPILE;
```

```
ALTER PACKAGE [SCHEMA.]package_name  
COMPILE [PACKAGE | SPECIFICATION | BODY];
```

```
ALTER TRIGGER trigger_name [COMPILE[DEBUG]] ;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Recompilación de Objetos PL/SQL

Si la recompilación se realiza correctamente, el objeto será válido. De lo contrario, el servidor de Oracle devolverá un error y el objeto permanecerá como no válido. Al recomilar un objeto PL/SQL, el servidor de Oracle primero recompila los objetos no válidos de los que depende.

Procedimiento: Los objetos locales que dependen de un procedimiento (como procedimientos que llaman al procedimiento recompilado o cuerpos de paquetes que definen los procedimientos que llaman al procedimiento recompilado) también se invalidan.

Paquetes: La opción COMPILE PACKAGE recompila tanto el cuerpo como la especificación del paquete, independientemente de que no sean válidos. La opción COMPILE SPECIFICATION recompila la especificación del paquete. La recompilación de la especificación de un paquete invalida los objetos locales que dependen de la especificación, como los subprogramas que utilizan el paquete. Tenga en cuenta que el cuerpo de un paquete también depende de su especificación. La opción COMPILE BODY sólo recompila el cuerpo del paquete.

Disparadores: La recompilación explícita elimina la necesidad de la recompilación implícita en tiempo de ejecución y evita los errores de compilación asociados en tiempo de ejecución y la sobrecarga de rendimiento.

La opción DEBUG indica al compilador PL/SQL que genere y almacene el código para que lo utilice el depurador PL/SQL.

Recompilación Incorrecta

La recompilación de las funciones y los procedimientos dependientes es incorrecta cuando:

- Se borra o se cambia el nombre del objeto al que se hace referencia**
- Se cambia el tipo de dato de la columna a la que se hace referencia**
- Se borra la columna a la que se hace referencia**
- Una vista a la que se hace referencia se sustituye por una vista con distintas columnas**
- Se modifica la lista de parámetros de un procedimiento al que se hace referencia**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Recompilación Incorrecta

A veces, la recompilación de procedimientos dependientes es incorrecta (por ejemplo, cuando se borra o se cambia el nombre de una tabla a la que se hace referencia).

El éxito de cualquier recompilación se basa en la dependencia exacta. Si se vuelve a crear una vista a la que se hace referencia, será necesario recompilar los objetos que dependan de dicha vista. El éxito de la recompilación dependerá de las columnas que la vista contenga ahora, así como de las columnas que los objetos dependientes necesiten para su ejecución. Si las columnas necesarias no forman parte de la nueva vista, el objeto seguirá siendo no válido.

Recompilación Correcta

La recompilación de las funciones y los procedimientos dependientes es correcta si:

- **La tabla a la que se hace referencia tiene nuevas columnas**
- **El tipo de dato de las columnas a las que se hace referencia no ha cambiado**
- **Se borra una tabla privada, pero existe una tabla pública con el mismo nombre y la misma estructura**
- **El cuerpo PL/SQL de un procedimiento al que se hace referencia se ha modificado y recompilado correctamente**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Recompilación Correcta

La recompilación de los objetos dependientes es correcta si:

- Se agregan nuevas columnas a la tabla a la que se hace referencia
- Todas las sentencias INSERT incluyen una lista de columnas
- Ninguna columna nueva se define como NOT NULL

Cuando un procedimiento dependiente hace referencia a una tabla privada y la tabla se borra, el estado del procedimiento dependiente pasa a ser no válido. Cuando el procedimiento se recompila (explícita o implícitamente) y existe una tabla pública, la recompilación se realizará correctamente pero el procedimiento pasará a depender de la tabla pública. La recompilación se realizará correctamente sólo si la tabla pública contiene las columnas que necesita el procedimiento. De lo contrario, el estado del procedimiento seguirá siendo no válido.

Recompilación de Procedimientos

Minimizar los fallos de dependencia mediante:

- **La declaración de registros con el atributo %ROWTYPE**
- **La declaración de variables con el atributo %TYPE**
- **La consulta con la notación SELECT ***
- **La incorporación de una lista de columnas con las sentencias INSERT**

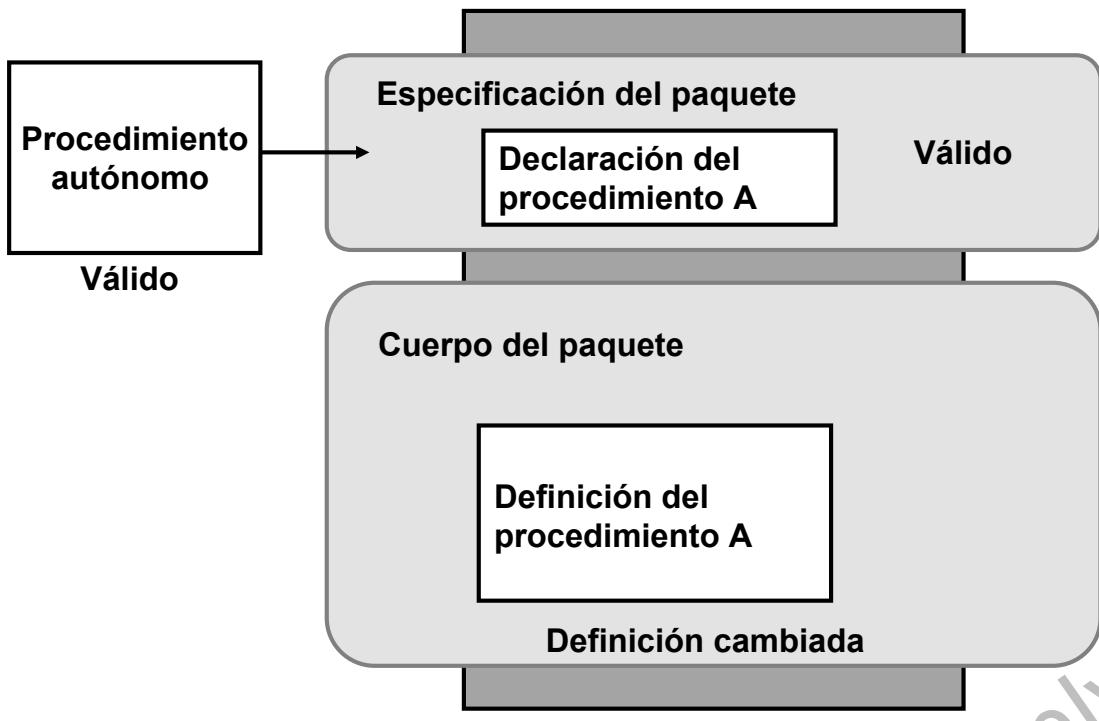
ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Recompilación de Procedimientos

Puede minimizar los fallos de recompilación siguiendo las instrucciones que se muestran en la transparencia.

Paquetes y Dependencias



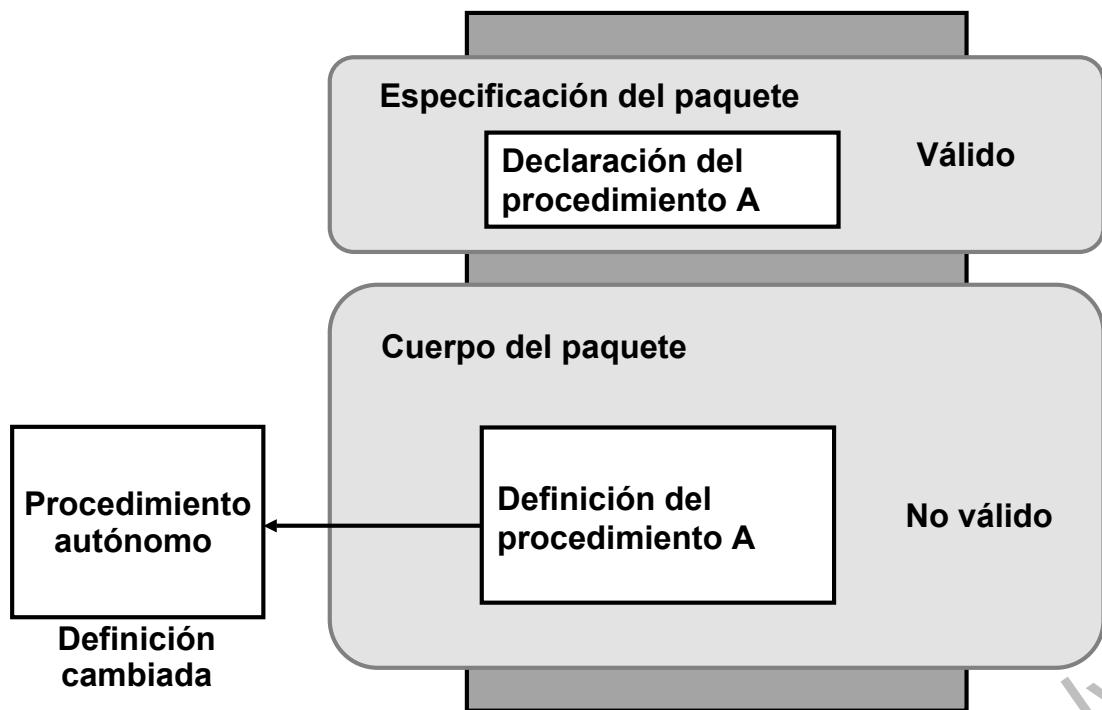
Copyright © 2004, Oracle. Todos los Derechos Reservados.

Gestión de Dependencias

Puede simplificar la gestión de dependencias con paquetes cuando se haga referencia a una función o procedimiento del paquete desde una función o procedimiento autónomo.

- Si el cuerpo del paquete cambia y la especificación del paquete no lo hace, el procedimiento autónomo que hace referencia a la construcción de un paquete seguirá siendo válido.
- Si la especificación del paquete cambia, el procedimiento exterior que hace referencia a la construcción de un paquete se invalidará, al igual que el cuerpo del paquete.

Paquetes y Dependencias



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Gestión de Dependencias (continuación)

Si un procedimiento autónomo al que se hace referencia en el paquete cambia, se invalidará todo el cuerpo del paquete, pero la especificación del paquete seguirá siendo válida. Por lo tanto, se recomienda poner el procedimiento en el paquete.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- **Realizar un seguimiento de los procedimientos dependientes**
- **Recompilar procedimientos de forma manual lo antes posible una vez que haya cambiado la definición de un objeto de base de datos**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen

Evite la interrupción de la producción realizando un seguimiento de los procedimientos dependientes y recompilándolos manualmente lo antes posible una vez que haya cambiado la definición de un objeto de base de datos.

Situación	Recompilación Automática
El procedimiento depende de un objeto local	Sí, en la primera reejecución
El procedimiento depende de un procedimiento remoto	Sí, pero en la segunda reejecución. Utilice la recompilación manual para la primera reejecución o vuelva a llamarlo una segunda vez.
El procedimiento depende de un objeto remoto distinto de un procedimiento	No

Práctica 8: Visión General

En esta práctica se abordan los siguientes temas:

- **Uso de DEPTREE_FILL y IDEPTREE para ver las dependencias**
- **Recompilación de procedimientos, funciones y paquetes**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica 8: Visión General

En esta práctica, utilice el procedimiento DEPTREE_FILL y la vista IDEPTREE para investigar las dependencias del esquema. Además, recompile vistas, paquetes, funciones y procedimientos no válidos.

Práctica 8

1. Responda a estas preguntas:
 - a. ¿Se puede invalidar una tabla o un sinónimo?
 - b. Considere el siguiente ejemplo de dependencia:

El procedimiento autónomo MY_PROC depende del procedimiento del paquete MY_PROC_PACK. Se cambia la definición del procedimiento MY_PROC_PACK mediante la recompilación del cuerpo del paquete. La declaración del procedimiento MY_PROC_PACK no se modifica en la especificación del paquete.

Dada esta situación, ¿se ha invalidado el procedimiento autónomo MY_PROC?
2. Cree una estructura de árbol que muestre todas las dependencias relacionadas con el procedimiento add_employee y con la función valid_deptid.

Nota: add_employee y valid_deptid se crearon en la lección 2 (“Creación de Funciones Almacenadas”). Puede ejecutar los archivos de comandos de la solución de la práctica 2 si tiene que crear el procedimiento y la función.

 - a. Cargue y ejecute el archivo de comandos utldtree.sql ubicado en la carpeta E:\lab\PLPU\labs.
 - b. Ejecute el procedimiento deptree_fill para el procedimiento add_employee.
 - c. Consulte la tabla IDEPTREE para ver los resultados.
 - d. Ejecute el procedimiento deptree_fill para la función valid_deptid.
 - e. Consulte la tabla IDEPTREE para ver los resultados.

Si tiene tiempo, realice el siguiente ejercicio:

3. Valide de forma dinámica los objetos no válidos.
 - a. Realice una copia de la tabla EMPLOYEES denominada EMPS.
 - b. Cambie la tabla EMPLOYEES y agregue la columna TOTSAL con el tipo de dato NUMBER(9, 2).
 - c. Cree y guarde una consulta para mostrar el nombre, el tipo y el estado de todos los objetos no válidos.
 - d. En compile_pkg (creado en la práctica 6 de la lección titulada “SQL Dinámico y Metadatos”), agregue un procedimiento denominado recompile que recompile todos los procedimientos, las funciones y los paquetes no válidos del esquema. Utilice SQL dinámico nativo para realizar operaciones ALTER en el tipo de objeto no válido así como operaciones COMPILE.
 - e. Ejecute el procedimiento compile_pkg.recompile.
 - f. Ejecute el archivo de comandos que ha creado en el paso 3c para comprobar el valor de la columna de estado. ¿Aún tiene objetos con estado INVALID?

Oracle Internal & OAI Use Only

Manipulación de Objetos Grandes

9

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- **Comparar y contrastar los tipos de dato LONG y LOB (objeto grande)**
- **Crear y mantener los tipos de dato LOB**
- **Diferenciar entre LOB internos y externos**
- **Utilizar el paquete PL/SQL DBMS_LOB**
- **Describir el uso de los LOB temporales**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

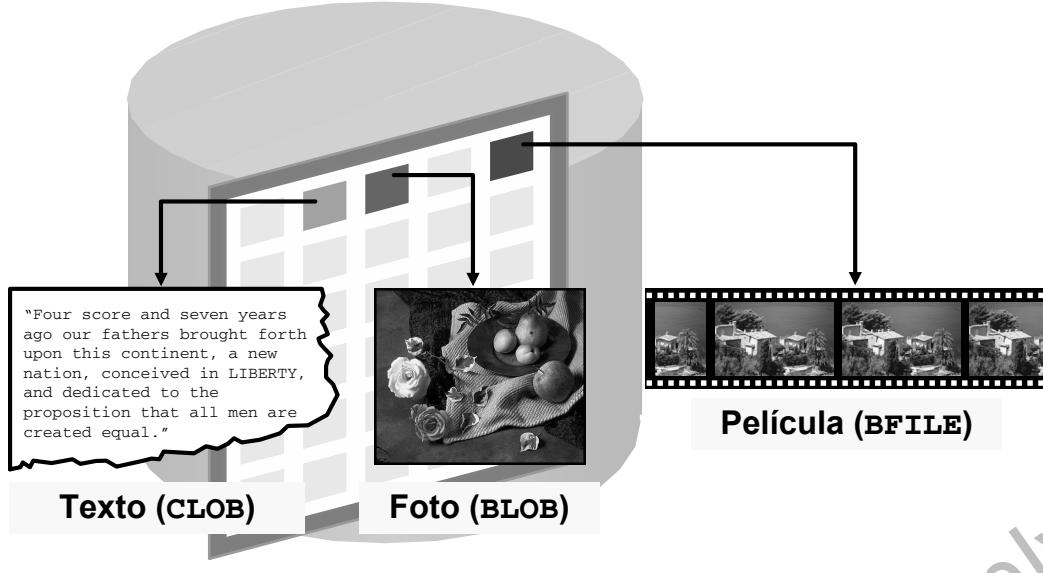
Objetivos

Las bases de datos se han utilizado durante mucho tiempo para almacenar objetos grandes. Sin embargo, los mecanismos incorporados en las bases de datos no han sido nunca tan útiles como los tipos de dato de objeto grande (LOB) que se han proporcionado a partir de Oracle8. Esta lección describe las características de los nuevos tipos de dato, comparándolos y contrastándolos con tipos de dato anteriores. Asimismo, incluye ejemplos, sintaxis y temas relacionados con los tipos LOB.

Nota: Un LOB es un tipo de dato y no se debe confundir con un tipo de objeto.

¿Qué es un LOB?

Los LOB se utilizan para almacenar datos grandes no estructurados, como texto, imágenes gráficas, películas y ondas sonoras.



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

LOB: Visión General

Un LOB es un tipo de dato que se utiliza para almacenar datos grandes no estructurados, como texto, imágenes gráficas, videos, etc. Los datos estructurados, como el registro de un cliente, pueden tener cientos de bytes, pero incluso pequeñas cantidades de datos multimedia pueden ser miles de veces más grandes. Asimismo, los datos multimedia pueden residir en archivos del sistema operativo, a los que puede ser necesario acceder desde una base de datos.

Existen cuatro tipos de dato de objeto grande:

- BLOB representa un objeto grande binario, como un videoclip.
- CLOB representa un objeto grande de caracteres.
- NCLOB representa un objeto grande de caracteres multibyte.
- BFILE representa un archivo binario almacenado en un archivo binario del sistema operativo fuera de la base de datos. El atributo o la columna BFILE almacena un localizador de archivo que apunta a un archivo externo.

Los LOB se califican de dos formas, según las interpretaciones del servidor de Oracle (binario o de caracteres) y los aspectos de almacenamiento. Los LOB se pueden almacenar internamente (dentro de la base de datos) o en los archivos de host. Hay dos categorías de LOB:

- LOB internos (CLOB, NCLOB, BLOB): Almacenados en la base de datos
- Archivos externos (BFILE): Almacenados fuera de la base de datos

LOB: Visión General (continuación)

La base de datos Oracle 10g realiza la conversión implícita entre los tipos de dato CLOB y VARCHAR2. Las demás conversiones implícitas entre los LOB no son posibles. Por ejemplo, si el usuario crea una tabla T con una columna CLOB y una tabla S con una columna BLOB, los datos no serán transferibles de forma directa entre estas dos columnas.

Sólo se puede acceder a los BFILE en modo de sólo lectura desde un servidor de Oracle.

Oracle Internal & OAI Use Only

Comparación de los Tipos de Dato LONG y LOB

LONG y LONG RAW	LOB
Columna única LONG por tabla	Varias columnas LOB por tabla
Hasta 2 GB	Hasta 4 GB
SELECT devuelve los datos	SELECT devuelve el localizador
Datos almacenados en línea	Datos almacenados en línea o fuera de línea
Acceso secuencial a los datos	Acceso aleatorio a los datos

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Tipos de Dato LONG y LOB

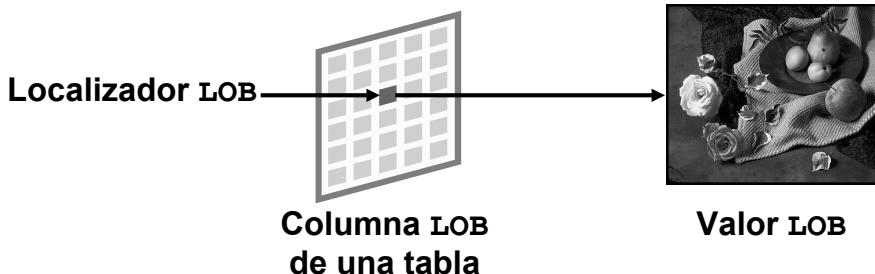
Los tipos de dato LONG y LONG RAW se han utilizado anteriormente para los datos no estructurados, como imágenes binarias, documentos o información geográfica. Estos tipos de dato se sustituyen por los tipos de dato LOB. La base de datos Oracle 10g proporciona una interfaz de programación de aplicaciones (API) LONG a LOB para la migración de columnas LONG a columnas LOB. La siguiente lista con viñetas compara la funcionalidad LOB con los tipos antiguos, donde los LONG hacen referencia a LONG y LONG RAW, y los LOB hacen referencia a todos los tipos de dato LOB:

- Una tabla puede tener varias columnas LOB y atributos del tipo de objeto. Una tabla sólo puede tener una columna LONG.
- El tamaño máximo de los LONG es de 2 GB; los LOB pueden tener hasta 4 GB.
- Los LOB devuelven el localizador; los LONG devuelven los datos.
- Los LOB almacenan un localizador en la tabla y los datos en un segmento diferente, a no ser que los datos tengan menos de 4.000 bytes; los LONG almacenan todos los datos en el mismo bloque de datos. Además, los LOB permiten que los datos se almacenen en un tablespace y segmento independiente o en un archivo de host.
- Los LOB pueden ser atributos del tipo de objeto; los LONG no.
- Los LOB soportan un acceso de piecewise aleatorio a los datos a través de una interfaz similar a un archivo; los LONG están restringidos al acceso de piecewise secuencial.

La función TO_LOB se puede utilizar para convertir los valores LONG y LONG RAW de una columna en los valores LOB. Utilice esto en la lista SELECT de una subconsulta en una sentencia INSERT.

Anatomía de un LOB

La columna LOB almacena un localizador para el valor de LOB.



ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Componentes de un LOB

En un LOB hay dos partes distintas:

- **Valor LOB:** Los datos que constituyen el objeto real que se está almacenando
- **Localizador LOB:** Puntero a la ubicación del valor LOB almacenado en la base de datos

Independientemente del lugar en el que se almacene el valor de LOB, el localizador se almacenará en la fila. Puede considerar un localizador LOB como un puntero a la ubicación real del valor LOB.

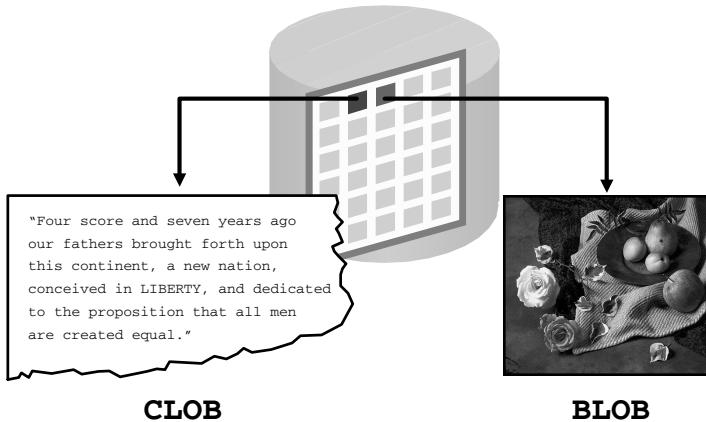
Una columna LOB no contiene los datos, sino el localizador del valor LOB.

Cuando un usuario crea un LOB interno, el valor se almacena en el segmento LOB y el localizador para el valor LOB fuera de línea se coloca en la columna LOB de la fila correspondiente de la tabla. Los LOB externos almacenan los datos fuera de la base de datos, por lo que en la tabla sólo se almacena un localizador para el valor LOB.

Para acceder a los LOB y manipularlos sin DML de SQL, debe crear un localizador LOB. Las interfaces programáticas operan con los valores LOB, utilizando los localizadores de forma similar a los manejadores de archivos del sistema operativo.

LOB Internos

El valor LOB se almacena en la base de datos.



ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Funciones de los LOB Internos

El LOB interno se almacena en el servidor de Oracle. Un BLOB, NCLOB o CLOB puede ser:

- Un atributo de un tipo definido por el usuario
- Una columna de una tabla
- Una variable de host o ligada
- Un resultado, parámetro o variable PL/SQL

Los LOB internos se pueden aprovechar de las funciones de Oracle, como:

- Mecanismos de simultaneidad
- Registro de redo y mecanismos de recuperación
- Transacciones con COMMIT o ROLLBACK

El servidor de Oracle interpreta el tipo de dato BLOB como un flujo de bits, similar al tipo de dato LONG RAW.

El tipo de dato CLOB se interpreta como un flujo de caracteres de un solo byte.

El tipo de dato NCLOB se interpreta como un flujo de caracteres de varios bytes, basado en la longitud de bytes del juego de caracteres nacional de la base de datos.

Gestión de los LOB Internos

- **Para interactuar totalmente con el LOB, las interfaces similares a los archivos se proporcionan en:**
 - El paquete PL/SQL DBMS_LOB
 - Oracle Call Interface (OCI)
 - Oracle Objects for OLE (enlace e incrustación de objetos)
 - Precompiladores Pro*C/C++ y Pro*COBOL
 - JDBC (Conectividad de Base de Datos Java)
- **El servidor de Oracle proporciona soporte para la gestión de LOB mediante SQL.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Gestión de los LOB

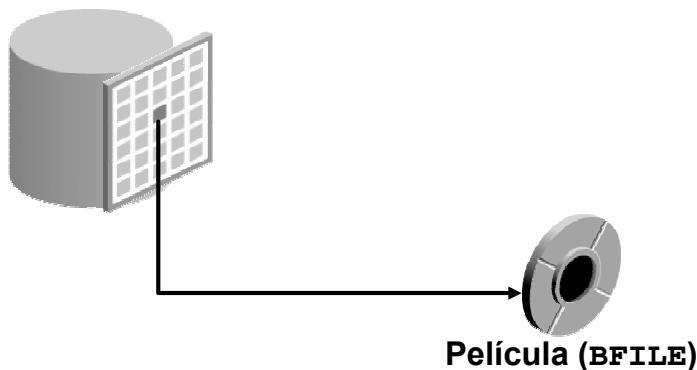
Para gestionar un LOB interno, realice los siguientes pasos:

1. Cree y rellene la tabla que contiene el tipo de dato LOB.
2. Declare e inicialice el localizador LOB en el programa.
3. Utilice SELECT FOR UPDATE para bloquear la fila que contiene el LOB en el localizador LOB.
4. Manipule el LOB con los procedimientos del paquete DBMS_LOB, las llamadas OCI, Oracle Objects for OLE, Oracle Precompilers o JDBC mediante el localizador LOB como referencia para el valor LOB.
También puede gestionar los LOB mediante SQL.
5. Utilice el comando COMMIT para que los cambios sean permanentes.

¿Qué son los BFILE?

El tipo de dato BFILE soporta un objeto grande externo o basado en archivos, como:

- **Atributos en un tipo de objeto**
- **Valores de columna en una tabla**



ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

¿Qué son los BFILE?

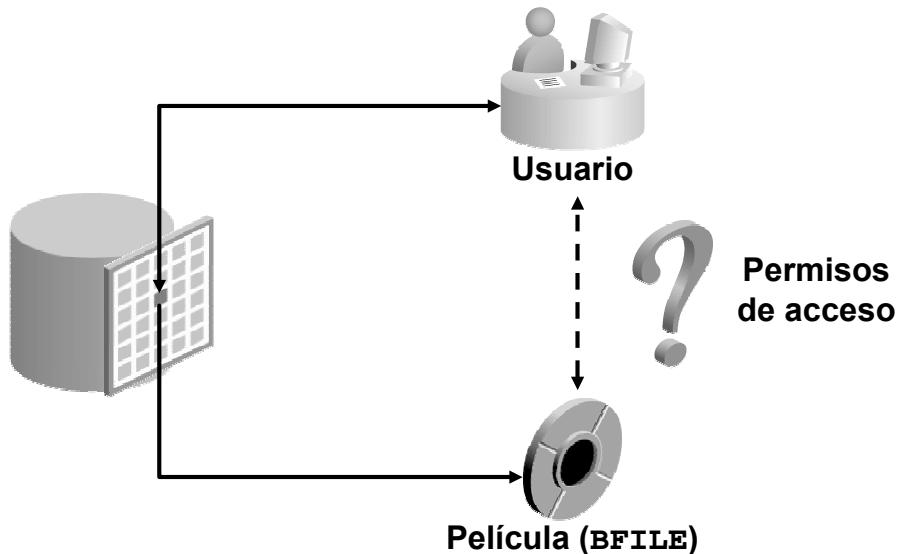
Los BFILE son objetos grandes externos (LOB) almacenados en archivos del sistema operativo externos a las tablas de la base de datos. El tipo de dato BFILE almacena un localizador para el archivo físico. Un BFILE puede estar en GIF, JPEG, MPEG, MPEG2, en texto o en otros formatos. Los LOB externos se pueden ubicar en discos duros, CD-ROM, CD gráficos u otros medios físicos, pero un único LOB no se puede ampliar de un medio o dispositivo a otro. El tipo de dato BFILE está disponible para que los usuarios de la base de datos puedan acceder al sistema de archivos externos. La base de datos Oracle 10g proporciona:

- Definición de los objetos BFILE
- Asociación de los objetos BFILE con los archivos externos correspondientes
- Seguridad para BFILE

Las demás operaciones, necesarias para el uso de BFILE, se pueden realizar mediante el paquete DBMS_LOB y OCI. Los BFILE son de sólo lectura y no participan en transacciones. El sistema operativo debe proporcionar el soporte para la integridad y durabilidad. El archivo se debe crear y colocar en el directorio adecuado, ofreciendo privilegios al proceso de Oracle para leerlo. Si se suprime el LOB, el servidor de Oracle no suprimirá el archivo. El administrador de la base de datos (DBA), el Administrador del Sistema o el usuario podrán gestionar la administración de los archivos y las estructuras de los directorios del sistema operativo. El tamaño máximo de un objeto grande externo depende del sistema operativo pero no puede superar los cuatro GB.

Nota: Los BFILE están disponibles en la base de datos Oracle8 y versiones posteriores.

Protección de BFILE



ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Protección de BFILE

El acceso no autenticado a archivos de un servidor presenta un riesgo de seguridad. La base de datos Oracle 10g puede actuar como un mecanismo de seguridad para proteger el sistema operativo de accesos no seguros, al mismo tiempo que se elimina la necesidad de gestionar cuentas de usuario adicionales en un sistema de computadoras de empresa.

Ubicación de Archivos y Privilegios de Acceso

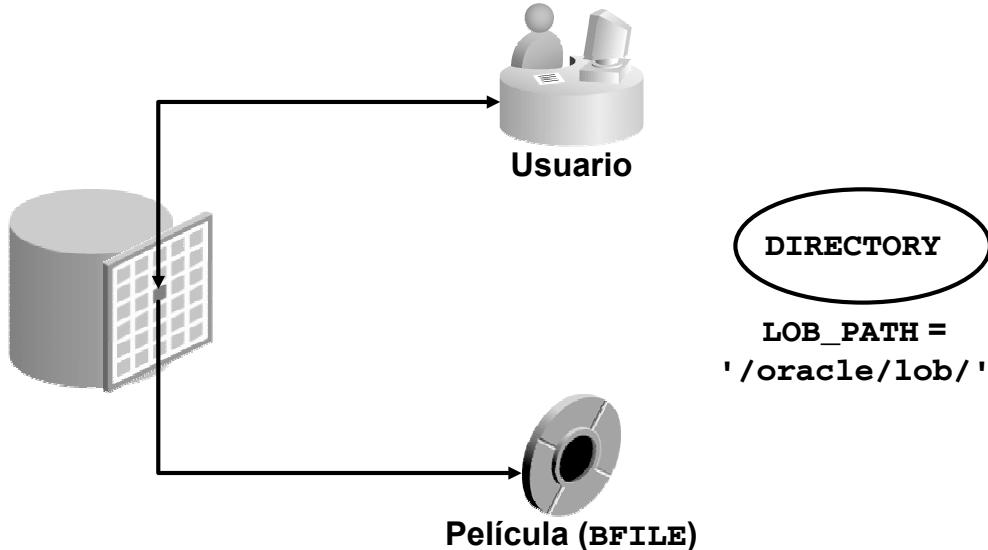
El archivo debe residir en la máquina en la que está la base de datos. El timeout para leer un BFILE que no existe, se basa en el valor del sistema operativo.

Puede leer un BFILE de la misma forma que lee un LOB interno. Sin embargo, podría haber restricciones relacionadas con el mismo archivo, como:

- Permisos de acceso
- Límites de espacio del sistema de archivos
- Manipulaciones de archivos que no son de Oracle
- Tamaño máximo del archivo del sistema operativo

La base de datos Oracle 10g no ofrece soporte transaccional a los BFILE. El sistema operativo y el sistema de archivos subyacente deben proporcionar el soporte para la integridad y durabilidad. Los métodos de copia de seguridad y recuperación de Oracle sólo soportan los localizadores LOB y no los BFILE físicos.

Nuevo Objeto de Base de Datos: DIRECTORY



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Nuevo Objeto de Base de Datos: DIRECTORY

Un DIRECTORY es un objeto de base de datos de no esquema que permite la administración del acceso y uso de los BFILE en la base de datos Oracle 10g.

Un DIRECTORY especifica un alias para un directorio del sistema de archivos del servidor en el que está ubicado BFILE. Al otorgar a los usuarios privilegios adecuados para estos elementos, puede proporcionar acceso seguro a los archivos de los directorios correspondientes por usuario (algunos directorios se pueden convertir a sólo lectura, a inaccesibles, etc.).

Además, estos alias del directorio se pueden utilizar mientras se hace referencia a los archivos (abrir, cerrar, leer, etc.) en PL/SQL y OCI. De esta forma, se proporciona abstracción de la aplicación a partir de los nombres de rutas de acceso codificados y se aporta flexibilidad a las ubicaciones de archivos de gestión portables.

El objeto DIRECTORY pertenece a SYS y ha sido creado por el DBA (o un usuario con el privilegio CREATE ANY DIRECTORY). Los objetos de directorio tienen privilegios de objeto, a diferencia de cualquier otro objeto de no esquema. Se pueden otorgar y revocar los privilegios para el objeto DIRECTORY. Los nombres lógicos de las rutas de acceso no están soportados.

Los permisos para el directorio real dependen del sistema operativo. Pueden diferir de los definidos para el objeto DIRECTORY y podrían cambiar después de la creación del objeto DIRECTORY.

Instrucciones para la Creación de Objetos DIRECTORY

- **No cree objetos DIRECTORY en rutas de acceso con archivos de base de datos.**
- **Limite el número de personas a las que se les otorgan los siguientes privilegios del sistema:**
 - CREATE ANY DIRECTORY
 - DROP ANY DIRECTORY
- **Todos los objetos DIRECTORY pertenecen a SYS.**
- **Cree rutas de acceso de directorios y permisos definidos correctamente antes de utilizar el objeto DIRECTORY para que el servidor de Oracle pueda leer el archivo.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Instrucciones para la Creación de Objetos DIRECTORY

Para asociar un archivo del sistema operativo a BFILE, debe crear primero un objeto DIRECTORY que es un alias para el nombre completo de la ruta de acceso.

Cree los objetos DIRECTORY siguiendo estas instrucciones:

- Los directorios deben apuntar a rutas de acceso que no contengan archivos de base de datos, ya que la manipulación de estos archivos puede dañar la base de datos. Actualmente, sólo se puede otorgar el privilegio READ al objeto DIRECTORY.
- Los privilegios del sistema CREATE ANY DIRECTORY y DROP ANY DIRECTORY se deben utilizar con cuidado y no se deben otorgar a los usuarios de forma indiscriminada.
- Los objetos DIRECTORY no son objetos de esquema y todos pertenecen a SYS.
- Cree las rutas de acceso de directorios con los permisos adecuados en el sistema operativo antes de crear el objeto DIRECTORY. Oracle no crea la ruta de acceso del sistema operativo.

Si migra la base de datos a otro sistema operativo, puede que necesite cambiar el valor de la ruta de acceso del objeto DIRECTORY.

La información del objeto DIRECTORY que cree con el comando CREATE DIRECTORY se almacena en las vistas del diccionario de datos DBA_DIRECTORIES y ALL_DIRECTORIES.

Gestión de los BFILE

El DBA o el Administrador del Sistema:

- 1. Crea un directorio del sistema operativo y proporciona archivos**
- 2. Crea un objeto DIRECTORY en la base de datos**
- 3. Otorga el privilegio READ en el objeto DIRECTORY a los usuarios adecuados de la base de datos**

El desarrollador o el usuario:

- 4. Crea una tabla de Oracle con una columna definida como tipo de dato BFILE**
- 5. Inserta filas en la tabla mediante la función BFILENAME para llenar la columna BFILE**
- 6. Escribe un subprograma PL/SQL que declara e inicializa un localizador LOB y lee BFILE**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Gestión de los BFILE

La gestión de los BFILE necesita de la cooperación entre la base de datos y el Administrador del Sistema y entre el desarrollador y el usuario de los archivos.

La base de datos o el Administrador del Sistema deben realizar las siguientes tareas con privilegios:

1. Crear el directorio del sistema operativo (como un usuario de Oracle) y definir permisos para que el servidor de Oracle pueda leer el contenido del directorio del sistema operativo. Cargar archivos en el directorio del sistema operativo.
2. Crear un objeto DIRECTORY de base de datos que haga referencia al directorio del sistema operativo.
3. Otorgar el privilegio READ en el objeto DIRECTORY de base de datos a los usuarios de la base de datos que necesiten el acceso al mismo.

El diseñador, el desarrollador de aplicaciones o el usuario deben realizar las siguientes tareas:

4. Crear una tabla de base de datos que contenga una columna definida como el tipo de dato BFILE.
5. Insertar filas en la tabla mediante la función BFILENAME para llenar la columna BFILE, asociando el campo a un archivo del sistema operativo en el DIRECTORY con nombre.

Gestión de los BFILE (continuación)

6. Escribir subprogramas PL/SQL que:
 - a. Declaren e inicialicen el localizador BFILE LOB.
 - b. Seleccionen la fila y la columna que contengan el BFILE en el localizador LOB.
 - c. Lean el BFILE con una función DBMS_LOB, mediante la referencia de archivos del localizador.

Oracle Internal & OAI Use Only

Preparación para Utilizar BFILE

1. Cree un directorio del sistema operativo para almacenar los archivos de datos físicos.

```
mkdir /temp/data_files
```

2. Cree un objeto DIRECTORY con el comando CREATE DIRECTORY.

```
CREATE DIRECTORY data_files  
AS '/temp/data_files';
```

3. Otorgue el privilegio READ en el objeto DIRECTORY a los usuarios adecuados.

```
GRANT READ ON DIRECTORY data_files  
TO SCOTT, MANAGER_ROLE, PUBLIC;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Preparación para Utilizar BFILE

Para utilizar BFILE en una tabla de Oracle, debe tener una tabla con una columna del tipo BFILE. Para que el servidor de Oracle pueda acceder a un archivo externo, el servidor debe conocer la ubicación física del archivo en la estructura de directorios del sistema operativo. El objeto DIRECTORY de base de datos proporciona el medio para especificar la ubicación de los BFILE. Utilice el comando CREATE DIRECTORY para especificar el puntero a la ubicación en la que están almacenados los BFILE. Necesita el privilegio CREATE ANY DIRECTORY.

Definición de sintaxis: CREATE DIRECTORY *dir_name* AS *os_path*;

En esta sintaxis, *dir_name* es el nombre del objeto de base de datos del directorio y *os_path* es la ubicación de los BFILE.

Los ejemplos de la transparencia muestran los comandos que se van a configurar:

- El directorio físico, por ejemplo /temp/data_files, en el sistema operativo
- Un objeto DIRECTORY con nombre, denominado data_files, que apunta al directorio físico del sistema operativo
- El derecho de acceso READ en el directorio que se va a otorgar a los usuarios de la base de datos, proporcionando el privilegio para leer los BFILE desde el directorio

Nota: El valor del parámetro de inicialización de la base de datos

SESSION_MAX_OPEN_FILES, que está definido en 10 por defecto, limita el número de BFILE que se pueden abrir en una sesión.

Relleno de Columnas BFILE con SQL

- Utilizar la función **BFILENAME** para inicializar una columna **BFILE**. La sintaxis de la función es la siguiente:

```
FUNCTION BFILENAME(directory_alias IN VARCHAR2,  
                   filename IN VARCHAR2)  
RETURN BFILE;
```

- Ejemplo:

- Agregar una columna **BFILE** a una tabla.

```
ALTER TABLE employees ADD video BFILE;
```

- Actualizar la columna con la función **BFILENAME**.

```
UPDATE employees  
SET video = BFILENAME('DATA_FILES', 'King.avi')  
WHERE employee_id = 100;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Relleno de Columnas BFILE con SQL

BFILENAME es una función incorporada que se utiliza para inicializar una columna BFILE, mediante los dos parámetros siguientes:

- *directory_alias* para el nombre del objeto DIRECTORY de base de datos que hace referencia al directorio del sistema operativo que contiene los archivos
- *filename* para el nombre del BFILE que se va a leer

La función BFILENAME crea un puntero (o un localizador LOB) para el archivo externo almacenado en un directorio físico, al que se le asigna un nombre de alias de directorio que se utiliza en el primer parámetro de la función. Rellene la columna BFILE mediante la función BFILENAME en una de las siguientes cláusulas:

- La cláusula VALUES de la sentencia INSERT
- La cláusula SET de la sentencia UPDATE

La operación UPDATE se puede utilizar para cambiar el destino de referencia del puntero de BFILE. También se puede inicializar la columna BFILE con un valor NULL y actualizarla más tarde con la función BFILENAME, como se muestra en la transparencia.

Después de asociar las columnas BFILE a un archivo, se pueden realizar operaciones de lectura posteriores en BFILE mediante el paquete PL/SQL DBMS_LOB y OCI. Sin embargo, estos archivos son de sólo lectura cuando se accede a ellos a través de BFILE. Por lo tanto, no se podrán actualizar ni suprimir mediante BFILE.

Relleno de Columnas BFILE con PL/SQL

```
CREATE PROCEDURE set_video(
    dir_alias VARCHAR2, dept_id NUMBER) IS
    filename VARCHAR2(40);
    file_ptr BFILE;
    CURSOR emp_csr IS
        SELECT first_name FROM employees
        WHERE department_id = dept_id FOR UPDATE;
    BEGIN
        FOR rec IN emp_csr LOOP
            filename := rec.first_name || '.gif';
            file_ptr := BFILENAME(dir_alias, filename);
            DBMS_LOB.FILEOPEN(file_ptr);
            UPDATE employees SET video = file_ptr
            WHERE CURRENT OF emp_csr;
            DBMS_OUTPUT.PUT_LINE('FILE: ' || filename ||
                ' SIZE: ' || DBMS_LOB.GETLENGTH(file_ptr));
            DBMS_LOB.FILECLOSE(file_ptr);
        END LOOP;
    END set_video;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Relleno de Columnas BFILE con PL/SQL

El ejemplo muestra un procedimiento PL/SQL denominado `set_video`, que acepta el nombre del alias de directorio que hace referencia al sistema de archivos del sistema operativo como un parámetro, así como un identificador de departamento. El procedimiento realiza las siguientes tareas:

- Utiliza un bucle FOR de cursor para obtener los registros de cada empleado
- Define `filename` agregando `.gif` al `last_name` del empleado
- Crea un localizador LOB en memoria para BFILE en la variable `file_ptr`
- Llama al procedimiento `DBMS_LOB.FILEOPEN` para verificar si el archivo existe y determinar el tamaño del archivo con la función `DBMS_LOB.GETLENGTH`
- Ejecuta una sentencia `UPDATE` para escribir el valor del localizador BFILE en la columna `video` BFILE
- Muestra el tamaño del archivo devuelto de la función `DBMS_LOB.GETLENGTH`
- Cierra el archivo con el procedimiento `DBMS_LOB.FILECLOSE`

Ejecución con la siguiente llamada:

```
EXECUTE set_video ('DATA_FILE', 60)
```

Resultados del ejemplo:

```
FILE: Alexander.gif SIZE: 5213
```

```
FILE: Bruce.gif SIZE: 26059
```

```
:
```

Uso de Rutinas DBMS_LOB con BFILE

La función DBMS_LOB.FILEEXISTS puede verificar si el archivo existe en el sistema operativo. La función:

- **Devuelve 0 si el archivo no existe**
- **Devuelve 1 si el archivo existe**

```
CREATE FUNCTION get_filesize(file_ptr BFILE)
  RETURN NUMBER IS
    file_exists      BOOLEAN;
    length NUMBER:= -1;
  BEGIN
    file_exists := DBMS_LOB.FILEEXISTS(file_ptr)=1;
    IF file_exists THEN
      DBMS_LOB.FILEOPEN(file_ptr);
      length := DBMS_LOB.GETLENGTH(file_ptr);
      DBMS_LOB.FILECLOSE(file_ptr);
    END IF;
    RETURN length;
  END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de Rutinas DBMS_LOB con BFILE

El procedimiento set_video de la página anterior terminará con una excepción si no existe el archivo. Para evitar que el bucle termine antes de tiempo, puede crear una función, como get_filesize, para determinar si un localizador BFILE determinado hace referencia a un archivo que realmente existe en el sistema de archivos del servidor. La función DBMS_LOB.FILEEXISTS espera el localizador BFILE como parámetro y devuelve INTEGER con:

- Un valor 0 si el archivo físico no existe
- Un valor 1 si el archivo físico existe

Si el parámetro BFILE no es válido, se puede dar una de las tres excepciones:

- NOEXIST_DIRECTORY si el directorio no existe
- NOPRIV_DIRECTORY si los procesos de la base de datos no tienen privilegios para el directorio
- INVALID_DIRECTORY si el directorio se ha invalidado después de abrir el archivo

En la función get_filesize, la salida de la función DBMS_LOB.FILEEXISTS se compara con el valor 1 y el resultado de la condición define la variable BOOLEAN file_exists. La llamada DBMS_LOB.FILEOPEN sólo se realiza si existe el archivo, evitando que se produzcan excepciones no deseadas. Si el archivo no existe, la función get_filesize devuelve un valor de -1. De lo contrario, devolverá el tamaño del archivo en bytes. Con esta información, el emisor de la llamada podrá realizar la acción adecuada.

Migración de LONG a LOB

La base de datos Oracle 10g activa la migración de las columnas LONG a las columnas LOB.

- **La migración de datos consiste en el procedimiento que mueve las tablas existentes que contienen columnas LONG para utilizar los LOB:**

```
ALTER TABLE [<schema>.] <table_name>
    MODIFY (<long_col_name> {CLOB | BLOB | NCLOB})
```

- **La migración de aplicaciones consiste en cambiar las aplicaciones LONG existentes para utilizar los LOB.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Migración de LONG a LOB

La base de datos Oracle 10g soporta la migración LONG a LOB mediante una API. En la migración de datos, es necesario mover las tablas existentes que contienen columnas LONG para utilizar las columnas LOB. Para ello, hay que utilizar el comando ALTER TABLE. En Oracle8i, ha sido necesario utilizar un operador denominado TO_LOB para copiar un LONG en un LOB. En la base de datos Oracle 10g, esta operación se puede realizar mediante la sintaxis que se muestra en la transparencia. Puede utilizar la sintaxis mostrada para:

- Convertir una columna LONG en una columna NCLOB o CLOB
- Convertir una columna LONG RAW en una columna BLOB

Las restricciones de la columna LONG (NULL y NOT NULL son las únicas restricciones permitidas) se mantienen para las nuevas columnas LOB. El valor por defecto especificado para la columna LONG se copia también en la nueva columna LOB. Por ejemplo, si tiene la siguiente tabla:

```
CREATE TABLE long_tab (id NUMBER, long_col LONG);
```

para cambiar la columna long_col de la tabla long_tab por el tipo de dato CLOB, utilice:

```
ALTER TABLE long_tab MODIFY (long_col CLOB);
```

Para obtener más información sobre las limitaciones de la migración LONG a LOB, consulte *Oracle Database Application Developer's Guide - Large Objects*. En la migración de aplicaciones, las aplicaciones LONG existentes cambian para el uso de LOB. Puede utilizar SQL y PL/SQL para acceder a los LONG y LOB. La API de migración LONG a LOB se proporciona para OCI y PL/SQL.

Migración de LONG a LOB

- **Conversión implícita: Desde una variable LONG (LONG RAW) o VARCHAR2 (RAW) a una variable CLOB (BLOB), y viceversa**
- **Conversión explícita:**
 - TO_CLOB() convierte LONG, VARCHAR2 y CHAR en CLOB.
 - TO_BLOB() convierte LONG RAW y RAW en BLOB.
- **Transferencia de parámetros de funciones y procedimientos:**
 - CLOB y BLOB como parámetros reales
 - VARCHAR2, LONG, RAW y LONG RAW son parámetros formales, y viceversa
- **Los datos LOB son aceptables en la mayoría de las funciones incorporadas y operadores SQL y PL/SQL.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Migración de LONG a LOB (continuación)

Con la nueva API LONG a LOB introducida en la base de datos Oracle 10g, las sentencias normales SQL y PL/SQL pueden hacer referencia a los datos de las columnas CLOB y BLOB.

Asignación implícita y transferencia de parámetros: La API de migración LONG a LOB soporta la asignación de una variable CLOB (BLOB) a una variable LONG (LONG RAW) o VARCHAR2(RAW), y viceversa.

Funciones de conversión explícita: En PL/SQL, las dos nuevas funciones siguientes de conversión explícita se han agregado a la base de datos Oracle 10g para convertir otros tipos de dato en CLOB y BLOB como parte de la migración LONG a LOB:

- TO_CLOB() convierte LONG, VARCHAR2 y CHAR en CLOB.
- TO_BLOB() convierte LONG RAW y RAW en BLOB.

Nota: TO_CHAR() se activa para convertir un CLOB en un tipo CHAR.

Transferencia de parámetros de funciones y procedimientos: Activa el uso de CLOB y BLOB como parámetros reales donde VARCHAR2, LONG, RAW y LONG RAW son parámetros formales, y viceversa. En los operadores y funciones incorporadas SQL y PL/SQL, un CLOB se puede transferir a las funciones incorporadas VARCHAR2 SQL y PL/SQL, comportándose exactamente igual que un VARCHAR2. O la variable VARCHAR2 se puede transferir a las API DBMS_LOB actuando como localizador LOB. Para obtener más información, consulte “Migrating from LONGs to LOBs” en *Application Developer’s Guide - Large Objects*.

Paquete DBMS_LOB

- **El trabajo con los LOB suele requerir el uso del paquete DBMS_LOB proporcionado por Oracle.**
- **DBMS_LOB proporciona rutinas para acceder a los LOB internos y externos y manipularlos.**
- **La base de datos Oracle 10g activa la recuperación de datos LOB directamente utilizando SQL sin una API LOB especial.**
- **En PL/SQL, puede definir un VARCHAR2 para un CLOB y un RAW para un BLOB.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Paquete DBMS_LOB

En versiones anteriores a Oracle9*i*, debe utilizar el paquete DBMS_LOB para la recuperación de datos desde los LOB. Para crear el paquete DBMS_LOB, los archivos de comandos dbmslob.sql y prvtlob.plb se deben ejecutar como SYS. El archivo de comandos catproc.sql ejecuta los archivos de comandos. A continuación, se podrán otorgar privilegios adecuados a los usuarios para utilizar el paquete.

El paquete no soporta ningún mecanismo de control de simultaneidad para las operaciones BFILE. El usuario es responsable de bloquear la fila que contiene el LOB interno de destino antes de llamar a los subprogramas que implican la acción de escribir en el valor LOB. Estas rutinas DBMS_LOB no bloquean implícitamente la fila que contiene el LOB.

Las dos constantes, LOBMAXSIZE y FILE_READONLY, definidas en la especificación del paquete se utilizan también en los procedimientos y funciones de DBMS_LOB. Por ejemplo, utilícelas para lograr el nivel máximo de pureza que se va a utilizar en las expresiones SQL. Los procedimientos y funciones DBMS_LOB se pueden clasificar, en líneas generales, en dos tipos: mutadores y observadores.

- Los mutadores pueden modificar los valores LOB: APPEND, COPY, ERASE, TRIM, WRITE, FILECLOSE, FILECLOSEALL y FILEOPEN.
- Los observadores pueden leer los valores LOB: COMPARE, FILEGETNAME, INSTR, GETLENGTH, READ, SUBSTR, FILEEXISTS y FILEISOPEN.

Paquete DBMS_LOB

- **Modifican los valores LOB:**
**APPEND, COPY, ERASE, TRIM, WRITE,
LOADFROMFILE**
- **Leen o examinan los valores LOB:**
GETLENGTH, INSTR, READ, SUBSTR
- **Específicos de BFILE:**
**FILECLOSE, FILECLOSEALL, FILEEXISTS,
FILEGETNAME, FILEISOPEN, FILEOPEN**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Paquete DBMS_LOB (continuación)

APPEND	Agrega el contenido del LOB de origen al LOB de destino
COPY	Copia todo o parte del LOB de origen en el LOB de destino
ERASE	Borra todo o parte de un LOB
LOADFROMFILE	Carga los datos BFILE en un LOB interno
TRIM	Recorta el valor LOB a una longitud menor especificada
WRITE	Escribe datos en el LOB desde el offset especificado
GETLENGTH	Obtiene la longitud del valor LOB
INSTR	Devuelve la posición de coincidencia de la <i>n</i> incidencia del patrón en el LOB
READ	Lee los datos del LOB desde el offset especificado
SUBSTR	Devuelve parte del valor LOB desde el offset especificado
FILECLOSE	Cierra el archivo
FILECLOSEALL	Cierra todos los archivos abiertos anteriormente
FILEEXISTS	Comprueba si el archivo existe en el servidor
FILEGETNAME	Obtiene el alias de directorio y el nombre del archivo
FILEISOPEN	Comprueba si el archivo se ha abierto con los localizadores BFILE de entrada
FILEOPEN	Abre un archivo

Paquete DBMS_LOB

- Los parámetros **NULL** obtienen devoluciones **NULL**.
- **Offsets:**
 - **BLOB, BFILE:** Se miden en bytes
 - **CLOB, NCLOB:** Se miden en caracteres
- **No hay ningún valor negativo para los parámetros.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de las Rutinas DBMS_LOB

Todas las funciones del paquete DBMS_LOB devuelven NULL si algún parámetro de entrada es NULL. Si el LOB/BFILE de destino se introduce como NULL, todos los procedimientos de los mutadores del paquete DBMS_LOB emitirán una excepción.

Sólo se permiten offsets positivos y absolutos. Representan el número de bytes o caracteres desde el principio de los datos LOB desde los que se inicia la operación. No se permiten los offsets negativos ni los rangos observados en los operadores y funciones de cadenas SQL. Las excepciones correspondientes se emiten si se produce una violación. El valor por defecto para un offset es 1, lo que indica el primer byte o carácter del valor LOB.

Del mismo modo, sólo se permiten valores de números naturales para el parámetro de cantidad (BUFSIZ). No se permiten los valores negativos.

DBMS_LOB.READ y DBMS_LOB.WRITE

```
PROCEDURE READ (
    lobsrc IN BFILE|BLOB|CLOB ,
    amount IN OUT BINARY_INTEGER,
    offset IN INTEGER,
    buffer OUT RAW|VARCHAR2 )
```

```
PROCEDURE WRITE (
    lobdst IN OUT BLOB|CLOB,
    amount IN OUT BINARY_INTEGER,
    offset IN INTEGER := 1,
    buffer IN RAW|VARCHAR2 ) -- RAW for BLOB
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

DBMS_LOB.READ

Llame al procedimiento READ para leer y devolver en piecewise una cantidad (AMOUNT) especificada de datos desde un LOB determinado, comenzando por OFFSET. La excepción se emite cuando no quedan más datos para leer desde el LOB de origen. El valor devuelto en AMOUNT es menor que el especificado, si se alcanza el final del LOB antes de que se pueda leer el número especificado de bytes o de caracteres. En el caso de los CLOB, el juego de caracteres de datos en BUFFER es igual al del LOB.

PL/SQL permite una longitud máxima de 32.767 para los parámetros RAW y VARCHAR2. Asegúrese de que los recursos asignados del sistema son adecuados para soportar los tamaños de buffer para un número determinado de sesiones de usuario. De lo contrario, el servidor de Oracle emitirá las excepciones de memoria correspondientes.

Nota: BLOB y BFILE devuelven RAW. Los demás devuelven VARCHAR2.

DBMS_LOB.WRITE

Llame al procedimiento WRITE para escribir en piecewise una cantidad (AMOUNT) especificada de datos en un LOB determinado, desde el BUFFER especificado por el usuario, comenzando por un OFFSET absoluto desde el principio del valor LOB.

Asegúrese (especialmente con caracteres multibyte) de que la cantidad de bytes se corresponde con la cantidad de datos del buffer. WRITE no puede comprobar si coinciden y escribirá los bytes AMOUNT del contenido del buffer en el LOB.

Inicialización de Columnas LOB Agregadas a una Tabla

- Crear la tabla con columnas mediante el tipo LOB o agregar las columnas LOB con ALTER TABLE.

```
ALTER TABLE employees  
ADD (resume CLOB, picture BLOB);
```

- Inicializar el valor del localizador LOB de la columna con la opción DEFAULT o las sentencias DML mediante:
 - La función EMPTY_CLOB() para una columna CLOB
 - La función EMPTY_BLOB() para una columna BLOB

```
CREATE TABLE emp_hiredata (  
    employee_id NUMBER(6),  
    full_name VARCHAR2(45),  
    resume CLOB DEFAULT EMPTY_CLOB(),  
    picture BLOB DEFAULT EMPTY_BLOB());
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Inicialización de Columnas LOB Agregadas a una Tabla

Las columnas LOB se definen mediante el lenguaje de definición de datos (DDL) de SQL, como en la sentencia ALTER TABLE de la transparencia. El contenido de una columna LOB se almacena en el segmento LOB, mientras que la columna de la tabla sólo contiene una referencia a esta área concreta de almacenamiento, que se denomina localizador LOB. En PL/SQL, puede definir una variable del tipo LOB, que sólo contiene el valor del localizador LOB. Puede inicializar los localizadores LOB mediante:

- La función EMPTY_CLOB() en un localizador LOB para una columna CLOB
- La función EMPTY_BLOB() en un localizador LOB para una columna BLOB

Nota: Estas funciones crean el valor del localizador LOB y no el contenido LOB. Por norma general, utilice las subrutinas del paquete DBMS_LOB para llenar el contenido. Las funciones están disponibles en DML de Oracle SQL y no forman parte del paquete DBMS_LOB.

El último ejemplo de la transparencia muestra la forma de utilizar las funciones EMPTY_CLOB() y EMPTY_BLOB() en la opción DEFAULT de una sentencia CREATE TABLE. De esta forma, cuando una fila se inserte en la tabla y las columnas LOB no se hayan especificado en la sentencia INSERT, los valores del localizador LOB se llenarán en sus columnas correspondientes.

La siguiente página muestra cómo utilizar las funciones en las sentencias INSERT y UPDATE para inicializar los valores del localizador LOB.

Relleno de Columnas LOB

- **Insertar una fila en una tabla con columnas LOB:**

```
INSERT INTO emp_hiredata
  (employee_id, full_name, resume, picture)
VALUES (405, 'Marvin Ellis', EMPTY_CLOB(), NULL);
```

- **Iniciar un LOB con la función EMPTY_BLOB():**

```
UPDATE emp_hiredata
  SET resume = 'Date of Birth: 8 February 1951',
      picture = EMPTY_BLOB()
 WHERE employee_id = 405;
```

- **Actualizar una columna CLOB:**

```
UPDATE emp_hiredata
  SET resume = 'Date of Birth: 1 June 1956'
 WHERE employee_id = 170;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Relleno de Columnas LOB

Puede insertar un valor directamente en una columna LOB con las variables de host en SQL o en PL/SQL, SQL embebido por 3GL u OCI. Puede utilizar las funciones especiales EMPTY_BLOB () y EMPTY_CLOB () en las sentencias INSERT o UPDATE de DML de SQL para inicializar un LOB interno NULL o no NULL que se vaya a vaciar. Para llenar una columna LOB, realice los siguientes pasos:

1. Inicialice la columna LOB en un valor no NULL es decir, defina un localizador LOB que apunte a un valor LOB vacío o llenado. Para ello, utilice las funciones EMPTY_BLOB () y EMPTY_CLOB ().
2. Rellene el contenido LOB con las rutinas del paquete DBMS_LOB.

Sin embargo, como se muestra en los ejemplos de la transparencia, las dos sentencias UPDATE inicializan el valor del localizador resume LOB y rellenan el contenido proporcionando un valor literal. También se puede realizar en una sentencia INSERT. Una columna LOB se puede actualizar a:

- Otro valor LOB
- Un valor NULL
- Un localizador LOB con contenido vacío mediante la función incorporada EMPTY_*LOB ()

Puede actualizar el LOB con una variable ligada en el SQL embebido. Al asignar un LOB a otro, se crea una nueva copia del valor LOB. Utilice una sentencia SELECT FOR UPDATE para bloquear la fila que contiene la columna LOB antes de actualizar una parte del contenido LOB.

Actualización de LOB con DBMS_LOB en PL/SQL

```
DECLARE
    lobloc CLOB;          -- serves as the LOB locator
    text  VARCHAR2(50) := 'Resigned = 5 June 2000';
    amount NUMBER ;        -- amount to be written
    offset INTEGER;        -- where to start writing
BEGIN
    SELECT resume INTO lobloc FROM emp_hiredata
    WHERE employee_id = 405 FOR UPDATE;
    offset := DBMS_LOB.GETLENGTH(lobloc) + 2;
    amount := length(text);
    DBMS_LOB.WRITE (lobloc, amount, offset, text);
    text := ' Resigned = 30 September 2000';
    SELECT resume INTO lobloc FROM emp_hiredata
    WHERE employee_id = 170 FOR UPDATE;
    amount := length(text);
    DBMS_LOB.WRITEAPPEND(lobloc, amount, text);
    COMMIT;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Actualización de LOB con DBMS_LOB en PL/SQL

En el ejemplo de la transparencia, la variable LOBLOC sirve como localizador LOB y la variable AMOUNT se define en la longitud del texto que desea agregar. La sentencia SELECT FOR UPDATE bloquea la fila y devuelve el localizador LOB para la columna RESUME LOB. Por último, se llama al procedimiento del paquete WRITE PL/SQL para escribir el texto en el valor LOB del offset especificado. WRITEAPPEND se agrega al valor LOB existente.

El ejemplo muestra cómo recuperar una columna CLOB en versiones anteriores a Oracle9i. En estas versiones, no se ha podido recuperar una columna CLOB directamente en una columna con caracteres. Ha sido necesario enlazar el valor de la columna a un localizador LOB, al que se accede mediante el paquete DBMS_LOB. Más adelante en esta misma lección, un ejemplo muestra cómo puede recuperar directamente una columna CLOB enlazándola con una variable de caracteres.

Nota: Las versiones anteriores a Oracle9i no han aceptado los LOB en la cláusula WHERE de las sentencias UPDATE y SELECT. Ahora se aceptan las funciones SQL de los LOB en los predicados de WHERE. Más adelante en esta misma lección se muestra un ejemplo.

Selección de Valores CLOB con SQL

```
SELECT employee_id, full_name , resume -- CLOB  
FROM emp_hiredata  
WHERE employee_id IN (405, 170);
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Selección de Valores CLOB con SQL

Es posible ver los datos en una columna CLOB con una sentencia SELECT. No es posible ver los datos en una columna BLOB o BFILE con una sentencia SELECT en iSQL*Plus. Debe utilizar una herramienta que muestre información binaria para un BLOB, así como el software relevante para un BFILE. Por ejemplo, puede utilizar Oracle Forms.

Selección de Valores CLOB con DBMS_LOB

- **DBMS_LOB.SUBSTR (lob, amount, start_pos)**
- **DBMS_LOB.INSTR (lob, pattern)**

```
SELECT DBMS_LOB.SUBSTR (resume, 5, 18),
       DBMS_LOB.INSTR (resume, ' = ')
  FROM emp_hiredata
 WHERE employee_id IN (170, 405);
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Selección de Valores CLOB con DBMS_LOB

DBMS_LOB.SUBSTR

Utilice DBMS_LOB.SUBSTR para mostrar parte de un LOB. Es similar en funcionalidad a la función SQL SUBSTR.

DBMS_LOB.INSTR

Utilice DBMS_LOB.INSTR para buscar información en el LOB. Esta función devuelve la posición numérica de la información.

Nota: Desde Oracle9i, también puede utilizar las funciones SQL SUBSTR y INSTR para realizar las operaciones que se muestran en la transparencia.

Selección de Valores CLOB en PL/SQL

```
SET LINESIZE 50 SERVEROUTPUT ON FORMAT WORD_WRAP
DECLARE
    text VARCHAR2(4001);
BEGIN
    SELECT resume INTO text
    FROM emp_hiredata
    WHERE employee_id = 170;
    DBMS_OUTPUT.PUT_LINE('text is: '|| text);
END;
/
```

```
text is: Date of Birth: 1 June 1956 Resigned = 30
September 2000
```

```
PL/SQL procedure successfully completed.
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Selección de Valores CLOB en PL/SQL

La transparencia muestra el código para acceder a los valores CLOB que se pueden convertir implícitamente a VARCHAR2 en Oracle10g. Cuando se seleccione, el valor de la columna RESUME se convertirá implícitamente de un CLOB a un VARCHAR2 para almacenarlo en la variable TEXT. Antes de Oracle9i, primero se recuperaba el valor del localizador CLOB en una variable CLOB y, a continuación, se leía el contenido LOB especificando la cantidad y offset en el procedimiento DBMS_LOB.READ:

```
DECLARE
    rlob    CLOB;
    text   VARCHAR2(4001);
    amt    NUMBER := 4001;
    offset NUMBER := 1;
BEGIN
    SELECT resume INTO rlob FROM emp_hiredata
    WHERE employee_id = 170;
    DBMS_LOB.READ(rlob, amt, offset, text);
    DBMS_OUTPUT.PUT_LINE('text is: '|| text);
END;
/
text is: Date of Birth: 1 June 1956 Resigned = 30 September 2000
PL/SQL procedure successfully completed.
```

Eliminación de LOB

- Suprimir una fila que contenga LOB:

```
DELETE  
FROM emp_hiredata  
WHERE employee_id = 405;
```

- Disocie un valor LOB de una fila:

```
UPDATE emp_hiredata  
SET resume = EMPTY_CLOB()  
WHERE employee_id = 170;
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Eliminación de LOB

Una instancia LOB se puede suprimir (destruir) con las sentencias DML de SQL adecuadas. La sentencia SQL DELETE suprime una fila y su valor interno LOB asociado. Para mantener la fila y destruir sólo la referencia al LOB, debe actualizar la fila sustituyendo el valor de la columna LOB por NULL o por una cadena vacía o utilizando la función EMPTY_B/CLOB().

Nota: Sustituir el valor de una columna por NULL no es lo mismo que utilizar EMPTY_B/CLOB. El uso de NULL define el valor en nulo. El uso de EMPTY_B/CLOB garantiza que no hay nada en la columna.

Un LOB se destruye cuando se suprime la fila que contiene la columna LOB, cuando se borra o se trunca la tabla o cuando se actualizan todos los datos LOB.

Debe eliminar explícitamente el archivo asociado a un BFILE con los comandos del sistema operativo.

Para borrar parte de un LOB interno, puede utilizar DBMS_LOB.ERASE.

LOB Temporales

- **LOB temporales:**
 - Proporcionan una interfaz que soporta la creación de LOB que actúan como variables locales
 - Pueden ser BLOB, CLOB o NCLOB
 - No están asociados a una tabla concreta
 - Se crean con el procedimiento DBMS_LOB.CREATETEMPORARY
 - Utilizan las rutinas DBMS_LOB
- La duración de un LOB temporal es una sesión.
- Los LOB temporales son útiles para transformar datos en LOB internos permanentes.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

LOB Temporales

Los LOB temporales proporcionan una interfaz para soportar la creación y supresión de LOB que actúan como variables locales. Los LOB temporales pueden ser BLOB, CLOB o NCLOB.

A continuación se indican las funciones de los LOB temporales:

- Los datos se almacenan en el tablespace temporal y no en tablas.
- Los LOB temporales son más rápidos que los persistentes porque no generan ninguna información de rollback o redo.
- La consulta de los LOB temporales se localiza en la propia sesión de cada usuario. Al LOB temporal sólo puede acceder el usuario que lo cree y todos los LOB temporales se suprimirán al final de la sesión en la que han sido creados.
- Puede crear un LOB temporal con DBMS_LOB.CREATETEMPORARY.

Los LOB temporales resultan útiles cuando deseé realizar alguna operación transformacional en un LOB (por ejemplo, cambiar un tipo de imagen de GIF a JPEG). Un LOB temporal estará vacío cuando una vez creado no soporte las funciones EMPTY_B/CLOB.

Utilice el paquete DBMS_LOB para usar y manipular los LOB temporales.

Creación de un LOB Temporal

Procedimiento PL/SQL para crear y comprobar un LOB temporal:

```
CREATE OR REPLACE PROCEDURE is_templob_open(
    lob IN OUT BLOB, retval OUT INTEGER) IS
BEGIN
    -- create a temporary LOB
    DBMS_LOB.CREATETEMPORARY (lob, TRUE);
    -- see if the LOB is open: returns 1 if open
    retval := DBMS_LOB.ISOPEN (lob);
    DBMS_OUTPUT.PUT_LINE (
        'The file returned a value...' || retval);
    -- free the temporary LOB
    DBMS_LOB.FREETEMPORARY (lob);
END;
/
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un LOB Temporal

El ejemplo de la transparencia muestra un procedimiento PL/SQL definido por el usuario, `is_templob_open`, que crea un LOB temporal. Este procedimiento acepta un localizador LOB como entrada, crea un LOB temporal, lo abre y comprueba si el LOB está abierto.

El procedimiento `IsTempLOBOpen` utiliza los procedimientos y funciones del paquete `DBMS_LOB` como se indica a continuación:

- El procedimiento `CREATETEMPORARY` se utiliza para crear el LOB temporal.
- La función `ISOPEN` se utiliza para comprobar si el LOB está abierto: si el LOB está abierto, esta función devuelve el valor 1.
- El procedimiento `FREETEMPORARY` se utiliza para liberar el LOB temporal. La memoria aumenta de forma incremental a medida que crece el número de LOB temporales y puede volver a utilizar el espacio del LOB temporal en la sesión liberando explícitamente LOB temporales.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- **Identificar cuatro tipos integrados para objetos grandes: BLOB, CLOB, NCLOB y BFILE**
- **Describir la forma en la que los LOB sustituyen a LONG y LONG RAW**
- **Describir dos opciones de almacenamiento para los LOB:**
 - **El servidor de Oracle (LOB internos)**
 - **Archivos de host externos (LOB externos)**
- **Utilizar el paquete PL/SQL DBMS_LOB para proporcionar rutinas para la gestión de LOB**
- **Utilizar LOB temporales en una sesión**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen

Hay cuatro tipos de dato LOB:

- Un BLOB es un objeto grande binario.
- Un CLOB es un objeto grande de caracteres.
- Un NCLOB almacena datos del juego de caracteres nacional multibyte.
- Un BFILE es un objeto grande almacenado en un archivo binario fuera de la base de datos.

Los LOB se pueden almacenar internamente (en la base de datos) o externamente (en un archivo del sistema operativo). Puede gestionar LOB con el paquete DBMS_LOB y su procedimiento.

Los LOB temporales proporcionan una interfaz para soportar la creación y supresión de LOB que actúan como variables locales.

Práctica 9: Visión General

En esta práctica se abordan los siguientes temas:

- **Creación de tipos de objetos con los tipos de dato CLOB y BLOB**
- **Creación de una tabla con tipos de dato LOB como columnas**
- **Uso del paquete DBMS_LOB para rellenar e interactuar con los datos LOB**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica 9: Visión General

En esta práctica, cree una tabla con las columnas BLOB y CLOB. A continuación, utilice el paquete DBMS_LOB para llenar la tabla y manipular los datos.

Práctica 9

1. Cree una tabla denominada PERSONNEL ejecutando el archivo de comandos E:\labs\PLPU\labs\lab_09_01.sql. La tabla contiene los siguientes atributos y tipos de dato:

Nombre de la Columna	Tipo de Dato	Longitud
ID	NUMBER	6
last_name	VARCHAR2	35
review	CLOB	N/A
picture	BLOB	N/A

2. Inserte dos filas en la tabla PERSONNEL, una por cada empleado 2034 (cuyo apellido es Allen) y para el empleado 2035 (cuyo apellido es Bond). Utilice la función vacía para CLOB y proporcione el valor NULL para BLOB.
3. Examine y ejecute el archivo de comandos E:\labs\PLPU\labs\lab_09_03.sql. El archivo de comandos crea una tabla denominada REVIEW_TABLE. Esta tabla contiene información de la revisión anual de cada empleado. El archivo de comandos también tiene dos sentencias para insertar datos de la revisión de dos empleados.
4. Actualice la tabla PERSONNEL.
 - a. Rellene la primera fila del CLOB utilizando esta subconsulta en una sentencia UPDATE:

```
SELECT ann_review
  FROM review_table
 WHERE employee_id = 2034;
```
 - b. Rellene la segunda fila de CLOB , utilizando PL/SQL y el paquete DBMS_LOB. Utilice la siguiente sentencia SELECT para proporcionar un valor para el localizador LOB.

```
SELECT ann_review
  FROM review_table
 WHERE employee_id = 2035;
```

Si tiene tiempo, realice el siguiente ejercicio:

5. Cree un procedimiento que agregue un localizador a un archivo binario en la columna PICTURE de la tabla COUNTRIES. El archivo binario es una imagen de una bandera. Los archivos de imagen se nombran según los identificadores de los países. Tiene que cargar un localizador de archivo de imagen en todas las filas de la región de Europa (REGION_ID = 1) en la tabla COUNTRIES. Debe crear un objeto de DIRECTORY denominado COUNTRY_PIC que hace referencia a la ubicación de los archivos binarios.
 - a. Agregue la columna de imagen a la tabla COUNTRIES utilizando:

```
ALTER TABLE countries ADD (picture BFILE);
```

También puede utilizar el archivo E:\labs\PLPU\labs\Lab_09_05_a.sql.
 - b. Cree un procedimiento PL/SQL denominado load_country_image que utilice DBMS_LOB.FILEEXISTS para comprobar si el archivo de la imagen del país existe. Si es así, defina el localizador BFILE para el archivo en la columna PICTURE; de lo contrario, aparece el mensaje de que el archivo no existe. Utilice el paquete DBMS_OUTPUT para proporcionar la información del tamaño del archivo para cada imagen asociada a la columna PICTURE.
 - c. Llame al procedimiento transfiriendo el nombre del objeto de directorio COUNTRY_PIC como valor de parámetro de literal de cadena.

Creación de Disparadores

10

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- **Describir los diferentes tipos de disparadores**
- **Describir los disparadores de base de datos y sus usos**
- **Crear disparadores de base de datos**
- **Describir las reglas de arranque de disparadores de base de datos**
- **Eliminar disparadores de base de datos**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetivos

En esta lección aprenderá a crear y utilizar disparadores de base de datos.

Tipos de Disparadores

Un disparador:

- **Es un bloque PL/SQL o un procedimiento PL/SQL asociado a una tabla, vista, esquema o base de datos**
- **Se ejecuta implícitamente siempre que se produce un evento concreto**
- **Puede ser de uno de los tipos siguientes:**
 - **Disparador de aplicación: Arranca siempre que se produce un evento con una aplicación concreta**
 - **Disparador de base de datos: Arranca siempre que se produce un evento de datos (como DML) o un evento del sistema (como una conexión o un cierre) en un esquema o base de datos**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Tipos de Disparadores

Los disparadores de aplicación se ejecutan implícitamente siempre que se produce un evento concreto de lenguaje de manipulación de datos (DML) en una aplicación. Un ejemplo de aplicación que utiliza mucho los disparadores es una desarrollada con Oracle Forms Developer.

Los disparadores de base de datos se ejecutan implícitamente cuando se produce cualquiera de los siguientes eventos:

- Operaciones DML en una tabla
- Operaciones DML en una vista, con un disparador INSTEAD OF
- Sentencias DDL, como CREATE y ALTER

Para ello, no importa el usuario que esté conectado ni la aplicación que se utilice. Los disparadores de base de datos también se ejecutan implícitamente cuando se producen algunas acciones del usuario o del sistema de base de datos (por ejemplo, cuando un usuario se conecta o el DBA cierra la base de datos).

Nota: Los disparadores de base de datos se pueden definir en tablas y en vistas. Si se emite una operación DML en una vista, el disparador INSTEAD OF define las acciones que tienen lugar. Si estas acciones incluyen operaciones DML en tablas, se arrancan los disparadores de las tablas base.

Tipos de Disparadores (continuación)

Los disparadores de base de datos pueden ser disparadores del sistema en una base de datos o en un esquema. Para las bases de datos, los disparadores se arrancan para cada evento para todos los usuarios; para un esquema, se arrancan para cada evento para un usuario concreto.

En este curso se explica cómo crear disparadores de base de datos. La creación de disparadores de base de datos basados en eventos del sistema se trata en la lección titulada “Más Conceptos de Disparadores”.

Oracle Internal & OAI Use Only

Instrucciones para el Diseño de Disparadores

- **Puede diseñar disparadores para lo siguiente:**
 - Realizar acciones relacionadas
 - Centralizar operaciones globales
- **No debe diseñar disparadores:**
 - Cuando la funcionalidad ya esté incorporada en el servidor de Oracle
 - Que dupliquen otros disparadores
- **Puede crear procedimientos almacenados y llamarlos en un disparador, si el código PL/SQL es muy largo.**
- **El uso excesivo de disparadores puede dar lugar a interdependencias complejas, que pueden resultar difíciles de mantener en grandes aplicaciones.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Instrucciones para el Diseño de Disparadores

- Utilice disparadores para garantizar que se realizan acciones relacionadas con una operación concreta.
- Utilice disparadores de base de datos para operaciones globales centralizadas que se deben arrancar para la sentencia disparadora, independientemente del usuario o la aplicación que emite dicha sentencia.
- No defina disparadores para duplicar o sustituir la funcionalidad ya incorporada en la base de datos Oracle. Por ejemplo, implemente reglas de integridad mediante restricciones declarativas y no con disparadores. Para recordar el orden de diseño de una regla de negocio:
 - Utilice restricciones incorporadas en el servidor de Oracle, como una clave primaria, etc.
 - Desarrolle un disparador de base de datos o una aplicación, como un servlet o Enterprise JavaBean (EJB) en la capa media.
 - Utilice una interfaz de presentación, como Oracle Forms, HTML, JavaServer Pages (JSP), etc., para las reglas de presentación de datos.
- El uso excesivo de disparadores puede dar lugar a interdependencias complejas, que pueden resultar difíciles de mantener. Utilice disparadores cuando sea necesario y tenga en cuenta los efectos recursivos y en cascada.

Instrucciones para el Diseño de Disparadores (continuación)

- Evite lógicas largas de disparadores mediante la creación de procedimientos almacenados o procedimientos empaquetados que se llaman en el cuerpo del disparador.
- Los disparadores de base de datos se arrancan para todos los usuarios cada vez que se produce el evento en el disparador creado.

Oracle Internal & OAI Use Only

Creación de Disparadores DML

Crear disparadores de tipo fila o sentencia DML mediante:

```
CREATE [OR REPLACE] TRIGGER trigger_name
  timing
  event1 [OR event2 OR event3]
  ON object_name
  [[REFERENCING OLD AS old / NEW AS new]
  FOR EACH ROW
  [WHEN (condition)]]
  trigger_body
```

- **Un disparador de sentencia se arranca una vez para una sentencia DML.**
- **Un disparador de fila se arranca una vez para cada fila afectada.**

Nota: Los nombres de los disparadores deben ser únicos con respecto a otros disparadores del mismo esquema.

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de Disparadores DML

Los componentes de la sintaxis del disparador son:

- *trigger_name* identifica el disparador como único.
- *timing* indica cuando se arranca el disparador en relación con el evento disparador. Los valores son BEFORE, AFTER o INSTEAD OF.
- *event* identifica la operación DML que provoca que se arranque el disparador. Los valores son INSERT, UPDATE [OF *column*] y DELETE.
- *object_name* indica la tabla o vista asociada al disparador.
- Para los disparadores de fila, puede especificar:
 - Una cláusula REFERENCING para seleccionar nombres de correlación para hacer referencia a los valores anterior y nuevo de la fila actual (los valores por defecto son OLD y NEW)
 - FOR EACH ROW para designar que se trata de un disparador de fila
 - Una cláusula WHEN para aplicar un predicado condicional, entre paréntesis, que se evalúe para cada fila para determinar si se debe ejecutar el cuerpo del disparador
- *trigger_body* es la acción realizada por el disparador, implementada como uno de los tipos siguientes:
 - Un bloque anónimo con una cláusula DECLARE o BEGIN y END
 - Una cláusula CALL para llamar a un procedimiento almacenado empaquetado o autónomo como, por ejemplo:
CALL my_procedure;

Tipos de Disparadores DML

El tipo de disparador determina si el cuerpo se ejecuta para cada fila o sólo una vez para la sentencia disparadora.

- **Un disparador de sentencia:**
 - Se ejecuta una vez para el evento disparador
 - Es el tipo por defecto de disparador
 - Se arranca una vez incluso aunque no haya filas afectadas en absoluto
- **Un disparador de fila:**
 - Se ejecuta una vez para cada fila afectada por el evento disparador
 - No se ejecuta si el evento disparador no afecta a ninguna fila
 - Se indica mediante la especificación de la cláusula **FOR EACH ROW**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Tipos de Disparadores DML

Puede especificar que el disparador se ejecute una vez para cada fila afectada por la sentencia disparadora (como UPDATE de varias filas) o una vez para la sentencia disparadora, independientemente del número de filas a las que afecte.

Disparador de Sentencia

Un disparador de sentencia se arranca una vez en nombre del evento disparador, incluso aunque no haya filas afectadas en absoluto. Los disparadores de sentencia son útiles si la acción del disparador no depende de los datos de las filas afectadas o de los datos proporcionados por el evento disparador en sí (por ejemplo, un disparador que realiza una comprobación de seguridad compleja en el usuario actual).

Disparador de Fila

Un disparador de fila se arranca cada vez que la tabla se ve afectada por el evento disparador. Si el evento disparador no afecta a ninguna fila, no se ejecuta ningún disparador de fila. Los disparadores de fila son útiles si la acción del disparador depende de los datos de las filas afectadas o de los datos proporcionados por el evento disparador en sí.

Nota: Los disparadores de fila utilizan nombres de correlación para acceder a los valores de columna anterior y nuevo de la fila procesada por el disparador.

Temporización de Disparadores

¿Cuándo se debe arrancar el disparador?

- **BEFORE:** Ejecutar el cuerpo del disparador antes del evento DML disparador en una tabla.
- **AFTER:** Ejecutar el cuerpo del disparador después del evento DML disparador en una tabla.
- **INSTEAD OF:** Ejecutar el cuerpo del disparador en lugar de la sentencia disparadora. Se utiliza para vistas que no se pueden modificar de otra forma.

Nota: Si se definen varios disparadores para el mismo objeto, el orden de arranque de los disparadores es arbitrario.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Temporización de Disparadores

La temporización del disparador **BEFORE** se utiliza con frecuencia en las siguientes situaciones:

- Para determinar si se debe permitir que termine la sentencia disparadora (de esta forma, se elimina el procesamiento innecesario y se puede realizar un rollback en casos en los que se produzca una excepción en la acción disparadora)
- Para derivar valores de columna antes de terminar una sentencia INSERT o UPDATE
- Para inicializar indicadores o variables globales y para validar reglas de negocio complejas

Los disparadores **AFTER** se utilizan con frecuencia en las siguientes situaciones:

- Para terminar la sentencia disparadora antes de ejecutar la acción disparadora
- Para realizar distintas acciones en la misma sentencia disparadora si ya hay un disparador BEFORE

Los disparadores **INSTEAD OF** proporcionan una forma transparente de modificar vistas que no se pueden modificar directamente mediante sentencias DML de SQL ya que las vistas no siempre son modificables. Puede escribir las sentencias DML adecuadas dentro del cuerpo de un disparador **INSTEAD OF** para realizar acciones directamente en las tablas subyacentes de las vistas.

Nota: Si se definen varios disparadores para una tabla, el orden en el que se arrancan varios disparadores del mismo tipo es arbitrario. Para asegurarse de que los disparadores del mismo tipo se arrancan en un orden concreto, consolide dichos disparadores en uno que llame a procedimientos independientes en el orden deseado.

Secuencia de Arranque de Disparadores

Utilizar la siguiente secuencia de arranque para un disparador en una tabla cuando se manipule una única fila:

Sentencia DML

```
INSERT INTO departments
(department_id, department_name, location_id)
VALUES (400, 'CONSULTING', 2400);
```

Acción disparadora

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
30	Purchasing	1700
...		
400	CONSULTING	2400

→ **Disparador de sentencia BEFORE**

→ **Disparador de fila BEFORE**

→ **Disparador de fila AFTER**

→ **Disparador de sentencia AFTER**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Secuencia de Arranque de Disparadores

Cree un disparador de sentencia o de fila basado en el requisito de que el disparador se debe arrancar una vez para cada fila afectada por la sentencia disparadora, o bien sólo una vez para la sentencia disparadora, independientemente del número de filas afectadas.

Cuando la sentencia DML disparadora afecta a una única fila, tanto el disparador de sentencia como el de fila se arrancan exactamente una vez.

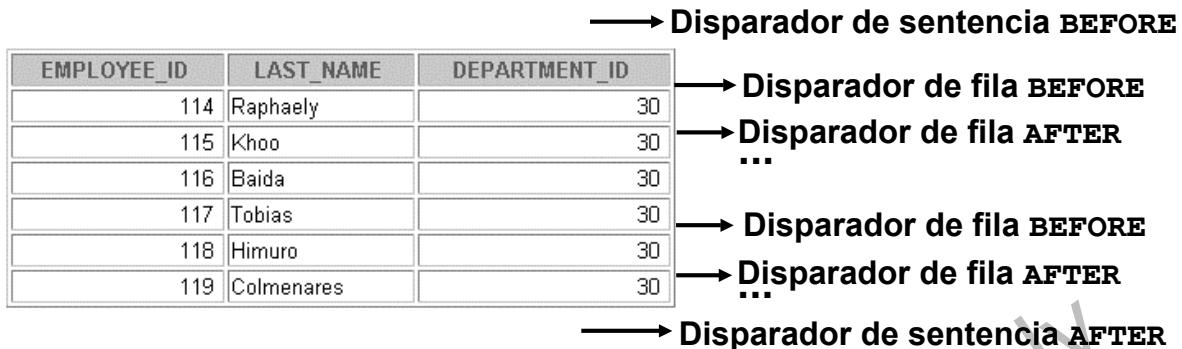
Ejemplo

La sentencia SQL de la transparencia no diferencia los disparadores de sentencia de los de fila, porque se inserta exactamente una fila en la tabla mediante la sintaxis de la sentencia `INSERT` mostrada.

Secuencia de Arranque de Disparadores

Utilizar la siguiente secuencia de arranque para un disparador en una tabla cuando se manipulen varias filas:

```
UPDATE employees  
SET salary = salary * 1.1  
WHERE department_id = 30;
```



Copyright © 2004, Oracle. Todos los Derechos Reservados.

ORACLE

Secuencia de Arranque de Disparadores (continuación)

Cuando la sentencia DML disparadora afecta a varias filas, el disparador de sentencia se arranca exactamente una vez y el disparador de fila se arranca una vez para cada fila afectada por la sentencia.

Ejemplo

La sentencia SQL de la transparencia hace que un disparador de nivel de fila se arranque un número de veces igual al número de filas que cumplen la cláusula WHERE, es decir, el número de empleados que pertenezcan al departamento 30.

Tipos de Evento y Cuerpo del Disparador

Un evento del disparador:

- Determina qué sentencia DML hace que se ejecute el disparador
- Los tipos son:
 - **INSERT**
 - **UPDATE [OF column]**
 - **DELETE**

El cuerpo del disparador:

- Determina qué acción se realiza
- Es un bloque PL/SQL o una llamada mediante **CALL** a un procedimiento

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Tipos de Evento Disparador

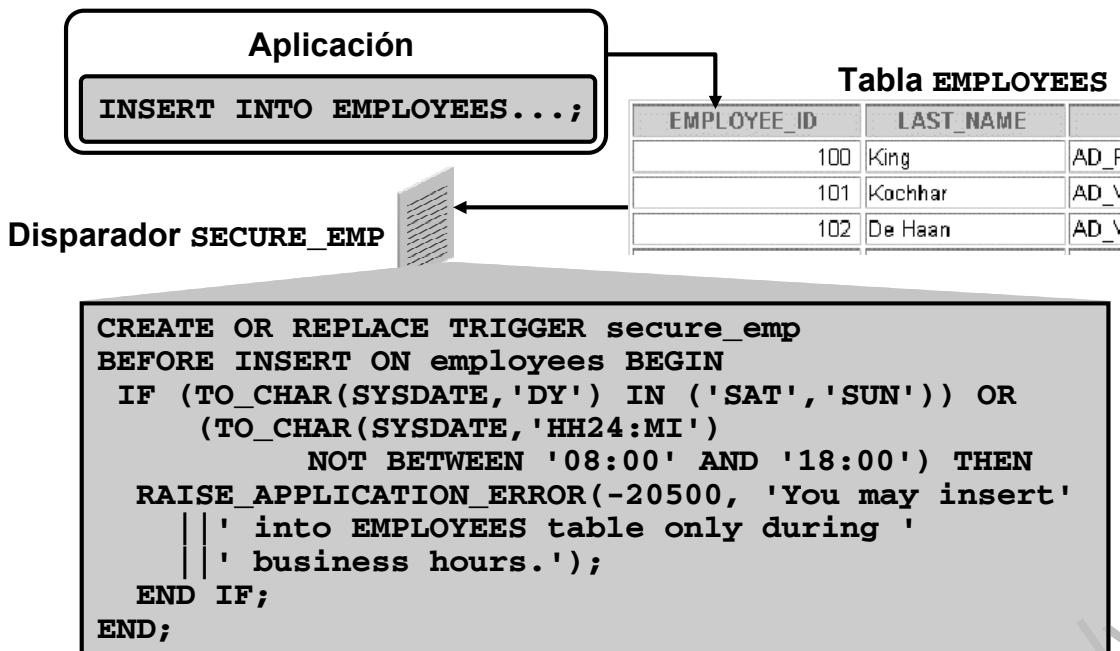
La sentencia o evento disparador puede ser una sentencia **INSERT**, **UPDATE** o **DELETE** en una tabla.

- Cuando el evento disparador es una sentencia **UPDATE**, puede incluir una lista de columnas para identificar las columnas que se deben cambiar para arrancar el disparador. No puede especificar una lista de columnas para una sentencia **INSERT** o **DELETE**, porque siempre afectan a todas las filas.
 . . . UPDATE OF salary . . .
- El evento disparador puede contener una, dos o tres de estas operaciones DML.
 . . . INSERT or UPDATE or DELETE
 . . . INSERT or UPDATE OF job_id . . .

El cuerpo del disparador define la acción, es decir, lo que se tiene que hacer cuando se emite el evento disparador. El bloque PL/SQL puede incluir sentencias SQL y PL/SQL y puede definir construcciones PL/SQL como, por ejemplo, variables, cursor, excepciones, etc. También puede llamar a un procedimiento PL/SQL o Java.

Nota: El tamaño de un disparador no puede ser superior a 32 KB.

Creación de un Disparador de Sentencia DML



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un Disparador de Sentencia DML

En este ejemplo, el disparador de base de datos SECURE_EMP es un disparador de sentencia BEFORE que impide que se realice correctamente la operación INSERT si se viola la condición de negocio. En este caso, el disparador restringe las inserciones en la tabla EMPLOYEES durante determinadas horas laborables, de lunes a viernes.

Si un usuario intenta insertar una fila en la tabla EMPLOYEES el sábado, verá un mensaje de error, el disparador fallará y se realizará un rollback de la sentencia disparadora. Recuerde que RAISE_APPLICATION_ERROR es un procedimiento incorporado del servidor que devuelve un error al usuario y hace que el bloque PL/SQL falle.

Cuando falla un disparador de base de datos, el servidor de Oracle realiza automáticamente un rollback de la sentencia disparadora.

Prueba de SECURE_EMP

```
INSERT INTO employees (employee_id, last_name,
                      first_name, email, hire_date,
                      job_id, salary, department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,
        'IT_PROG', 4500, 60);
```

```
INSERT INTO employees (employee_id, last_name, first_name, email,
                      *)
```

ERROR at line 1:

ORA-20500: You may insert into EMPLOYEES table only during business hours.

ORA-06512: at "PLSQL.SECURE_EMP", line 4

ORA-04088: error during execution of trigger 'PLSQL.SECURE_EMP'

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Prueba de SECURE_EMP

Inserte una fila en la tabla EMPLOYEES durante horas no laborables. Cuando la fecha y la hora están fuera del horario laboral especificado en el disparador, obtiene un mensaje de error como el que se muestra en la transparencia.

Uso de Predicados Condicionales

```
CREATE OR REPLACE TRIGGER secure_emp BEFORE
INSERT OR UPDATE OR DELETE ON employees BEGIN
  IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
    (TO_CHAR(SYSDATE,'HH24') NOT BETWEEN '08' AND '18') THEN
    IF [DELETING] THEN RAISE_APPLICATION_ERROR(
      -20502,'You may delete from EMPLOYEES table' ||
      'only during business hours.');
    ELSIF [INSERTING] THEN RAISE_APPLICATION_ERROR(
      -20500,'You may insert into EMPLOYEES table' ||
      'only during business hours.');
    ELSIF [UPDATING('SALARY')] THEN
      RAISE_APPLICATION_ERROR(-20503, 'You may' ||
      'update SALARY only during business hours.');
    ELSE RAISE_APPLICATION_ERROR(-20504,'You may' ||
      ' update EMPLOYEES table only during' ||
      ' normal hours.');
    END IF;
  END IF;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Combinación de Eventos Disparadores

Puede combinar varios eventos disparadores en uno aprovechando los predicados condicionales especiales INSERTING, UPDATING y DELETING en el cuerpo del disparador.

Ejemplo

Cree un disparador para restringir todos los eventos de manipulación de datos en la tabla EMPLOYEES a determinadas horas laborables, de lunes a viernes.

Creación de un Disparador de Fila DML

```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
    IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
        AND :NEW.salary > 15000 THEN
        RAISE_APPLICATION_ERROR (-20202,
            'Employee cannot earn more than $15,000.');
    END IF;
END;
/
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un Disparador de Fila DML

Puede crear un disparador de fila BEFORE con el fin de impedir que la operación disparadora se realice correctamente si se viola una determinada condición.

En el ejemplo, se crea un disparador para permitir que determinados empleados puedan ganar un salario superior a 15.000. Si un usuario intenta ejecutar la siguiente sentencia UPDATE:

```
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell';
```

El disparador produce la siguiente excepción:

```
UPDATE EMPLOYEES
*
ERROR at line 1:
ORA-20202: Employee cannot earn more than $15,000.
ORA-06512: at "PLSQL.RESTRICT_SALARY", line 5
ORA-04088: error during execution of trigger
"PLSQL.RESTRICT_SALARY"
```

Uso de los Cualificadores OLD y NEW

```
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO audit_emp(user_name, time_stamp, id,
        old_last_name, new_last_name, old_title,
        new_title, old_salary, new_salary)
    VALUES (USER, SYSDATE, :OLD.employee_id,
        :OLD.last_name, :NEW.last_name, :OLD.job_id,
        :NEW.job_id, :OLD.salary, :NEW.salary);
END;
/
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de los Cualificadores OLD y NEW

En un disparador ROW, haga referencia al valor de una columna antes y después del cambio de los datos agregando los cualificadores OLD y NEW como prefijo.

Operación de Datos	Valor Anterior	Valor Nuevo
INSERT	NULL	Valor insertado
UPDATE	Valor anterior a la actualización	Valor posterior a la actualización
DELETE	Valor anterior a la supresión	NULL

Notas de Uso:

- Los cualificadores OLD y NEW sólo están disponibles en los disparadores ROW.
- Agregue a estos cualificadores dos puntos (:) como prefijo en cada sentencia SQL y PL/SQL.
- No hay ningún prefijo de dos puntos (:) si se hace referencia a los cualificadores en la condición de restricción WHEN.

Nota: Los disparadores de fila pueden disminuir el rendimiento si realiza muchas actualizaciones en tablas más grandes.

Uso de los Cualificadores OLD y NEW: Ejemplo con audit_emp

```
INSERT INTO employees
  (employee_id, last_name, job_id, salary, ...)
VALUES (999, 'Temp emp', 'SA_REP', 6000,...);

UPDATE employees
  SET salary = 7000, last_name = 'Smith'
 WHERE employee_id = 999;
```

```
SELECT user_name, timestamp, ...
  FROM audit_emp;
```

USER_NAME	TIMESTAMP	ID	OLD_LAST_N	NEW_LAST_N	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
PLSQL	28-SEP-01			Temp emp		SA REP		1000
PLSQL	28-SEP-01	999	Temp emp	Smith	SA REP	SA REP	1000	2000

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de los Cualificadores OLD y NEW: Ejemplo con la Tabla AUDIT_EMP

Cree un disparador en la tabla EMPLOYEES para agregar filas a una tabla de usuario, AUDIT_EMP, que registra la actividad de un usuario en relación con la tabla EMPLOYEES. El disparador registra los valores de varias columnas antes y después de los cambios en los datos mediante los cualificadores OLD y NEW con el nombre de columna correspondiente. Existe una columna adicional denominada COMMENTS en AUDIT_EMP que no se muestra en la transparencia.

Restricción de un Disparador de Fila: Ejemplo

```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
BEGIN
  IF INSERTING THEN
    :NEW.commission_pct := 0;
  ELSIF :OLD.commission_pct IS NULL THEN
    :NEW.commission_pct := 0;
  ELSE
    :NEW.commission_pct := :OLD.commission_pct+0.05;
  END IF;
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Restricción de un Disparador de Fila: Ejemplo

Para restringir la acción del disparador a aquellas filas que cumplan una determinada condición, proporcione una cláusula WHEN.

Cree un disparador en la tabla EMPLOYEES para calcular la comisión de un empleado cuando se agrega una fila a la tabla EMPLOYEES, o bien cuando se modifica el salario de un empleado.

No se puede agregar dos puntos como prefijo al cualificador NEW en la cláusula WHEN, porque la cláusula WHEN está fuera de los bloques PL/SQL.

Resumen del Modelo de Ejecución de Disparadores

- 1. Ejecutar todos los disparadores BEFORE STATEMENT.**
- 2. Bucle para cada fila afectada:**
 - a. Ejecutar todos los disparadores BEFORE ROW.**
 - b. Ejecutar la sentencia DML y realizar la comprobación de restricción de integridad.**
 - c. Ejecutar todos los disparadores AFTER ROW.**
- 3. Ejecutar todos los disparadores AFTER STATEMENT.**

Nota: La comprobación de integridad se puede diferir hasta que se realice la operación COMMIT.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Modelo de Ejecución de Disparadores

Una única sentencia DML podría arrancar hasta cuatro tipos de disparadores:

- Los disparadores de sentencia BEFORE y AFTER
- Los disparadores de fila BEFORE y AFTER

Un evento disparador o una sentencia dentro del disparador puede hacer que se compruebe una o más restricciones de integridad. Sin embargo, puede diferir la comprobación de integridad hasta que se realice una operación COMMIT.

Los disparadores también pueden hacer que se arranquen otros disparadores (denominados disparadores en cascada).

Todas las acciones y comprobaciones realizadas como resultado de una sentencia SQL se deben ejecutar correctamente. Si se produce una excepción en un disparador y ésta no se maneja explícitamente, se realiza un rollback de todas las acciones realizadas debido a la sentencia SQL original (incluidas las acciones realizadas arrancando los disparadores). De esta forma, se garantiza que las restricciones de integridad no se vean nunca comprometidas por los disparadores.

Cuando se arranca un disparador, las tablas a las que se hace referencia en la acción del disparador pueden sufrir cambios debido a transacciones de otros usuarios. En todos los casos, se garantiza una imagen de lectura consistente para los valores modificados que el disparador tiene que leer (consultar) o escribir (actualizar).

Implementación de una Restricción de Integridad con un Disparador

```
UPDATE employees SET department_id = 999  
WHERE employee_id = 170;  
-- Integrity constraint violation error
```

```
CREATE OR REPLACE TRIGGER employee_dept_fk_trg  
AFTER UPDATE OF department_id  
ON employees FOR EACH ROW  
BEGIN  
    INSERT INTO departments VALUES (:new.department_id,  
        'Dept ' || :new.department_id, NULL, NULL);  
EXCEPTION  
    WHEN DUP_VAL_ON_INDEX THEN  
        NULL; -- mask exception if department exists  
END;  
/
```

```
UPDATE employees SET department_id = 999  
WHERE employee_id = 170;  
-- Successful after trigger is fired
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Implementación de una Restricción de Integridad con un Disparador

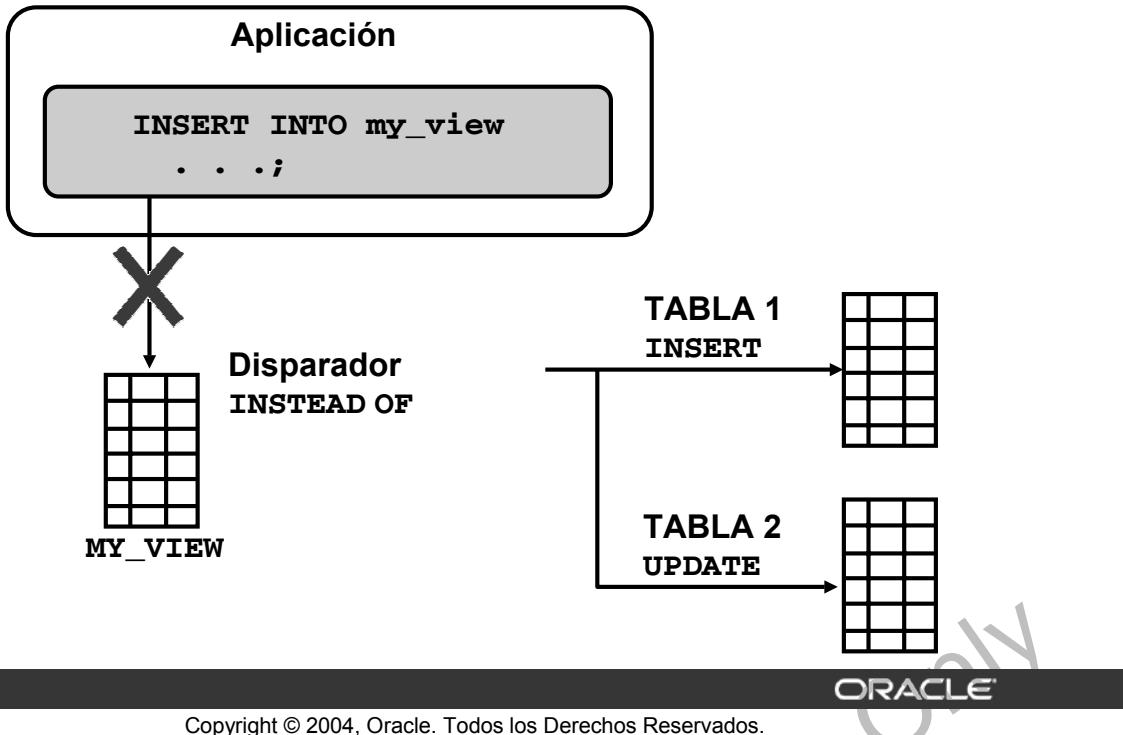
En el ejemplo de la transparencia se explica una situación en la que se puede ocupar de la restricción de integridad mediante un disparador. La tabla EMPLOYEES tiene una restricción de clave ajena en la columna DEPARTMENT_ID de la tabla DEPARTMENTS.

En la primera sentencia SQL, se modifica el valor de DEPARTMENT_ID del empleado 170 con 999. Puesto que el departamento 999 no existe en la tabla DEPARTMENTS, la sentencia produce la excepción -2292 para la violación de restricción de integridad.

Se crea el disparador EMPLOYEE_DEPT_FK_TRG que inserta una nueva fila en la tabla DEPARTMENTS, mediante :NEW.DEPARTMENT_ID para el valor de DEPARTMENT_ID del nuevo departamento. El disparador se arranca cuando la sentencia UPDATE modifica el valor de DEPARTMENT_ID del empleado 170 con 999. Cuando se comprueba la restricción de clave ajena, se realiza correctamente, ya que el disparador insertó el departamento 999 en la tabla DEPARTMENTS. Por lo tanto, no se produce ninguna excepción, a menos que el departamento ya exista cuando el disparador intenta insertar la nueva fila. Sin embargo, el manejador EXCEPTION detecta y oculta la excepción, lo que permite que la operación se realice correctamente.

Nota: Este ejemplo funciona con Oracle8i y versiones posteriores, pero produce un error de tiempo de ejecución en versiones anteriores a Oracle8i.

Disparadores INSTEAD OF



Copyright © 2004, Oracle. Todos los Derechos Reservados.

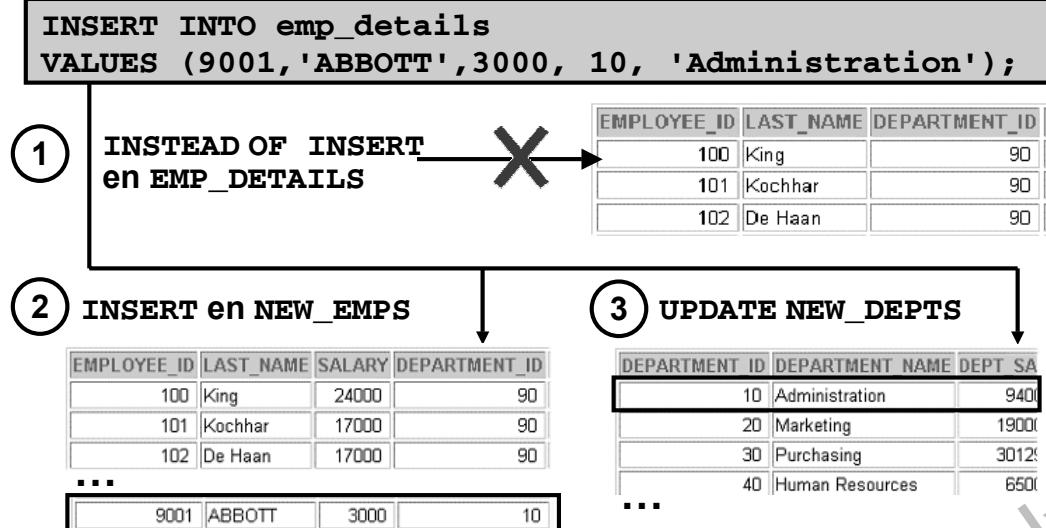
Disparadores INSTEAD OF

Utilice los disparadores INSTEAD OF para modificar datos en los que se ha emitido la sentencia DML en relación con una vista no actualizable intrínsecamente. Estos disparadores se denominan INSTEAD OF porque, a diferencia de otros disparadores, el servidor de Oracle arranca el disparador en lugar de ejecutar la sentencia disparadora. Este disparador se utiliza para realizar una operación INSERT, UPDATE o DELETE directamente en las tablas subyacentes. Puede escribir sentencias INSERT, UPDATE o DELETE en relación con una vista y el disparador INSTEAD OF funciona sin ser visto en segundo plano para que se realicen las acciones correctas. Una vista no se puede modificar con sentencias DML normales si la consulta de la vista contiene operadores de definición, funciones de grupo, cláusulas como GROUP BY, CONNECT BY, START, el operador DISTINCT o uniones. Por ejemplo, si una vista se compone de más de una tabla, una inserción en la vista puede implicar una inserción en una tabla y una actualización en otra. Por lo tanto, debe escribir un disparador INSTEAD OF que se arranque cuando escriba una inserción con respecto a la vista. En lugar de la inserción original, se ejecuta el cuerpo del disparador, cuyo resultado es una inserción de datos en una tabla y una actualización en otra.

Nota: Si una vista es actualizable intrínsecamente y tiene disparadores INSTEAD OF, éstos tienen prioridad. Los disparadores INSTEAD OF son disparadores de fila. La opción CHECK para vistas no se aplica cuando se realizan inserciones o actualizaciones en la vista mediante los disparadores INSTEAD OF. El cuerpo del disparador INSTEAD OF debe forzar la comprobación.

Creación de un Disparador INSTEAD OF

Realizar la operación INSERT en EMP_DETAILS basada en las tablas EMPLOYEES y DEPARTMENTS:



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un Disparador INSTEAD OF

Puede crear un disparador INSTEAD OF con el fin de mantener las tablas base en las que se basa la vista. El ejemplo ilustra la inserción de un empleado en la vista **EMP_DETAILS**, cuya consulta se basa en las tablas **EMPLOYEES** y **DEPARTMENTS**. El disparador **NEW_EMP_DEPT** (INSTEAD OF) se ejecuta en lugar de la operación **INSERT** que hace que se arranque el disparador. A continuación, el disparador INSTEAD OF emite las operaciones **INSERT** y **UPDATE** adecuadas a las tablas base utilizadas por la vista **EMP_DETAILS**. Por lo tanto, en lugar de insertar el registro del nuevo empleado en la tabla **EMPLOYEES**, se realizan las siguientes acciones:

1. Se arranca el disparador **NEW_EMP_DEPT** INSTEAD OF.
2. Se inserta una fila en la tabla **NEW_EMPS**.
3. Se actualiza la columna **DEPT_SAL** de la tabla **NEW_DEPTS**. El valor de salario proporcionado para el nuevo empleado se agrega al salario total existente del departamento al que se ha asignado el nuevo empleado.

Nota: El código de este supuesto se muestra en las páginas siguientes.

Creación de un Disparador INSTEAD OF

Utilizar INSTEAD OF para realizar DML en vistas complejas:

```
CREATE TABLE new_emps AS
  SELECT employee_id, last_name, salary, department_id
    FROM employees;

CREATE TABLE new_depts AS
  SELECT d.department_id, d.department_name,
         sum(e.salary) dept_sal
    FROM employees e, departments d
   WHERE e.department_id = d.department_id;

CREATE VIEW emp_details AS
  SELECT e.employee_id, e.last_name, e.salary,
         e.department_id, d.department_name
    FROM   employees e, departments d
   WHERE e.department_id = d.department_id
  GROUP BY d.department_id, d.department_name;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de un Disparador INSTEAD OF (continuación)

El ejemplo crea dos nuevas tablas, NEW_EMPS y NEW_DEPTS, basadas en las tablas EMPLOYEES y DEPARTMENTS, respectivamente. También crea una vista EMP_DETAILS de las tablas EMPLOYEES y DEPARTMENTS.

Si una vista tiene una estructura de consulta compleja, no siempre es posible ejecutar DML directamente en la vista para que afecte a las tablas subyacentes. El ejemplo necesita la creación de un disparador INSTEAD OF, denominado NEW_EMP_DEPT, que se muestra en la página siguiente. El disparador NEW_DEPT_EMP maneja DML de la siguiente forma:

- Cuando se inserta una fila en la vista EMP_DETAILS, en lugar de insertar la fila directamente en la vista, se agregan filas a las tablas NEW_EMPS y NEW_DEPTS, con los valores de datos proporcionados con la sentencia INSERT.
- Cuando se modifica o suprime una fila en la vista EMP_DETAILS, las filas correspondientes de las tablas NEW_EMPS y NEW_DEPTS se ven afectadas.

Nota: Los disparadores INSTEAD OF sólo se pueden escribir para vistas y las opciones de temporización BEFORE y AFTER no son válidas.

Creación de un Disparador **INSTEAD OF** (continuación)

```
CREATE OR REPLACE TRIGGER new_emp_dept
INSTEAD OF INSERT OR UPDATE OR DELETE ON emp_details
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO new_emps
        VALUES (:NEW.employee_id, :NEW.last_name,
                :NEW.salary, :NEW.department_id);
        UPDATE new_depts
        SET dept_sal = dept_sal + :NEW.salary
        WHERE department_id = :NEW.department_id;
    ELSIF DELETING THEN
        DELETE FROM new_emps
        WHERE employee_id = :OLD.employee_id;
        UPDATE new_depts
        SET dept_sal = dept_sal - :OLD.salary
        WHERE department_id=:OLD.department_id;
    ELSIF UPDATING ('salary') THEN
        UPDATE new_emps
        SET salary = :NEW.salary
        WHERE employee_id = :OLD.employee_id;
        UPDATE new_depts
        SET dept_sal = dept_sal +
                      (:NEW.salary - :OLD.salary)
        WHERE department_id=:OLD.department_id;
    ELSIF UPDATING ('department_id') THEN
        UPDATE new_emps
        SET department_id=:NEW.department_id
        WHERE employee_id = :OLD.employee_id;
        UPDATE new_depts
        SET dept_sal = dept_sal - :OLD.salary
        WHERE department_id=:OLD.department_id;
        UPDATE new_depts
        SET dept_sal = dept_sal + :NEW.salary
        WHERE department_id = :NEW.department_id;
    END IF;
END;
/
```

Comparación de Disparadores de Base de Datos y Procedimientos Almacenados

Disparadores	Procedimientos
Definidos con CREATE TRIGGER. El diccionario de datos contiene código de origen en USER_TRIGGERS . Se llaman implícitamente mediante DML. No se permiten COMMIT , SAVEPOINT y ROLLBACK .	Definidos con CREATE PROCEDURE. El diccionario de datos contiene código de origen en USER_SOURCE . Se llaman explícitamente. Se permiten COMMIT , SAVEPOINT y ROLLBACK .

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Comparación de Disparadores de Base de Datos y Procedimientos Almacenados

Existen diferencias entre los disparadores de base de datos y los procedimientos almacenados:

Disparador de Base de Datos	Procedimiento Almacenado
Se llama implícitamente.	Se llama explícitamente.

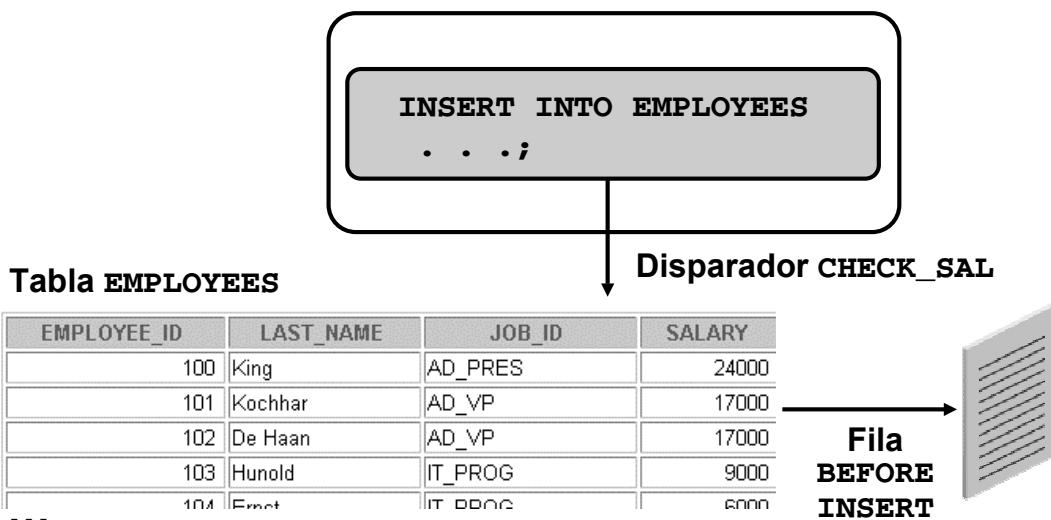
No se permiten las sentencias COMMIT, ROLLBACK y SAVEPOINT en el cuerpo del disparador. Se puede confirmar o realizar un rollback indirectamente llamando a un procedimiento, pero no se recomienda debido a los efectos secundarios en las transacciones.

Se permiten las sentencias COMMIT, ROLLBACK y SAVEPOINT en el cuerpo del procedimiento.

Los disparadores se compilan completamente cuando se emite el comando **CREATE TRIGGER** y se almacena el código ejecutable en el diccionario de datos.

Nota: Si se producen errores durante la compilación de un disparador, éste se crea de todos modos.

Comparación de Disparadores de Base de Datos y Disparadores de Oracle Forms



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Comparación de Disparadores de Base de Datos y Disparadores de Oracle Forms

Los disparadores de base de datos son distintos de los de Forms Builder.

Disparador de Base de Datos	Disparador de Forms Builder
Se ejecuta mediante acciones de cualquier herramienta o aplicación de base de datos.	Se ejecuta sólo en una aplicación concreta de Forms Builder.
Siempre se dispara mediante SQL DML, DDL o una determinada acción de base de datos.	Se pueden disparar pasando de un campo a otro, pulsando una tecla o mediante otras muchas acciones.
Se distingue entre disparador de sentencia y de fila.	Se distingue entre disparador de sentencia y de fila.
Si se produce un fallo, se realiza un rollback de la sentencia disparadora.	Si se produce un fallo, el cursor se congela y puede que se realice un rollback de toda la transacción.
Se arranca independientemente y además de los disparadores de Forms Builder.	Se arranca independientemente y además de los disparadores de base de datos.
Se ejecuta en el dominio de seguridad del autor del disparador.	Se ejecuta en el dominio de seguridad del usuario de Forms Builder.

Gestión de Disparadores

- Desactivar o volver a activar un disparador de base de datos:

```
ALTER TRIGGER trigger_name DISABLE | ENABLE
```

- Desactivar o volver a activar todos los disparadores para una tabla:

```
ALTER TABLE table_name DISABLE | ENABLE  
ALL TRIGGERS
```

- Recompilar un disparador para una tabla:

```
ALTER TRIGGER trigger_name COMPILE
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Gestión de Disparadores

Un disparador tiene dos modos o estados: ENABLED y DISABLED. Cuando se crea por primera vez un disparador, se activa por defecto. El servidor de Oracle comprueba las restricciones de integridad de los disparadores activados y garantiza que éstos no pueden comprometerlas. Además, el servidor de Oracle proporciona vistas de lectura consistente para consultas y restricciones, gestiona las dependencias y proporciona un proceso de confirmación en dos fases si un disparador actualiza tablas remotas de una base de datos distribuida.

Desactivación de un Disparador

- Mediante la sintaxis ALTER TRIGGER o puede desactivar todos los disparadores en una tabla mediante la sintaxis ALTER TABLE.
- Para mejorar el rendimiento o evitar comprobaciones de integridad de datos al cargar grandes cantidades de datos con utilidades como, por ejemplo, SQL*Loader. Puede desactivar un disparador cuando hace referencia a un objeto de base de datos que en ese momento no esté disponible, debido a un fallo de la conexión de red, un fallo del disco, un archivo de datos offline o un tablespace offline.

Recompilación de un Disparador

- Mediante el comando ALTER TRIGGER para recompilar explícitamente un disparador que no es válido.
- Mediante la emisión de una sentencia ALTER TRIGGER con la opción COMPILE, independientemente de si es válido o no.

Eliminación de Disparadores

Para eliminar un disparador de la base de datos, utilice la sentencia **DROP TRIGGER**:

```
DROP TRIGGER trigger_name;
```

Ejemplo:

```
DROP TRIGGER secure_emp;
```

Nota: Todos los disparadores de una tabla se eliminan cuando ésta se elimina.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Eliminación de Disparadores

Cuando ya no se necesita un disparador, utilice una sentencia SQL en iSQL*Plus para eliminarlo.

Prueba de Disparadores

- **Probar cada una de las operaciones de datos disparadoras, así como las no disparadoras.**
- **Probar cada caso de la cláusula WHEN.**
- **Hacer que el disparador se arranque directamente a partir de una operación de datos básica, así como indirectamente a partir de un procedimiento.**
- **Probar el efecto del disparador en otros disparadores.**
- **Probar el efecto de otros disparadores en el disparador.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Prueba de Disparadores

El proceso de prueba del código puede tardar bastante tiempo. Sin embargo, al probar los disparadores:

- Asegúrese de que el disparador funciona correctamente probando diferentes casos de forma independiente:
 - Pruebe primero los supuestos correctos más comunes.
 - Pruebe las condiciones de fallo más comunes para ver si se gestionan de forma adecuada.
- Cuanto más complejo sea el disparador, más detallada será la prueba. Por ejemplo, si tiene un disparador de fila con una cláusula WHEN especificada, deberá asegurarse de que el disparador se arranca cuando se cumplen las condiciones. O bien, si tiene disparadores en cascada, tendrá que probar el efecto de un disparador en los otros y asegurarse de que termina con los resultados deseados.
- Utilice el paquete DBMS_OUTPUT para depurar los disparadores.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- **Crear disparadores de base de datos que se llaman mediante operaciones DML**
- **Crear disparadores del tipo sentencia y fila**
- **Utilizar las reglas de arranque de disparadores de base de datos**
- **Activar, desactivar y gestionar disparadores de base de datos**
- **Desarrollar una estrategia de prueba de disparadores**
- **Eliminar disparadores de base de datos**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen

En esta lección se ha abordado la creación de disparadores de base de datos que se ejecutan antes, después o en lugar de una operación DML especificada. Los disparadores se asocian a las tablas o vistas de base de datos. Las temporizaciones BEFORE y AFTER se aplican a las operaciones DML en las tablas. El disparador INSTEAD OF se utiliza como una forma de sustituir operaciones DML en una vista por sentencias DML adecuadas en relación con otras tablas de la base de datos.

Los disparadores se activan por defecto, pero se pueden desactivar para inhibir su operación hasta que se vuelva a activar. Si las reglas de negocio cambian, los disparadores se pueden eliminar o modificar según sea necesario.

Práctica 10: Visión General

En esta práctica se abordan los siguientes temas:

- Creación de disparadores de fila
- Creación de un disparador de sentencia
- Llamada de procedimientos desde un disparador

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica 10: Visión General

En esta práctica, creará disparadores de sentencia y de fila. Creará procedimientos que se llamarán desde los disparadores.

Oracle Internal & OA Use Only

Práctica 10

1. Las filas de la tabla JOBS almacenan los salarios mínimo y máximo permitidos para los distintos valores de JOB_ID. Le piden que escriba un código para garantizar que el salario de los empleados está dentro del rango permitido por su tipo de trabajo, para operaciones de inserción y actualización.
 - a. Escriba un procedimiento denominado CHECK_SALARY que acepte dos parámetros, uno para la cadena del identificador de trabajo del empleado y el otro para el salario. El procedimiento utiliza el identificador de trabajo para determinar el salario mínimo y máximo para el trabajo especificado. Si el parámetro del salario, mínimo y máximo incluidos, no está dentro del rango de salarios aparecerá una excepción de aplicación con el mensaje “Invalid salary <sal>. Salaries for job <jobid> must be between <min> and <max>”. Sustituya los distintos elementos del mensaje por los valores que proporcionan los parámetros y las variables rellenados con consultas.
 - b. Cree un disparador denominado CHECK_SALARY_TRG en la tabla EMPLOYEES que arranque ante una operación INSERT o UPDATE en cada fila. El disparador debe llamar al procedimiento CHECK_SALARY para ejecutar la lógica de negocio. El disparador transferirá el nuevo identificador de trabajo y salario a los parámetros de procedimiento.
2. Pruebe CHECK_SAL_TRG utilizando los siguientes casos:
 - a. Utilice el procedimiento EMP_PKG .ADD_EMPLOYEE para agregar a la empleada Eleanor Beh al departamento 30. ¿Qué sucede? ¿Por qué?
 - b. Actualice el salario del empleado 115 a 2.000 dólares. En otra operación de actualización, cambie el identificador de trabajo del empleado a HR_REP. ¿Qué sucede en cada caso?
 - c. Actualice el salario del empleado 115 a 2.800 dólares. ¿Qué sucede?
3. Actualice el disparador CHECK_SALARY_TRG para que arranque sólo cuando los valores del identificador de trabajo o el salario hayan cambiado en realidad.
 - a. Implemente la regla de negocio utilizando una cláusula WHEN para comprobar si los valores JOB_ID o SALARY han cambiado.

Nota: Asegúrese de que la condición maneja NULL en los valores de OLD.column_name si se realiza una operación INSERT; si no es así, la operación de inserción fallará.
 - b. Compruebe el disparador ejecutando el procedimiento EMP_PKG .ADD_EMPLOYEE con los siguientes valores de parámetros: first_name='Eleanor', last_name='Beh', email='EBEH', job='IT_PROG', sal=5000.
 - c. Actualice a los empleados con un trabajo IT_PROG incrementando su salario en 2.000 dólares. ¿Qué sucede?
 - d. Actualice a 9.000 dólares el salario de Eleanor Beh.

Indicación: Utilice una sentencia UPDATE con una subconsulta en la cláusula WHERE. ¿Qué sucede?

 - e. Cambie el trabajo de Eleanor Beh a ST_MAN utilizando otra sentencia UPDATE con una subconsulta. ¿Qué sucede?

Práctica 10 (continuación)

4. Se le pide que evite que se suprima a los empleados durante las horas laborables.
 - a. Escriba un disparador de sentencia denominado DELETE_EMP_TRG en la tabla EMPLOYEES para evitar que las filas se supriman durante horas laborables entre semana, es decir, de las 9:00 a.m. a las 6:00 p.m.
 - b. Intente suprimir los empleados con JOB_ID de SA_REP que no estén asignados a un departamento.

Indicación: Empleado Grant con identificador 178.

Oracle Internal & OAI Use Only

Aplicaciones para Disparadores

11

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- **Crear disparadores de base de datos adicionales**
- **Explicar las reglas que rigen a los disparadores**
- **Implementar disparadores**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetivos

En esta lección, aprenderá a crear más disparadores de base de datos y conocerá las reglas que rigen los disparadores. Asimismo, conocerá las diversas aplicaciones de los disparadores.

Creación de Disparadores de Base de Datos

- **Disparo de un evento de usuario:**
 - CREATE, ALTER O DROP
 - Conexión o desconexión
- **Disparo de un evento de base de datos o sistema:**
 - Cierre o inicio de la base de datos
 - Un error específico (o cualquier error) que se produzca

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creating Database Triggers

Before coding the trigger body, decide on the components of the trigger.

Triggers on system events can be defined at the database or schema level. For example, a database shutdown trigger is defined at the database level. Triggers on data definition language (DDL) statements, or a user logging on or off, can also be defined at either the database level or schema level. Triggers on DML statements are defined on a specific table or a view.

A trigger defined at the database level fires for all users, and a trigger defined at the schema or table level fires only when the triggering event involves that schema or table.

Triggering events that can cause a trigger to fire:

- A data definition statement on an object in the database or schema
- A specific user (or any user) logging on or off
- A database shutdown or startup
- Any error that occurs

Creación de Disparadores en Sentencias DDL

Sintaxis:

```
CREATE [OR REPLACE] TRIGGER trigger_name
Timing
[ddl_event1 [OR ddl_event2 OR ...]]
ON {DATABASE | SCHEMA}
trigger_body
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de Disparadores en Sentencias DDL

DDL_Event	Valores Posibles
CREATE	Hace que el servidor de Oracle arranque el disparador cuando una sentencia CREATE agregue un nuevo objeto de base de datos al diccionario
ALTER	Hace que el servidor de Oracle arranque el disparador cuando una sentencia ALTER modifique un objeto de base de datos en el diccionario de datos
DROP	Hace que el servidor de Oracle arranque el disparador cuando una sentencia DROP elimine un objeto de base de datos en el diccionario de datos

El cuerpo del disparador representa un bloque PL/SQL completo.

Puede crear disparadores para estos eventos en DATABASE o SCHEMA. Especifique también BEFORE o AFTER para la temporización del disparador.

Los disparadores DDL sólo se arrancan si el objeto que se crea es cluster, función, índice, paquete, procedimiento, rol, secuencia, sinónimo, tabla, tablespace, disparador, tipo, vista o usuario.

Creación de Disparadores en Eventos de Sistema

Sintaxis:

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
[database_event1 [OR database_event2 OR ...]]
ON {DATABASE|SCHEMA}
trigger_body
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Creación de la Sintaxis del Disparador

Database_event	Valores Posibles
AFTER SERVERERROR	Hace que el servidor de Oracle arranque el disparador cuando se registre un mensaje de error del servidor
AFTER LOGON	Hace que el servidor de Oracle arranque el disparador cuando un usuario se conecte a la base de datos
BEFORE LOGOFF	Hace que el servidor de Oracle arranque el disparador cuando un usuario se desconecte de la base de datos
AFTER STARTUP	Hace que el servidor de Oracle arranque el disparador cuando se abra la base de datos
BEFORE SHUTDOWN	Hace que el servidor de Oracle arranque el disparador cuando se cierre la base de datos

Puede crear disparadores para estos eventos en DATABASE o SCHEMA, excepto SHUTDOWN y STARTUP, que sólo se aplican a DATABASE.

Disparadores LOGON y LOGOFF: Ejemplo

```
CREATE OR REPLACE TRIGGER logon_trig
AFTER LOGON ON SCHEMA
BEGIN
  INSERT INTO log_trig_table(user_id,log_date,action)
  VALUES (USER, SYSDATE, 'Logging on');
END;
/
```

```
CREATE OR REPLACE TRIGGER logoff_trig
BEFORE LOGOFF ON SCHEMA
BEGIN
  INSERT INTO log_trig_table(user_id,log_date,action)
  VALUES (USER, SYSDATE, 'Logging off');
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Disparadores LOGON y LOGOFF: Ejemplo

Puede crear estos disparadores para controlar la frecuencia de conexión y desconexión o puede que desee escribir un informe que controle el tiempo durante el cual está conectado. Al especificar ON SCHEMA, el disparador se arranca para el usuario especificado. Si especifica ON DATABASE, el disparador se arranca para todos los usuarios.

Sentencias CALL

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
event1 [OR event2 OR event3]
ON table_name
[REFERENCING OLD AS old | NEW AS new]
[ FOR EACH ROW ]
[WHEN condition]
CALL procedure_name
/
```

```
CREATE OR REPLACE TRIGGER log_employee
BEFORE INSERT ON EMPLOYEES
CALL log_execution
/
```

Nota: No hay punto y coma al final de la sentencia CALL.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Sentencias CALL

Una sentencia CALL le permite llamar a un procedimiento almacenado, en lugar de codificar el cuerpo PL/SQL en el mismo disparador. El procedimiento se puede implementar en PL/SQL, en C o en Java.

La llamada puede hacer referencia a los atributos del disparador :NEW y :OLD como parámetros, como se muestra en el siguiente ejemplo:

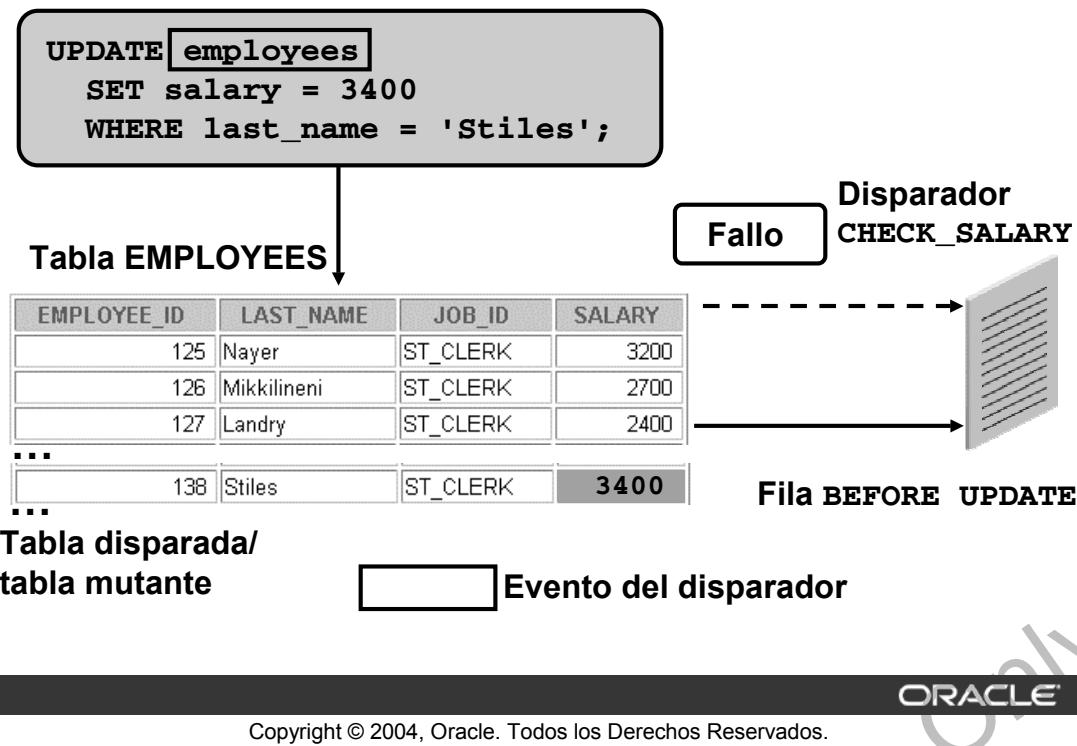
```
CREATE TRIGGER salary_check
BEFORE UPDATE OF salary, job_id ON employees
FOR EACH ROW
WHEN (NEW.job_id <> 'AD_PRES')
CALL check_salary(:NEW.job_id, :NEW.salary)
/
```

Nota: No hay punto y coma al final de la sentencia CALL.

En el ejemplo anterior, el disparador llama a un procedimiento *check_salary*.

El procedimiento compara el nuevo salario con el rango de salarios para el nuevo identificador de trabajo desde la tabla JOBS.

Lectura de Datos en una Tabla Mutante



ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Reglas que Rigen los Disparadores

Las acciones de leer y escribir datos con los disparadores están sujetas a determinadas reglas.

Las restricciones sólo se aplican a los disparadores de fila, a menos que se arranque un disparador de sentencia como resultado de ON DELETE CASCADE.

Tabla Mutante

Se trata de una tabla que está siendo modificada actualmente por una sentencia UPDATE, DELETE o INSERT o de una tabla que puede que necesite ser actualizada por los efectos de una acción de integridad referencial declarativa DELETE CASCADE. Para los disparadores STATEMENT, una tabla no se considera una tabla mutante.

La tabla disparada es una tabla mutante, así como cualquier tabla a la que se haga referencia con la restricción FOREIGN KEY. Esta restricción evita que el disparador de fila vea un juego de datos inconsistente.

Tabla Mutante: Ejemplo

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE INSERT OR UPDATE OF salary, job_id
  ON employees
  FOR EACH ROW
  WHEN (NEW.job_id <> 'AD_PRES')
DECLARE
  minsalary employees.salary%TYPE;
  maxsalary employees.salary%TYPE;
BEGIN
  SELECT MIN(salary), MAX(salary)
    INTO minsalary, maxsalary
   FROM employees
  WHERE job_id = :NEW.job_id;
  IF :NEW.salary < minsalary OR
     :NEW.salary > maxsalary THEN
    RAISE_APPLICATION_ERROR(-20505, 'Out of range');
  END IF;
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Tabla Mutante: Ejemplo

El disparador CHECK_SALARY del ejemplo trata de garantizar que, siempre que un nuevo empleado se agregue a la tabla EMPLOYEES o siempre que el identificador de trabajo o el salario de un empleado existente se modifique, el salario del empleado esté dentro del rango de salarios establecido para el trabajo del empleado.

Cuando se actualiza el registro de un empleado, el disparador CHECK_SALARY se arranca para cada fila actualizada. El código del disparador consulta la misma tabla que está siendo actualizada. Por lo tanto, se considera que la tabla EMPLOYEES es una tabla mutante.

Tabla Mutante: Ejemplo

```
UPDATE employees  
SET salary = 3400  
WHERE last_name = 'Stiles';
```

```
UPDATE employees  
*  
ERROR at line 1:  
ORA-04091: table PLSQL.EMPLOYEES is mutating, trigger/function may not see it  
ORA-06512: at "PLSQL.CHECK_SALARY", line 5  
ORA-04088: error during execution of trigger 'PLSQL.CHECK_SALARY'
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Tabla Mutante: Ejemplo (continuación)

En el ejemplo, el código del disparador trata de leer o seleccionar los datos en una tabla mutante.

Si restringe el salario que hay dentro del rango entre el valor mínimo y el valor máximo existente, obtendrá un error de tiempo de ejecución. La tabla EMPLOYEES es mutante o está en un estado de cambio. Por lo tanto, el disparador no puede leer en ella.

Recuerde que las funciones también pueden causar un error en la tabla mutante al ser llamadas en una sentencia DML.

Posibles Soluciones

A continuación, se indican las posibles soluciones a este problema de la tabla mutante:

- Almacenar los datos de resumen (los salarios mínimos y los máximos) en otra tabla de resumen, que se mantiene actualizada con otros disparadores DML.
- Almacenar los datos de resumen en un paquete PL/SQL y acceder a los datos desde el paquete. Se puede realizar en un disparador de sentencia BEFORE.

Según la naturaleza del problema, éste puede ser más enrevesado y difícil de resolver.

En este caso, considere la implementación de las reglas en la aplicación o capa media y evite el uso de los disparadores de base de datos para ejecutar reglas de negocio demasiado complejas.

Ventajas de los Disparadores de Base de Datos

- **Seguridad de datos mejorada:**
 - Proporciona comprobaciones de seguridad complejas y mejoradas
 - Proporciona auditorías complejas y mejoradas
- **Integridad de datos mejorada:**
 - Fuerza las restricciones dinámicas de integridad de los datos
 - Fuerza las restricciones complejas de integridad referencial
 - Garantiza que las operaciones relacionadas se realizan juntas de forma implícita

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ventajas de los Disparadores de Base de Datos

Puede utilizar disparadores de base de datos:

- Como alternativa a las funciones que proporciona el servidor de Oracle
- En el caso de que sus requisitos sean más complejos o más simples que los que proporciona el servidor de Oracle
- En el caso de que el servidor de Oracle no proporcione sus requisitos

Gestión de Disparadores

Los siguientes privilegios del sistema son necesarios para gestionar los disparadores:

- **Privilegio CREATE/ALTER/DROP (ANY) TRIGGER:** le permite crear un disparador en cualquier esquema
- **Privilegio ADMINISTER DATABASE TRIGGER:** le permite crear un disparador en DATABASE
- **Privilegio EXECUTE (si el disparador hace referencia a los objetos que no están en el esquema)**

Nota: Las sentencias del cuerpo del disparador utilizan los privilegios del propietario del disparador, no los privilegios del usuario que ejecuta la operación que arranca el disparador.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Gestión de Disparadores

Para crear un disparador en el esquema, necesita el privilegio del sistema CREATE TRIGGER y debe ser el propietario de la tabla especificada en la sentencia disparadora, tener el privilegio ALTER para la tabla en la sentencia disparadora o tener el privilegio del sistema ALTER ANY TABLE. Puede modificar o borrar los disparadores sin que sean necesarios otros privilegios.

Si se utiliza la palabra clave ANY, podrá crear, modificar o borrar sus propios disparadores y los de otro esquema y se podrán asociar a la tabla de cualquier usuario.

No es necesario que ningún privilegio llame a un disparador en el esquema. Las sentencias DML que emite son las que llaman a un disparador. Pero si el disparador hace referencia a objetos que no están en el esquema, el usuario que crea el disparador deberá tener el privilegio EXECUTE en los paquetes, funciones o procedimientos a los que se hace referencia y no mediante roles.

Al igual que con los procedimientos almacenados, las sentencias del cuerpo del disparador utilizan los privilegios del propietario del disparador, no los privilegios del usuario que ejecuta la operación que arranca el disparador.

Para crear un disparador en DATABASE, debe tener el privilegio ADMINISTER DATABASE TRIGGER. Si este privilegio se revoca más tarde, podrá borrar el disparador pero no podrá modificarlo.

Supuestos de Aplicación de Negocio para la Implementación de Disparadores

Puede utilizar los disparadores para:

- **Seguridad**
- **Auditoría**
- **Integridad de los datos**
- **Integridad referencial**
- **Replicación de tablas**
- **Cálculo automático de datos derivados**
- **Registro de eventos**

Nota: El Apéndice C trata cada uno de estos ejemplos de forma más detallada.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Supuestos de Aplicación de Negocio para la Implementación de Disparadores

Desarrolle disparadores de base de datos para mejorar las funciones que, de lo contrario, no podría implementar el servidor de Oracle, o como alternativa a las que proporciona el servidor.

Función	Mejora
Seguridad	El servidor de Oracle permite el acceso a la tabla a los usuarios o roles. Los disparadores permiten el acceso a la tabla según los valores de datos.
Auditoría	El servidor de Oracle realiza el seguimiento de las operaciones de datos en las tablas. Los disparadores realizan el seguimiento de los valores para las operaciones de datos en las tablas.
Integridad de los datos	El servidor de Oracle fuerza las restricciones de integridad. Los disparadores implementan las reglas complejas de integridad.
Integridad referencial	El servidor de Oracle fuerza reglas estándar de integridad referencial. Los disparadores implementan las funciones no estándar.
Replicación de tablas	El servidor de Oracle copia tablas de forma asíncrona en las instantáneas. Los disparadores copian tablas de forma síncrona en las réplicas.
Datos derivados	El servidor de Oracle calcula los valores de datos derivados de forma manual. Los disparadores calculan los valores de datos derivados de forma automática.
Registro de eventos	El servidor de Oracle registra los eventos explícitamente. Los disparadores registran los eventos de forma transparente.

Visualización de Información de Disparador

Puede ver la siguiente información del disparador:

- **Vista del diccionario de datos USER_OBJECTS: información del objeto**
- **Vista del diccionario de datos USER_TRIGGERS: texto del disparador**
- **Vista del diccionario de datos USER_ERRORS: errores de sintaxis PL/SQL (errores de compilación) del disparador**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Visualización de Información de Disparador

La transparencia muestra las vistas del diccionario de datos a las que puede acceder para obtener la información relacionada con los disparadores.

La vista USER_OBJECTS contiene el nombre y estado del disparador, así como la fecha y la hora en las que se creó.

La vista USER_ERRORS contiene los detalles de los errores de compilación producidos durante la compilación del disparador. El contenido de estas vistas es similar al de los subprogramas.

La vista USER_TRIGGERS contiene detalles como el nombre, el tipo, el evento disparador, la tabla en la que se ha creado el disparador y el cuerpo del disparador.

La sentencia SELECT Username FROM USER_USERS; proporciona el nombre del propietario del disparador, no el nombre del usuario que está actualizando la tabla.

Uso de USER_TRIGGERS

Columna	Descripción de la Columna
TRIGGER_NAME	Nombre del disparador
TRIGGER_TYPE	El tipo es BEFORE, AFTER, INSTEAD OF
TRIGGERING_EVENT	Operación DML que arranca el disparador
TABLE_NAME	Nombre de la tabla de base de datos
REFERENCING_NAMES	Nombre utilizado para :OLD y :NEW
WHEN_CLAUSE	Cláusula when_clause utilizada
STATUS	Estado del disparador
TRIGGER_BODY	Acción que se va a realizar

* Lista de columnas abreviada

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de USER_TRIGGERS

Si el archivo de origen no está disponible, puede utilizar iSQL*Plus para regenerarlo desde USER_TRIGGERS. También puede examinar las vistas ALL_TRIGGERS y DBA_TRIGGERS, que contiene la columna adicional OWNER para el propietario del objeto.

Listado de Códigos de Disparadores

```
SELECT trigger_name, trigger_type, triggering_event,
       table_name, referencing_names,
       status, trigger_body
  FROM user_triggers
 WHERE trigger_name = 'RESTRICT_SALARY';
```

TRIGGER_NAME	TRIGGER_TYPE	TRIGGERING_EVENT	TABLE_NAME	REFERENCING_NAMES	WHEN_CLAUS	STATUS	trigger_body
RESTRICT_SALARY	BEFORE EACH ROW	INSERT OR UPDATE	EMPLOYEES	REFERENCING NEW AS NEW OLD AS OLD		ENABLED	BEGIN IF NOT (:NEW.JOB_ID IN ('AD_PRES','AD_VP')) AND :NEW.SAL

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Ejemplo

Utilice la vista del diccionario de datos USER_TRIGGERS para mostrar la información acerca del disparador RESTRICT_SALARY.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Utilizar disparadores de base de datos avanzados
- Mostrar las reglas mutantes y restringidas para los disparadores
- Describir la aplicación real de los disparadores
- Gestionar disparadores
- Ver la información del disparador

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Práctica 11: Visión General

En esta práctica se abordan los siguientes temas:

- **Creación de disparadores avanzados para gestionar las reglas de integridad de datos**
- **Creación de disparadores que provocan una excepción de tabla mutante**
- **Creación de disparadores que utilizan el estado del paquete para resolver el problema de la tabla mutante**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica 11: Visión General

En esta práctica, implemente una regla de negocio sencilla para garantizar la integridad de los datos de los salarios de los empleados con respecto al rango de salarios válidos para sus trabajos. Cree un disparador para esta regla.

Durante este proceso, los nuevos disparadores darán lugar a un efecto en cascada con disparadores creados en la práctica de la lección anterior. El efecto en cascada originará una excepción de tabla mutante en la tabla `JOB$`. A continuación, cree un paquete PL/SQL y disparadores adicionales para resolver el problema de la tabla mutante.

Práctica 11

1. Los empleados reciben un aumento de sueldo automáticamente si el salario mínimo de un trabajo se aumenta a un valor superior que su salario actual. Implemente este requisito con un procedimiento empaquetado al que llame el disparador de la tabla JOBS. Cuando intenta actualizar el salario mínimo en la tabla JOBS e intenta actualizar el salario de los empleados, el disparador CHECK_SALARY intenta leer la tabla JOBS, que está sujeta a cambios y obtendrá una excepción de tabla mutante que se resuelve creando un nuevo paquete y disparadores adicionales.
 - a. Actualice el paquete EMP_PKG (de la práctica 7) agregando un procedimiento denominado SET_SALARY que actualiza los salarios de los empleados. Este procedimiento acepta dos parámetros: el identificador de trabajo para estos salarios que es posible que tenga que actualizarse y el nuevo salario mínimo para el identificador de trabajo. El procedimiento define todos los salarios de los empleados en el mínimo para su trabajo si el salario actual es menor que el nuevo valor mínimo.
 - b. Cree un disparador de fila denominado UPD_MINSALARY_TRG en la tabla JOBS que llame al procedimiento EMP_PKG.SET_SALARY, cuando el salario mínimo de la tabla JOBS se actualice para un identificador de trabajo especificado.
 - c. Escriba una consulta para mostrar el identificador de empleado, el apellido, el identificador de trabajo, el salario actual y el salario mínimo para los empleados que sean programadores, es decir, su JOB_ID es 'IT_PROG'. A continuación, actualice el salario mínimo en la tabla JOBS para aumentarlo en 1.000 dólares. ¿Qué sucede?
2. Para resolver el problema de la tabla mutante, cree JOBS_PKG para mantener en memoria una copia de las filas de la tabla JOBS. A continuación, se modifica el procedimiento CHECK_SALARY para utilizar los datos del paquete en vez de emitir una consulta en una tabla mutante para evitar la excepción. Sin embargo, se debe crear un disparador de sentencia BEFORE INSERT OR UPDATE en la tabla EMPLOYEES para inicializar el estado del paquete JOBS_PKG antes de que el disparador de fila CHECK_SALARY se arranque.
 - a. Cree un nuevo paquete denominado JOBS_PKG con la siguiente especificación:

```
PROCEDURE initialize;
FUNCTION get_minsalary(jobid VARCHAR2) RETURN NUMBER;
FUNCTION get_maxsalary(jobid VARCHAR2) RETURN NUMBER;
PROCEDURE set_minsalary(jobid VARCHAR2,min_salary NUMBER);
PROCEDURE set_maxsalary(jobid VARCHAR2,max_salary NUMBER);
```
 - b. Implemente el cuerpo de JOBS_PKG donde:

Se declara una tabla de índice PL/SQL privada denominada jobs_tabtype indexada por un tipo de cadena basada en JOBS.JOB_ID%TYPE.

Se declara una variable privada denominada jobstab basada en jobs_tabtype. El procedimiento INITIALIZE lee las filas en la tabla JOBS con un bucle de cursor y utiliza el valor JOB_ID para el índice jobstab que se le asigne a la fila correspondiente. La función GET_MINSalary utiliza un parámetro jobid como índice para jobstab y devuelve min_salary para ese elemento. La función GET_MAXSalary utiliza un parámetro jobid como índice para jobstab y devuelve max_salary para ese elemento.

Práctica 11 (continuación)

El procedimiento SET_MINSALARY utiliza jobid como índice para jobstab para definir el campo de su elemento min_salary en el valor del parámetro min_salary.

El procedimiento SET_MAXSALARY utiliza jobid como índice para jobstab para definir el campo de su elemento max_salary en el valor del parámetro max_salary.

- c. Copie el procedimiento CHECK_SALARY de la práctica 10, ejercicio 1^a, y modifique el código sustituyendo la consulta de la tabla JOBS con sentencias para definir las variables locales minsal y maxsal con valores de los datos JOBS_PKG llamando a las funciones GET_*SALARY apropiadas. Este paso eliminará la excepción de disparador mutante.
 - d. Implemente un disparador de sentencia BEFORE INSERT OR UPDATE denominado INIT_JOBPKG_TRG que utilice la sintaxis CALL para llamar al procedimiento JOBS_PKG. INITIALIZE con el fin de garantizar que el estado del paquete sea actual antes de que se realicen las operaciones DML.
 - e. Pruebe los cambios de código ejecutando la consulta para mostrar los empleados que son programadores, a continuación, emita una sentencia de actualización para aumentar el salario mínimo del tipo de trabajo IT_PROG en 1000 en la tabla JOBS, seguido de una consulta sobre los empleados con tipo de trabajo IT_PROG para comprobar los cambios resultantes. ¿Los salarios de qué empleados se han definido en el mínimo para su trabajo?
3. Debido a que CHECK_SALARY_TRG arranca el procedimiento CHECK_SALARY, antes de insertar o actualizar un empleado, debe comprobar si aún funciona como se esperaba.
 - a. Pruébelo agregando un nuevo empleado mediante EMP_PKG. ADD_EMPLOYEE con los siguientes parámetros: ('Steve', 'Morse', 'SMORSE', sal => 6500). ¿Qué sucede?
 - b. Para corregir el problema encontrado al agregar o actualizar un empleado, cree un disparador de sentencia BEFORE INSERT OR UPDATE denominado EMPLOYEE_INITJOBS_TRG en la tabla EMPLOYEES que llame al procedimiento JOBS_PKG. INITIALIZE. Utilice la sintaxis CALL en el cuerpo del disparador.
 - c. Pruebe el disparador agregando el empleado Steve Morse de nuevo. Confirme el registro insertado en la tabla employees mostrando el identificador de empleado, nombre y apellido, salario, identificador de trabajo e identificador de departamento.

Descripción e Influencia del Compilador PL/SQL

12

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

Objetivos

Al finalizar esta lección, debería estar capacitado para lo siguiente:

- Describir las compilaciones nativas e interpretadas
- Enumerar las funciones de la compilación nativa
- Cambiar entre compilaciones nativas e interpretadas
- Definir parámetros que influyan en la compilación PL/SQL
- Consultar vistas de diccionario de datos sobre la compilación del código PL/SQL
- Utilizar el mecanismo de advertencia del compilador y el paquete DBMS_WARNING para implementar advertencias del compilador

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Objetivos

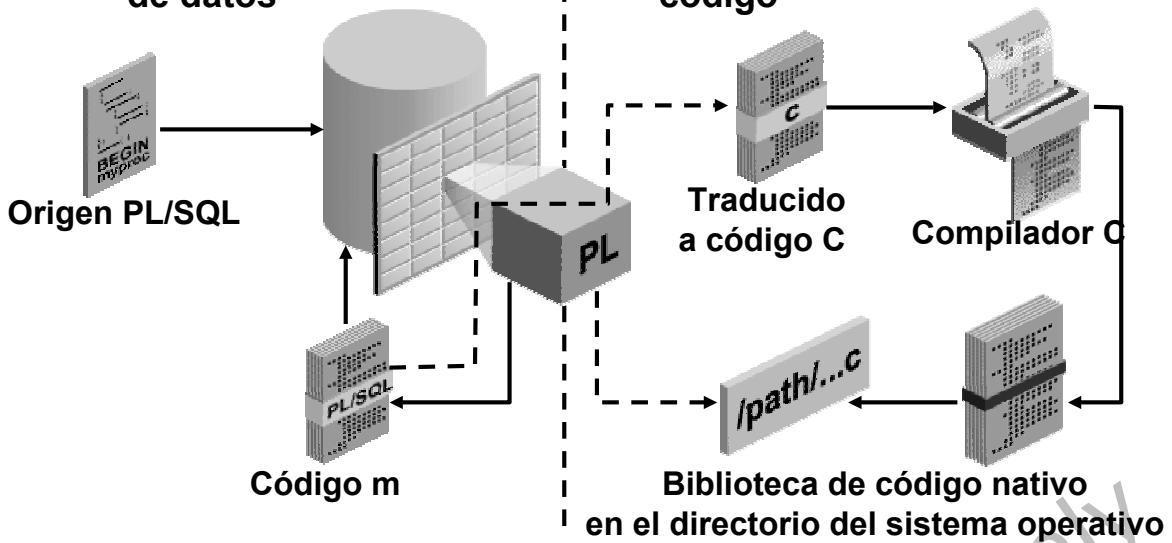
En esta lección, aprenderá a distinguir entre la compilación nativa e interpretada del código PL/SQL. En la lección se trata cómo utilizar la compilación nativa, que es la por defecto, para la base de datos Oracle 10g que tiene la ventaja de tener un tiempo de ejecución más rápido para el código PL/SQL.

También aprenderá a influir en los valores del compilador mediante la definición de parámetros de sesión variables, o bien con la interfaz programática proporcionada por el paquete DBMS_WARNING. En la lección se trata la consulta de valores de compilación mediante las vistas de diccionario de datos USER_STORED_SETTINGS y USER_PLSQL_OBJECTS.

Compilación Nativa e Interpretada

Código interpretado

- Compilado en código m
- Almacenado en la base de datos



Código compilado de forma nativa

- C traducido y compilado
- Copiado en una biblioteca de código

Copyright © 2004, Oracle. Todos los Derechos Reservados.

ORACLE

Compilación Nativa e Interpretada

Como se muestra en la transparencia, a la izquierda de la línea de puntos vertical, una unidad de programa procesada como PL/SQL interpretado se compila en código que puede leer una máquina (código m), que se almacena en la base de datos y se interpreta en tiempo de ejecución.

A la derecha de la línea de puntos vertical, el código PL/SQL está sujeto a la compilación nativa, en la que las sentencias PL/SQL se compilan en código m que se traduce en código C. No se retiene el código m. El código C se compila con el compilador C habitual y se enlaza al proceso de Oracle por medio de la biblioteca de código de máquina nativo. La biblioteca de código se almacena en la base de datos, pero se copia en una ruta de acceso de directorio especificada en el sistema operativo, desde la que se carga en tiempo de ejecución. El código nativo omite la interpretación de código típica en tiempo de ejecución.

Nota: La compilación nativa no puede acelerar demasiado las sentencias SQL llamadas desde PL/SQL, pero es más eficaz para los procedimientos PL/SQL que realizan muchos cálculos y que no pasan la mayor parte del tiempo ejecutando SQL.

Puede compilar de forma nativa los paquetes proporcionados por Oracle y su propio código PL/SQL. La compilación de todo el código PL/SQL de la base de datos significa que verá la aceleración en su propio código y en todos los paquetes PL/SQL incorporados. Si decide que tendrá una considerable mejora del rendimiento en las operaciones de base de datos gracias a la compilación nativa de PL/SQL, Oracle recomienda compilar toda la base de datos mediante el valor NATIVE.

Funciones y Ventajas de la Compilación Nativa

La compilación nativa:

- Utiliza un archivo **makefile** genérico que usa el siguiente software del sistema operativo:
 - Compilador C
 - Enlace
 - Utilidad Make
- Genera bibliotecas compartidas que se copian en el sistema de archivos y se cargan en tiempo de ejecución.
- Proporciona un mejor rendimiento, hasta un 30% más rápido que el código interpretado, para operaciones de procedimiento que realizan muchos cálculos.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Funciones y Ventajas de la Compilación Nativa

El proceso de compilación nativa de PL/SQL utiliza un archivo **makefile**, denominado **spnc_makefile.mk**, ubicado en el directorio **\$ORACLE_HOME/plsql**. El archivo **makefile** se procesa mediante la utilidad Make que llama al compilador C, que es el enlace en el sistema operativo soportado, para compilar y enlazar el código C resultante en las bibliotecas compartidas. Las bibliotecas compartidas se almacenan en la base de datos y se copian en el sistema de archivos. En tiempo de ejecución, las bibliotecas compartidas se cargan y ejecutan cuando se llama al subprograma PL/SQL.

De acuerdo con las recomendaciones de Arquitectura Flexible Óptima (OFA), las bibliotecas compartidas se deben almacenar cerca de los archivos de datos. El código C se ejecuta más rápidamente que PL/SQL, pero tarda más en compilarse que el código m. La compilación nativa de PL/SQL proporciona las mayores mejoras de rendimiento para operaciones de procedimiento que realizan muchos cálculos.

Los ejemplos de este tipo de operaciones son aplicaciones de almacén de datos y aplicaciones con muchas transformaciones de datos por parte del servidor para la visualización. En dichos casos, puede esperar aumentos de velocidad de hasta un 30%.

Consideraciones para el Uso de la Compilación Nativa

Considere lo siguiente:

- Las herramientas de depuración para PL/SQL no pueden depurar el código compilado de forma nativa.
- Es más lento compilar el código compilado de forma nativa que el interpretado.
- Una gran cantidad de subprogramas compilados de forma nativa puede afectar al rendimiento debido a las limitaciones impuestas por el sistema operativo al manejar bibliotecas compartidas. Las limitaciones del directorio del sistema operativo se pueden gestionar mediante la definición de parámetros de inicialización de la base de datos:
 - `PLSQL_NATIVE_LIBRARY_SUBDIR_COUNT` y
 - `PLSQL_NATIVE_LIBRARY_DIR`

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Limitaciones de la Compilación Nativa

Como se ha indicado, la ventaja clave del código compilado de forma nativa es una ejecución un 30% más rápida, especialmente para el código PL/SQL que realiza muchos cálculos. Considere lo siguiente:

- Las herramientas de depuración para PL/SQL no manejan procedimientos compilados para la ejecución nativa. Por lo tanto, utilice la compilación interpretada en entornos de desarrollo y compile de forma nativa el código de un entorno de producción.
- El tiempo de compilación aumenta cuando se utiliza la compilación nativa, debido al requisito de traducir la sentencia PL/SQL a su C equivalente y de ejecutar la utilidad Make para llamar al compilador C y al enlace para generar la biblioteca de código compilado resultante.
- Si se compilan muchos procedimientos y paquetes (más de 5.000) para la ejecución nativa, puede que un gran número de objetos compartidos de un único directorio afecte al rendimiento. Las limitaciones del directorio del sistema operativo se pueden gestionar mediante la distribución automática de bibliotecas en varios subdirectorios. Para ello, realice las siguientes tareas antes de compilar de forma nativa el código PL/SQL:
 - Defina el parámetro de inicialización de la base de datos `PLSQL_NATIVE_LIBRARY_SUBDIR_COUNT` en un valor grande, como 1.000, antes de crear la base de datos o compilar los paquetes o procedimientos PL/SQL.
 - Cree subdirectorios `PLSQL_NATIVE_LIBRARY_SUBDIR_COUNT` en la ruta de acceso especificada en el parámetro de inicialización `PLSQL_NATIVE_LIBRARY_DIR`.

Parámetros que Influyen en la Compilación

Parámetros del sistema

- **Se definen en el archivo `initSID.ora` o mediante SPFILE**

```
PLSQL_NATIVE_LIBRARY_DIR = full-directory-path-name  
PLSQL_NATIVE_LIBRARY_SUBDIR_COUNT = count
```

Parámetros del sistema o de sesión

```
PLSQL_COMPILER_FLAGS = 'NATIVE' or 'INTERPRETED'
```

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Parámetros que Influyen en la Compilación

En todas las circunstancias, tanto si pretende compilar una base de datos como NATIVE como si planea compilar unidades PL/SQL individuales a nivel de sesión, debe definir todos los parámetros necesarios.

Los parámetros del sistema se definen en el archivo `initSID.ora` mediante el mecanismo SPFILE. Dos parámetros que se definen como parámetros de nivel de sistema son los siguientes:

- El valor `PLSQL_NATIVE_LIBRARY_DIR`, que especifica la ruta de acceso completa y el nombre de directorio utilizado para almacenar las bibliotecas compartidas que contienen código PL/SQL compilado de forma nativa.
- El valor `PLSQL_NATIVE_LIBRARY_SUBDIR_COUNT`, que indica el número de subdirectorios del directorio especificado por el parámetro `PLSQL_NATIVE_LIBRARY_DIR`. Utilice un archivo de comandos para crear directorios con nombres consistentes (por ejemplo, `d0, d1, d2`, etc.) y, a continuación, el compilador PL/SQL distribuirá automáticamente las bibliotecas entre estos subdirectorios.

Por defecto, las unidades de programa PL/SQL se mantienen en un directorio.

El parámetro `PLSQL_COMPILER_FLAGS` se puede definir en un valor NATIVE o INTERPRETED, como una inicialización de la base de datos para un valor por defecto para todo el sistema o para cada sesión mediante una sentencia `ALTER SESSION`.

Cambio entre Compilación Nativa e Interpretada

- Definición de compilación nativa
 - Para el sistema

```
ALTER SYSTEM SET plsql_compiler_flags='NATIVE';
```

- Para la sesión

```
ALTER SESSION SET plsql_compiler_flags='NATIVE';
```

- Definición de la compilación interpretada
 - Para el nivel de sistema

```
ALTER SYSTEM  
    SET plsql_compiler_flags='INTERPRETED';
```

- Para la sesión

```
ALTER SESSION  
    SET plsql_compiler_flags='INTERPRETED';
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Cambio entre Compilación Nativa e Interpretada

El parámetro PLSQL_COMPILER_FLAGS determina si el código PL/SQL está compilado de forma nativa o es interpretado, y determina si se incluye la información de depuración. El valor por defecto es INTERPRETED, NON_DEBUG. Para activar la compilación nativa de PL/SQL, debe definir el valor PLSQL_COMPILER_FLAGS en NATIVE.

Si compila toda la base de datos como NATIVE, Oracle recomienda definir PLSQL_COMPILER_FLAGS a nivel de sistema.

Para definir el tipo de compilación a nivel de sistema (lo que normalmente realiza un DBA), ejecute las siguientes sentencias:

```
ALTER SYSTEM SET plsql_compiler_flags='NATIVE'  
ALTER SYSTEM SET plsql_compiler_flags='INTERPRETED'
```

Para definir el tipo de compilación a nivel de sesión, ejecute una de las siguientes sentencias:

```
ALTER SESSION SET plsql_compiler_flags='NATIVE'  
ALTER SESSION SET plsql_compiler_flags='INTERPRETED'
```

Visualización de Información de Compilación en el Diccionario de Datos

Consultar la información de las siguientes vistas:

- **USER_STORED_SETTINGS**
- **USER_PLSQL_OBJECTS**

Por ejemplo:

```
SELECT param_value
  FROM user_stored_settings
 WHERE param_name = 'PLSQL_COMPILER_FLAGS'
   AND object_name = 'GET_EMPLOYEES';
```

Nota: La columna PARAM_VALUE tiene el valor NATIVE para los procedimientos que se compilan para la ejecución nativa; de lo contrario, tiene el valor INTERPRETED.

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Visualización de Información de Compilación en el Diccionario de Datos

Para comprobar si un procedimiento existente se ha compilado para la ejecución nativa, puede consultar las siguientes vistas del diccionario de datos:

```
[USER | ALL | DBA]_STORED_SETTINGS
[USER | ALL | DBA ]_PLSQL_OBJECTS
```

En el ejemplo de la transparencia se muestra cómo puede comprobar el estado del procedimiento denominado GET_EMPLOYEES. La columna PARAM_VALUE tiene el valor NATIVE para los procedimientos que se compilan para la ejecución nativa; de lo contrario, tiene el valor INTERPRETED.

Una vez que los procedimientos se han compilado de forma nativa y se han convertido en bibliotecas compartidas, se enlazan automáticamente con el proceso de Oracle. No es necesario que reinicie la base de datos ni que mueva las bibliotecas compartidas a otra ubicación. Puede llamar una y otra vez distintos procedimientos almacenados, ya sean todos compilados interpretados (valor por defecto), todos compilados para la ejecución nativa, o bien una mezcla de ambos.

Puesto que el valor PLSQL_COMPILER_FLAGS se almacena en la unidad de biblioteca para cada procedimiento, los procedimientos compilados para la ejecución nativa se compilan de la misma forma cuando el procedimiento se recompila automáticamente después de invalidarse, por ejemplo, cuando se vuelve a crear una tabla de la que depende.

Uso de la Compilación Nativa

Para activar la compilación nativa, realice los siguientes pasos:

- 1. Edite el archivo `makefile` proporcionado e introduzca rutas de acceso adecuadas y otros valores para el sistema.**
- 2. Defina el parámetro `PLSQL_COMPILER_FLAGS` (a nivel de sistema o de sesión) en el valor `NATIVE`. El valor por defecto es `INTERPRETED`.**
- 3. Compile los procedimientos, las funciones y los paquetes.**
- 4. Consulte el diccionario de datos para comprobar que un procedimiento se compila para la ejecución nativa.**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de la Compilación Nativa

Para activar la compilación nativa, realice los siguientes pasos:

1. Compruebe y edite el compilador, el enlace, las rutas de acceso a utilidades y otros valores, si es necesario.
2. Defina `PLSQL_COMPILER_FLAGS` en `NATIVE`.
3. Compile los procedimientos, las funciones y los paquetes. La compilación se puede realizar:
 - Mediante las sentencias `ALTER PROCEDURE`, `ALTER FUNCTION` o `ALTER PACKAGE` adecuadas con la opción `COMPILE`
 - Borrando el procedimiento y volviéndolo a crear
 - Ejecutando uno de los archivos de comandos de SQL*Plus que configura un juego de paquetes proporcionados por Oracle
 - Creando una base de datos mediante un archivo de inicialización preconfigurado con `PLSQL_COMPILER_FLAGS` definido en `NATIVE`
4. Confirme el tipo de compilación mediante las tablas adecuadas del diccionario de datos.

Nota: Las dependencias entre los objetos de base de datos se manejan del mismo modo que en las versiones anteriores de la base de datos Oracle. Si cambia un objeto del que depende una unidad de programa PL/SQL compilada de forma nativa, el módulo PL/SQL se invalida. La próxima vez que se ejecute la misma unidad de programa, RDBMS intentará volver a validar el módulo. Cuando un módulo se recompila como parte de la revalidación, se compila con el valor utilizado la última vez que se compiló el módulo y se guarda en la vista `*_STORED_SETTINGS`.

Infraestructura de Advertencias del Compilador

El compilador PL/SQL de la base de datos Oracle 10g se ha mejorado para producir advertencias para subprogramas.

Los niveles de advertencia:

- **Se pueden definir:**
 - De forma declarativa con el parámetro de inicialización `PLSQL_WARNINGS`
 - Mediante programación con el paquete `DBMS_WARNINGS`
- **Se organizan en tres categorías: grave, de rendimiento e informativa**
- **Se pueden activar y desactivar por categoría o por un mensaje concreto**

Ejemplos de mensajes de advertencia:

SP2-0804: Procedimiento creado con advertencias de compilación

PLW-07203: El parámetro 'IO_TBL' puede aprovechar el uso de la indicación del compilador NOCOPY

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Infraestructura de Advertencias del Compilador

El compilador PL/SQL de Oracle puede emitir advertencias cuando compile subprogramas que producen resultados ambiguos o utilizan construcciones ineficaces. Puede activar y desactivar de forma selectiva estas advertencias:

- De forma declarativa definiendo el parámetro de inicialización `PLSQL_WARNINGS`
- Mediante programación con el paquete `DBMS_WARNINGS`

El nivel de advertencia se organiza en las siguientes categorías: grave, de rendimiento e informativa. Los niveles de advertencia se pueden activar o desactivar por categoría o por un número de mensaje de advertencia concreto.

Ventajas de las Advertencias del Compilador

El uso de advertencias del compilador puede ayudar a lo siguiente:

- Hacer que los programas sean más robustos y evitar problemas en tiempo de ejecución
- Identificar posibles problemas de rendimiento
- Indicar factores que producen resultados no definidos

Nota: Puede activar la comprobación de determinadas condiciones de advertencia cuando éstas no sean lo suficientemente graves para producir un error e impedir que compile un subprograma.

Definición de Niveles de Advertencia del Compilador

Definir el parámetro de inicialización PLSQL_WARNINGS para permitir que la base de datos emita mensajes de advertencia.

```
ALTER SESSION SET PLSQL_WARNINGS = 'ENABLE:SEVERE',
'DISABLE:INFORMATIONAL';
```

- **PLSQL_WARNINGS combina un valor de cualificador ENABLE, DISABLE o ERROR con una lista separada por comas de números de mensaje o con uno de los valores de modificador siguientes:**
 - ALL, SEVERE, INFORMATIONAL o PERFORMANCE
- **Los mensajes de advertencia utilizan un prefijo PLW.**
PLW-07203: El parámetro 'IO_TBL' puede aprovechar el uso de la indicación del compilador NOCOPY.

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Definición de Niveles de Advertencia del Compilador

El valor PLSQL_WARNINGS activa o desactiva la creación de informes de mensajes de advertencia por parte del compilador PL/SQL y especifica los mensajes de advertencia que se deben mostrar como errores. El parámetro PLSQL_WARNINGS se puede definir para el sistema mediante el archivo de inicialización o la sentencia ALTER SYSTEM, o bien para la sesión mediante la sentencia ALTER SESSION como se muestra en el ejemplo de la transparencia. Por defecto, el valor se define en DISABLE:ALL.

El valor del parámetro comprende una lista separada por comas de palabras clave de cualificador y modificador entre comillas, en la que las palabras clave están separadas por dos puntos. Los valores de cualificador son:

- ENABLE: Para activar una advertencia concreta o un juego de advertencias.
- DISABLE: Para desactivar una advertencia concreta o un juego de advertencias.
- ERROR: Para tratar una advertencia concreta o un juego de advertencias como errores.

El valor de modificador ALL se aplica a todos los mensajes de advertencia. SEVERE, INFORMATIONAL y PERFORMANCE se aplican a mensajes de su propia categoría y a una lista entera de mensajes de advertencia específicos. Por ejemplo:

```
PLSQL_WARNINGS='ENABLE:SEVERE','DISABLE:INFORMATIONAL';
PLSQL_WARNINGS='DISABLE:ALL';
PLSQL_WARNINGS='DISABLE:5000','ENABLE:5001','ERROR:5002';
PLSQL_WARNINGS='ENABLE:(5000,5001)','DISABLE:(6000)';
```

Instrucciones para el Uso de **PLSQL_WARNINGS**

El valor PLSQL_WARNINGS:

- **Se puede definir en DEFERRED a nivel de sistema**
- **Se almacena con cada subprograma compilado**
- **Que es actual para la sesión se utiliza, por defecto, al recompilar con:**
 - **Una sentencia CREATE OR REPLACE**
 - **Una sentencia ALTER...COMPILE**
- **Que se almacena con el subprograma compilado se utiliza si especifica REUSE SETTINGS al recompilar con una sentencia ALTER...COMPILE**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Instrucciones para el Uso de PLSQL_WARNINGS

Como ya se ha indicado, el parámetro PLSQL_WARNINGS se puede definir a nivel de sesión o a nivel de sistema. Al definirlo a nivel de sistema, puede incluir el valor DEFERRED para que se aplique a futuras sesiones pero no a la actual.

Los valores para el parámetro PLSQL_WARNINGS se almacenan junto con cada subprograma compilado. Si recompila el subprograma con una sentencia CREATE OR REPLACE, se utilizarán los valores actuales para dicha sesión. Si recompila un subprograma con una sentencia ALTER...COMPILE, se utiliza el valor de la sesión actual a menos que especifique la cláusula REUSE SETTINGS en dicha sentencia, que utiliza el valor original almacenado con el subprograma.

Paquete DBMS_WARNING

El paquete DBMS_WARNING proporcionar una forma de manipular mediante programación el comportamiento de los valores de advertencia PL/SQL del sistema o sesión actual. Con subprogramas DBMS_WARNING, puede:

- **Consultar valores existentes**
- **Modificar los valores para requisitos específicos o restaurar valores originales**
- **Suprimir los valores**

Ejemplo: Guardado y restauración de valores de advertencia para un entorno de desarrollo que llama al código que compila subprogramas PL/SQL y suprime advertencias debido a los requisitos del negocio

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Paquete DBMS_WARNING

El paquete DBMS_WARNING proporciona una forma de manipular el comportamiento de los mensajes de advertencia PL/SQL, en concreto mediante la lectura y el cambio del valor del parámetro de inicialización PLSQL_WARNINGS para controlar los tipos de advertencias que se deben suprimir, mostrar o tratar como errores. Este paquete proporciona la interfaz para consultar, modificar y suprimir los valores actuales del sistema o sesión.

El paquete DBMS_WARNINGS es valioso si está escribiendo un entorno de desarrollo que compila subprogramas PL/SQL. Con las rutinas de la interfaz del paquete, puede controlar los mensajes de advertencia PL/SQL mediante programación para que se adapten a los requisitos.

Por ejemplo: Si que escribe código para compilar código PL/SQL, sabe que el compilador emitirá advertencias de rendimiento al transferir variables de recopilación como parámetros OUT o IN OUT sin especificar la indicación NOCOPY. El entorno general que llama a la utilidad de compilación puede tener valores de nivel de advertencia adecuados o no. En cualquier caso, las reglas de negocio indican que el juego del entorno de llamada se debe mantener y que el proceso de compilación debe suprimir las advertencias. Al llamar a subprogramas del paquete DBMS_WARNINGS, puede detectar los valores de advertencia actuales, cambiar el valor para que se ajuste a los requisitos del negocio y restaurar el valor original una vez terminado el procesamiento.

Uso de Procedimientos DBMS_WARNING

- Los procedimientos de paquetes cambian las advertencias PL/SQL:

```
ADD_WARNING_SETTING_CAT(w_category,w_value,scope)
ADD_WARNING_SETTING_NUM(w_number,w_value,scope)
SET_WARNING_SETTING_STRING(w_value, scope)
```

- Todos los parámetros son parámetros IN y tiene el tipo de dato VARCHAR2. Sin embargo, el parámetro w_number es un tipo de dato NUMBER.
- Los valores de cadena de parámetros no son sensibles a mayúsculas y minúsculas.
- Los valores del parámetro w_value son ENABLE, DISABLE y ERROR.
- Los valores de w_category son ALL, INFORMATIONAL, SEVERE y PERFORMANCE.
- El valor de scope es SESSION o SYSTEM. El uso de SYSTEM necesita el privilegio ALTER SYSTEM.

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de Procedimientos DBMS_WARNING

Los procedimientos de paquetes son los siguientes:

- ADD_WARNING_SETTING_CAT: Modifica los valores de advertencia actuales de sesión o de sistema de warning_category previamente proporcionados.
- ADD_WARNING_SETTING_NUM: Modifica los valores de advertencia actuales de sesión o de sistema de warning_number previamente proporcionados.
- SET_WARNING_SETTING_STRING: Sustituye los valores anteriores por el nuevo valor.

Mediante SET_WARNING_SETTING_STRING, puede definir un valor de advertencia. Si tiene varios valores de advertencia, debe realizar los siguientes pasos:

1. Llamar a SET_WARNING_SETTING_STRING para definir la cadena inicial de valor de advertencia.
2. Llamar a ADD_WARNING_SETTING_CAT (o ADD_WARNING_SETTING_NUM) repetidas veces para agregar valores adicionales a la cadena inicial.

A continuación se muestra un ejemplo para establecer la siguiente cadena de valor de advertencia en la sesión actual:

ENABLE: INFORMATIONAL, DISABLE: PERFORMANCE, ENABLE: SEVERE

Ejecute las dos líneas de código siguientes:

```
dbms_warning.set_warning_setting_string('ENABLE:ALL','session');
dbms_warning.add_warning_setting_cat('PERFORMANCE','disable',
'session');
```

Uso de Funciones de DBMS_WARNING

- Las funciones de paquete leen las advertencias PL/SQL:

```
GET_CATEGORY(w_number) RETURN VARCHAR2  
GET_WARNING_SETTING_CAT(w_category) RETURN VARCHAR2  
GET_WARNING_SETTING_NUM(w_number) RETURN VARCHAR2  
GET_WARNING_SETTING_STRING RETURN VARCHAR2
```

- GET_CATEGORY devuelve un valor ALL, INFORMATIONAL, SEVERE o PERFORMANCE para un número de mensaje determinado.
- GET_WARNING_SETTING_CAT devuelve ENABLE, DISABLE o ERROR como valor de advertencia actual para un nombre de categoría y GET_WARNING_SETTING_NUM devuelve el valor para un número de mensaje determinado.
- GET_WARNING_SETTING_STRING devuelve toda la cadena de advertencia para la sesión actual.

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de Funciones de DBMS_WARNING

A continuación se muestra una lista de funciones de paquete:

- GET_CATEGORY devuelve el nombre de categoría para el número de mensaje especificado.
- GET_WARNING_SETTING_CAT devuelve el valor de advertencia de sesión actual para la categoría especificada.
- GET_WARNING_SETTING_NUM devuelve el valor de advertencia de sesión actual para el número de mensaje especificado.
- GET_WARNING_SETTING_STRING devuelve toda la cadena de advertencia para la sesión actual.

Para determinar los valores actuales de advertencia de sesión, introduzca:

```
EXECUTE DBMS_OUTPUT.PUT_LINE( -  
DBMS_WARNING.GET_WARNING_SETTING_STRING) ;
```

Para determinar la categoría para el número de mensaje de advertencia PLW-07203, utilice:

```
EXECUTE DBMS_OUTPUT.PUT_LINE( -  
DBMS_WARNING.GET_CATEGORY(7203))
```

La cadena de resultado debe ser PERFORMANCE.

Nota: Los números de mensaje se deben especificar como enteros positivos, porque el tipo de dato para el parámetro GET_CATEGORY es PLS_INTEGER (que permite valores enteros positivos).

Uso de DBMS_WARNING: Ejemplo

Considere el siguiente supuesto:

Guardar los valores de advertencia actuales, desactivar las advertencias para la categoría PERFORMANCE, compilar un paquete PL/SQL y restaurar el valor de advertencia original.

```
CREATE PROCEDURE compile(pkg_name VARCHAR2) IS
  warn_value VARCHAR2(200);
  compile_stmt VARCHAR2(200) :=
    'ALTER PACKAGE ' || pkg_name ||' COMPILE';
BEGIN
  warn_value := -- Save current settings
    DBMS_WARNING.GET_WARNING_SETTING_STRING;
  DBMS_WARNING.ADD_WARNING_SETTING_CAT( -- change
    'PERFORMANCE', 'DISABLE', 'SESSION');
  EXECUTE IMMEDIATE compile_stmt;
  DBMS_WARNING.SET_WARNING_SETTING_STRING(--restore
    warn_value, 'SESSION');
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de DBMS_WARNING: Ejemplo

En la transparencia, el ejemplo del procedimiento `compile` se designa para compilar un paquete PL/SQL con nombre. Las reglas de negocio necesitan lo siguiente:

- Las advertencias de la categoría de rendimiento se deben suprimir.
- Los valores de advertencia del entorno de llamada se deben restaurar después de realizar la compilación.

El código no conoce o no se preocupa de cuáles son los valores de advertencia del entorno de llamada; simplemente utiliza la función

`DBMS_WARNING.GET_WARNING_SETTING_STRING` para guardar el valor actual.

Este valor se utiliza para restaurar el valor del entorno de llamada mediante el procedimiento `DBMS_WARNING.SET_WARNING_SETTING_STRING` de la última línea del código de ejemplo. Antes de compilar el paquete con SQL dinámico nativo, el procedimiento `compile` modifica el nivel de advertencia de sesión actual desactivando las advertencias para la categoría PERFORMANCE.

Por ejemplo, el compilador suprimirá las advertencias sobre parámetros PL/SQL transferidos mediante los modos OUT o IN OUT que no especifican la indicación NOCOPY para obtener un mejor rendimiento.

Uso de DBMS_WARNING: Ejemplo

Para probar el procedimiento `compile`, puede utilizar la siguiente secuencia del archivo de comandos en iSQL*Plus:

```
DECLARE
  PROCEDURE print(s VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(s);
  END;
BEGIN
  print('Warning settings before: ' ||
        DBMS_WARNING.GET_WARNING_SETTING_STRING);
  compile('my_package');
  print('Warning settings after: ' ||
        DBMS_WARNING.GET_WARNING_SETTING_STRING);
END;
/
SHOW ERRORS PACKAGE MY_PACKAGE
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Uso de DBMS_WARNING: Ejemplo (continuación)

En la transparencia se muestra un bloque anónimo que se utiliza para mostrar los valores de advertencia actuales para la sesión antes de que se realice la compilación, se ejecuta el procedimiento de compilación y se vuelve a imprimir los valores de advertencia actuales para la sesión. Los valores anteriores y posteriores para la advertencia deben ser idénticos.

La última línea que contiene `SHOW ERRORS PACKAGE MY_PACKAGE` se utiliza para verificar si los mensajes de advertencia de la categoría de rendimiento se suprimen (es decir, no se muestran mensajes de advertencia relacionados con el rendimiento).

Para probar de forma adecuada el comportamiento del procedimiento `compile`, el paquete `MY_PACKAGE` debe contener un subprograma con una recopilación (tabla PL/SQL) especificada como argumento `OUT` o `IN OUT` sin utilizar la indicación `NOCOPY`. Normalmente, con la categoría `PERFORMANCE` activada, se emitirá una advertencia del compilador. Con los ejemplos de código que se muestran en las dos últimas transparencias, se suprime las advertencias relacionadas con la indicación `NOCOPY`.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- **Cambiar entre compilaciones nativas e interpretadas**
- **Definir parámetros que influyan en la compilación nativa de programas PL/SQL**
- **Consultar vistas de diccionario de datos que proporcionen información sobre los valores de compilación PL/SQL**
- **Utilizar el mecanismo de advertencia del compilador PL/SQL:**
 - De forma declarativa definiendo el parámetro `PLSQL_WARNINGS`
 - Mediante programación con el paquete `DBMS_WARNING`

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Resumen

En la lección se tratan los detalles del funcionamiento de las compilaciones nativas e interpretadas y del uso de parámetros para influir en la forma en que se compila el código PL/SQL.

La recomendación clave es activar la compilación nativa por defecto, cuyo resultado es un rendimiento un 30% más rápido (en algunos casos) para la lógica PL/SQL. Los índices de rendimiento han mostrado que la activación de la compilación nativa en la base de datos Oracle 10g tiene como resultado el doble de rendimiento en comparación con las bases de datos Oracle8i y Oracle9i, y hasta el triple del rendimiento de la ejecución del código PL/SQL en un entorno de base de datos Oracle8. Para obtener más información, consulte la documentación técnica de Oracle con el título “*PL/SQL Just Got Faster*”, de Bryn Llewellyn y Charles Wetherell, en la dirección Web de Oracle Technology Network (OTN) en <http://otn.oracle.com>.

En la lección también se abordan las dos formas siguientes para influir en el nuevo sistema de advertencia del compilador que se ha agregado a la base de datos Oracle 10g:

- Definición del parámetro `PLSQL_WARNINGS`
- Uso de la interfaz programática del paquete `DBMS_WARNING`

Práctica 12: Visión General

En esta práctica se abordan los siguientes temas:

- **Activación de la compilación nativa para la sesión y compilación de un procedimiento**
- **Creación de un subprograma para compilar un procedimiento, una función o un paquete PL/SQL; supresión de advertencias para la categoría de advertencia del compilador PERFORMANCE y restauración de los valores originales de advertencia de sesión**
- **Ejecución del procedimiento para compilar un paquete PL/SQL que contenga un procedimiento que utilice una tabla PL/SQL como parámetro IN OUT sin especificar la indicación NOCOPY**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Práctica 12: Visión General

En esta práctica, activará la compilación nativa para la sesión y compilará un procedimiento. A continuación, creará un subprograma para compilar un procedimiento, una función o un paquete PL/SQL y suprimirá las advertencias para la categoría de advertencia del compilador PERFORMANCE. El procedimiento debe restaurar los valores originales de advertencia de sesión. Después, ejecutará el procedimiento para compilar un paquete PL/SQL que creará, el cual contendrá un procedimiento con un parámetro IN OUT sin especificar la indicación NOCOPY.

Práctica 12

1. Modifique el parámetro PLSQL_COMPILER_FLAGS para activar la compilación nativa para la sesión y compilar cualquier subprograma que haya escrito.
 - a. Ejecute el comando ALTER SESSION para activar la compilación nativa.
 - b. Compile el procedimiento EMPLOYEE_REPORT. ¿Qué sucede durante la compilación?
 - c. Ejecute EMPLOYEE_REPORT con el valor 'UTL_FILE' como primer parámetro y 'native_salrepXX.txt' donde XX es el número de estudiante.
 - d. Cambie la compilación para utilizar compilación interpretada.
2. En COMPILE_PKG (de la práctica 6), agregue una versión sobrecargada del procedimiento MAKE, que compilará un procedimiento, función o paquete con nombre.
 - a. En la especificación, declare un procedimiento MAKE que acepte dos argumentos de cadena, uno para el nombre de la construcción PL/SQL y el otro para el tipo de programa PL/SQL, como PROCEDURE, FUNCTION, PACKAGE o PACKAGE BODY.
 - b. En el cuerpo, escriba el procedimiento MAKE para llamar al paquete DBMS_WARNINGS para suprimir la categoría PERFORMANCE. Sin embargo, guarde los valores actuales de las advertencias del compilador antes de modificarlos. A continuación, escriba una sentencia EXECUTE IMMEDIATE para compilar el objeto PL/SQL utilizando una sentencia ALTER...COMPILE adecuada con los valores de parámetros proporcionados. Por último, restaure los valores de las advertencias del compilador que existían para el entorno de llamada antes de que se llame al procedimiento.
3. Escriba un nuevo paquete PL/SQL denominado TEST_PKG que contenga un procedimiento denominado GET_EMPLOYEES que utilice un argumento IN OUT.
 - a. En la especificación, declare el procedimiento GET_EMPLOYEES con dos parámetros: un parámetro de entrada que especifique un identificador de departamento y un parámetro IN OUT que especifique una tabla PL/SQL de filas de empleados.
Indicación: Debe declarar un TYPE en la especificación del paquete para el tipo de dato del parámetro de la tabla PL/SQL.
 - b. En el cuerpo del paquete, implemente el procedimiento GET_EMPLOYEES para recuperar todas las filas de empleados de un departamento especificado en el parámetro IN OUT en la tabla PL/SQL.
Indicación: Utilice la sintaxis SELECT ... BULK COLLECT INTO para simplificar el código.
4. Utilice la sentencia ALTER SESSION para definir PLSQL_WARNINGS para que todas las categorías de las advertencias del compilador estén activadas.
5. Recompile TEST_PKG creado dos pasos antes (en el ejercicio 3). ¿Qué advertencias del compilador (si hay alguna) se muestran?
6. Escriba un bloque anónimo PL/SQL para compilar el paquete TEST_PKG utilizando el procedimiento sobrecargado COMPILE_PKG.MAKE con dos parámetros. El bloque anónimo mostrará el valor de la cadena de advertencia de la sesión actual antes y después de que llame al procedimiento COMPILE_PKG.MAKE. ¿Ve algún mensaje de advertencia? Confirme las observaciones ejecutando el comando SHOW ERRORS PACKAGE para TEST_PKG.