

Oracle Database: Desarrollo de Unidades de Programa en PL/SQL

Volumen I • Guía del Alumno

D64250CS10

Edición 1.0

Enero de 2010

D73734

ORACLE®

Autor

Lauran Serhal

Colaboradores Técnicos y Responsables de Revisión

Anjulaponni Azhagulekshmi
Christian Bauwens
Christoph Burandt
Zarko Cesljas
Yanti Chang
Salome Clement
Laszlo Czinkoczki
Ingrid DelaHaye
Steve Friedberg
Laura Garza
Joel Goodman
Nancy Greenberg
Manish Pawar
Brian Pottle
Helen Robertson
Tulika Srivastava
Ted Witiuk
Hilda Simon

Redactores

Arijit Ghosh
Raj Kumar

Editores

Pavithran Adka
Shaik Mahaboob Basha
Sheryl Domingue

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Renuncia

Este documento contiene información propiedad de Oracle Corporation y se encuentra protegido por la legislación de derechos de autor y otras leyes sobre la propiedad intelectual. Usted sólo podrá realizar copias o imprimir este documento para uso exclusivo por usted en los cursos de formación de Oracle. Este documento no podrá ser modificado ni alterado en modo alguno. Salvo que la legislación de los derechos de autor lo considere un uso excusable o legal o "fair use", no podrá utilizar, compartir, descargar, cargar, copiar, imprimir, mostrar, representar, reproducir, publicar, conceder licencias, enviar, transmitir ni distribuir este documento total ni parcialmente sin autorización expresa por parte de Oracle.

La información contenida en este documento está sujeta a cambio sin previo aviso. Si detecta cualquier problema en el documento, le agradeceremos que nos lo comunique por escrito a: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. Oracle Corporation no garantiza que este documento esté exento de errores.

Aviso sobre Restricción de Derechos

Si esta documentación se entrega al Gobierno de los EE.UU. o a cualquier entidad que la utilice en nombre del Gobierno de los EE.UU., se aplicará la siguiente advertencia:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Aviso de Marca Registrada

Oracle es una marca comercial registrada de Oracle Corporation y/o sus filiales. Todos los demás nombres pueden ser marcas comerciales de sus respectivos propietarios.

Contenido

I Introducción

- Objetivos I-2
- Agenda de la Lección I-3
- Objetivos del Curso I-4
- Agenda Sugerida del Curso I-5
- Agenda de la Lección I-7
- Esquema Human Resources (HR) que Se Utiliza en Este Curso I-8
- Información de las Cuentas de Clase I-9
- Apéndices Utilizados en Este Curso I-10
- Entornos de Desarrollo de PL/SQL I-11
- ¿Qué Es Oracle SQL Developer? I-12
- Codificación de PL/SQL en SQL*Plus I-13
- Codificación de PL/SQL en Oracle JDeveloper I-14
- Activación de la Salida de un Bloque PL/SQL I-15
- Agenda de la Lección I-16
- Documentación de Oracle 11g SQL y PL/SQL I-17
- Recursos Adicionales I-18
- Resumen I-19
- Visión General de la Práctica I: Introducción I-20

1 Creación de Procedimientos

- Objetivos 1-2
- Agenda de la Lección 1-3
- Creación de Diseños de Subprogramas Basados en Módulos 1-4
- Creación de Diseños de Subprogramas Basados en Capas 1-5
- Desarrollo Basado en Módulos con Bloques PL/SQL 1-6
- Bloques Anónimos: Visión General 1-7
- Arquitectura de Tiempo de Ejecución de PL/SQL 1-8
- ¿Qué Son los Subprogramas PL/SQL? 1-9
- Ventajas del Uso de Subprogramas PL/SQL 1-10
- Diferencias entre Bloques Anónimos y Subprogramas 1-11
- Agenda de la Lección 1-12
- ¿Qué Son los Procedimientos? 1-13
- Creación de Procedimientos: Visión General 1-14
- Creación de Procedimientos con la Sentencia SQL CREATE OR REPLACE 1-15
- Creación de Procedimientos mediante SQL Developer 1-16

Compilación de Procedimientos y Visualización de Errores de Compilación en SQL Developer	1-17
Corrección de Errores de Compilación en SQL Developer	1-18
Reglas de Nomenclatura de Estructuras PL/SQL Utilizadas en este Curso	1-19
¿Qué Son los Parámetros y los Modos de Parámetros?	1-20
Parámetros Formales y Reales	1-21
Modos de Parámetros de Procedimiento	1-22
Comparación de Modos de Parámetro	1-23
Uso del Modo de Parámetro <code>IN</code> : Ejemplo	1-24
Uso del Modo de Parámetro <code>OUT</code> : Ejemplo	1-25
Uso del Modo de Parámetro <code>IN OUT</code> : Ejemplo	1-26
Visualización de los Parámetros <code>OUT</code> : Uso de la Subrutina <code>DBMS_OUTPUT.PUT_LINE</code>	1-27
Visualización de los Parámetros <code>OUT</code> : Uso de Variables de Host SQL*Plus	1-28
Notificaciones Disponibles para la Transferencia de Parámetros Reales	1-29
Transferencia de Parámetros Reales: Creación del Procedimiento <code>add_dept</code>	1-30
Transferencia de Parámetros Reales: Ejemplos	1-31
Uso de la Opción <code>DEFAULT</code> para los Parámetros	1-32
Llamada a Procedimientos	1-34
Llamada a Procedimientos mediante SQL Developer	1-35
Agenda de la Lección	1-36
Excepciones Manejadas	1-37
Excepciones Manejadas: Ejemplo	1-38
Excepciones No Manejadas	1-39
Excepciones no Manejadas: Ejemplo	1-40
Eliminación de Procedimientos: Uso de la Sentencia SQL <code>DROP</code> o SQL Developer	1-41
Visualización de Información de Procedimientos mediante Vistas de Diccionario de Datos	1-42
Visualización de Información de Procedimientos mediante SQL Developer	1-43
Prueba	1-44
Resumen	1-45
Visión General de la Práctica 1: Creación, Compilación y Llamada de Procedimientos	1-46

2 Creación de Funciones y Depuración de Subprogramas

Objetivos	2-2
Agenda de la Lección	2-3
Visión General de las Funciones Almacenadas	2-4
Creación de Funciones	2-5
Diferencia entre Procedimientos y Funciones	2-6

Creación y Ejecución de Funciones: Visión General	2-7
Creación y Llamada a Funciones Almacenadas mediante la Sentencia CREATE FUNCTION: Ejemplo	2-8
Uso de Diferentes Métodos para Ejecutar Funciones	2-9
Creación y Compilación de Funciones mediante SQL Developer	2-11
Ejecución de Funciones mediante SQL Developer	2-12
Ventajas del Uso de Funciones Definidas por el Usuario en Sentencias SQL	2-13
Uso de Funciones en Expresiones SQL: Ejemplo	2-14
Llamada a Funciones Definidas por el Usuario en Sentencias SQL	2-15
Restricciones al Llamar a Funciones desde Expresiones SQL	2-16
Control de Efectos Secundarios al Llamar a Funciones desde Expresiones SQL	2-17
Restricciones al Llamar a Funciones desde SQL: Ejemplo	2-18
Notación con Nombre y Mixta desde SQL	2-19
Notación con Nombre y Mixta desde SQL: Ejemplo	2-20
Eliminación de Funciones: Uso de la Sentencia SQL DROP o SQL Developer	2-21
Visualización de Funciones mediante Vistas de Diccionario de Datos	2-22
Visualización de Información de Funciones mediante SQL Developer	2-23
Prueba	2-24
Práctica 2-1: Visión General	2-25
Agenda de la Lección	2-26
Depuración de Subprogramas PL/SQL mediante el Depurador de SQL Developer	2-27
Depuración de Subprogramas: Visión General	2-28
Separador de Edición de Código de Función o Procedimiento	2-29
Barra de Herramientas del Separador de Función o Procedimiento	2-30
Barra de Herramientas del Separador Debugging – Log	2-31
Separadores Adicionales	2-33
Ejemplo de Depuración de Procedimiento: Creación de un Nuevo Procedimiento <code>emp_list</code>	2-34
Ejemplo de Depuración de Procedimiento: Creación de una Nueva Función <code>get_location</code>	2-35
Definición de Puntos de División y Compilación de <code>emp_list</code> para Modo de Depuración	2-36
Compilación de la Función <code>get_location</code> para el Modo de Depuración	2-37
Depuración de <code>emp_list</code> e Introducción de Valores para el Parámetro PMAXROWS	2-38
Depuración de <code>emp_list</code> : Ejecución de Step Into (F7) en el Código	2-39
Visualización de Datos	2-41
Modificación de Variables al Depurar el Código	2-42
Depuración de <code>emp_list</code> : Ejecución de Step Over en el Código	2-43
Depuración de <code>emp_list</code> : Ejecución Step Out del Código (Mayús + F7)	2-44

Depuración de emp_list: Run to Cursor (F4) 2-45
Depuración de emp_list: Step to End of Method 2-46
Depuración de Subprogramas Remotamente: Visión General 2-47
Visión General de la Práctica 2-2: Introducción sobre el Depurador de SQL Developer 2-48
Resumen 2-49

3 Creación de Paquetes

Objetivos 3-2
Agenda de la Lección 3-3
¿Que Son los Paquetes PL/SQL? 3-4
Ventajas del Uso de Paquetes 3-5
Componentes de un Paquete PL/SQL 3-7
Visibilidad Interna y Externa de los Componentes de un Paquete 3-8
Desarrollo de Paquetes PL/SQL: Visión General 3-9
Agenda de la Lección 3-10
Creación de Especificación del Paquete: Mediante la Sentencia CREATE PACKAGE 3-11
Creación de la Especificación del Paquete: mediante SQL Developer 3-12
Creación del Cuerpo del Paquete: mediante SQL Developer 3-13
Ejemplo de Especificación de un Paquete: comm_pkg 3-14
Creación del Cuerpo del Paquete 3-15
Ejemplo del Cuerpo del Paquete: comm_pkg 3-16
Llamada a Subprogramas de Paquete: Ejemplos 3-17
Llamada a Subprogramas de Paquete: mediante SQL Developer 3-18
Creación y Uso de Paquetes sin Cuerpo 3-19
Eliminación de Paquetes: mediante la sentencia SQL DROP o SQL Developer 3-20
Visualización de Paquetes mediante Diccionario 3-21
Visualización de Paquetes mediante SQL Developer 3-22
Instrucciones para la Escritura de Paquetes 3-23
Prueba 3-24
Resumen 3-25
Visión General de la Práctica 3: Creación y Uso de Paquetes 3-26

4 Trabajar con Paquetes

Objetivos 4-2
Agenda de la Lección 4-3
Sobrecarga de Subprogramas en PL/SQL 4-4
Ejemplo de Sobrecarga de Procedimientos: Creación de la Especificación del Paquete 4-6
Ejemplo de Sobrecarga de Procedimientos: Creación del Cuerpo del Paquete 4-7

Sobrecarga y Paquete STANDARD	4-8
Referencia a Procedimiento No Válido	4-9
Uso de Declaraciones Anticipadas para Solucionar una Referencia a Procedimiento No Válido	4-10
Inicialización de Paquetes	4-11
Uso de Funciones de Paquete en SQL	4-12
Control de Efectos Secundarios de Subprogramas PL/SQL	4-13
Función de Paquete en SQL: Ejemplo	4-14
Agenda de la Lección	4-15
Estado Persistente de Paquetes	4-16
Estado Persistente de Variables de Paquetes: Ejemplo	4-18
Estado Persistente de un Cursor de Paquete: Ejemplo	4-19
Ejecución del Paquete CURS_PKG	4-21
Uso de Matrices Asociativas en Paquetes	4-22
Prueba	4-23
Resumen	4-24
Práctica 4: Visión General	4-25
5 Uso de Paquetes Proporcionados por Oracle en el Desarrollo de Aplicaciones	
Objetivos	5-2
Agenda de la Lección	5-3
Uso de Paquetes Proporcionados por Oracle	5-4
Ejemplos de Algunos Paquetes Proporcionados por Oracle	5-5
Agenda de la Lección	5-6
Funcionamiento del Paquete DBMS_OUTPUT	5-7
Uso del Paquete UTL_FILE para Interactuar con Archivos del Sistema Operativo	5-8
Algunos de los Procedimientos y Funciones de UTL_FILE	5-9
Procesamiento de Archivos con el Paquete UTL_FILE: Visión General	5-10
Uso de las Excepciones Declaradas Disponibles en el Paquete UTL_FILE	5-11
Funciones FOPEN e IS_OPEN: Ejemplo	5-12
Uso de UTL_FILE: Ejemplo	5-14
¿Qué Es el Paquete UTL_MAIL?	5-16
Configuración y Uso de UTL_MAIL: Visión General	5-17
Resumen de los Subprogramas UTL_MAIL	5-18
Instalación y Uso de UTL_MAIL	5-19
Sintaxis del Procedimiento SEND	5-20
Procedimiento SEND_ATTACH_RAW	5-21
Envío de Correo Electrónico con Anexos Binarios: Ejemplo	5-22
Procedimiento SEND_ATTACH_VARCHAR2	5-24
Envío de Correo Electrónico con Anexos de Texto: Ejemplo	5-25

Prueba 5-27

Resumen 5-28

Práctica 5: Visión General 5-29

6 Uso de SQL Dinámico

Objetivos 6-2

Agenda de la Lección 6-3

Flujo de Ejecución de SQL 6-4

Trabajar con SQL Dinámico 6-5

Uso de SQL Dinámico 6-6

SQL Dinámico Nativo (NDS) 6-7

Uso de la Sentencia EXECUTE IMMEDIATE 6-8

Métodos Disponibles para Usar NDS 6-9

SQL Dinámico con una Sentencia DDL: Ejemplos 6-11

SQL Dinámico con Sentencias DML 6-12

SQL Dinámico con una Consulta de una Sola Fila: Ejemplo 6-13

Ejecución Dinámica de un Bloque PL/SQL Anónimo 6-14

Uso de SQL Dinámico Nativo para Compilar Código PL/SQL 6-15

Agenda de la Lección 6-16

Uso del Paquete DBMS_SQL 6-17

Uso de los Subprogramas del Paquete DBMS_SQL 6-18

Uso de DBMS_SQL con una Sentencia DML: Supresión de Filas 6-20

Uso de DBMS_SQL con una Sentencia DML con Parámetros 6-21

Prueba 6-22

Resumen 6-23

Visión General de la Práctica 6: Uso de SQL Dinámico Nativo 6-24

7 Consideraciones de Diseño para Código PL/SQL

Objetivos 7-2

Agenda de la Lección 7-3

Estandarización de Constantes y Excepciones 7-4

Estandarización de Excepciones 7-5

Estandarización del Manejo de Excepciones 7-6

Estandarización de Constantes 7-7

Subprogramas Locales 7-8

Derechos del Responsable de la Definición frente a Derechos del Invocador 7-9

Especificación de Derechos del Invocador: Definición de AUTHID en

CURRENT_USER 7-10

Transacciones Autónomas 7-11

Funciones de las Transacciones Autónomas 7-12

Uso de Transacciones Autónomas: Ejemplo	7-13
Agenda de la Lección	7-15
Uso de la Indicación NOCOPY	7-16
Efectos de la Indicación NOCOPY	7-17
¿Cuándo Ignora el Compilador PL/SQL la Indicación NOCOPY?	7-18
Uso de la Indicación PARALLEL_ENABLE	7-19
Uso de la Caché de Resultados de Funciones PL/SQL entre Sesiones	7-20
Activación del Almacenamiento en Caché de Resultados para una Función	7-21
Declaración y Definición de una Función de Resultados Almacenados en Caché:	
Ejemplo	7-22
Uso de la Cláusula DETERMINISTIC con Funciones	7-24
Agenda de la Lección	7-25
Uso de la Cláusula RETURNING	7-26
Uso de Enlaces en Bloque	7-27
Enlaces en Bloque: Sintaxis y Palabras Clave	7-28
Enlace en Bloque FORALL: Ejemplo	7-30
Uso de BULK COLLECT INTO con Consultas	7-32
Uso de BULK COLLECT INTO con Cursor	7-33
Uso de BULK COLLECT INTO con una Cláusula RETURNING	7-34
Uso de Enlaces en Bloque en Recopilaciones Dispersas	7-35
Uso de Enlaces en Bloque con Matrices de Índice	7-38
Prueba	7-39
Resumen	7-40
Práctica 7: Visión General	7-41

8 Creación de Disparadores

Objetivos	8-2
¿Qué Son los Disparadores?	8-3
Definición de Disparadores	8-4
Tipos de Evento de Disparador	8-5
Disparadores de Aplicación y de Base de Datos	8-6
Supuestos de Aplicación de Negocio para la Implantación de Disparadores	8-7
Tipos de Disparadores Disponibles	8-8
Tipos de Evento y Cuerpo del Disparador	8-9
Creación de Disparadores de DML mediante la Sentencia CREATE TRIGGER	8-10
Especificación del Arranque del Disparador (Temporización)	8-11
Disparadores de Nivel de Sentencia frente a Disparadores de Nivel de Fila	8-12
Creación de Disparadores de DML mediante SQL Developer	8-13
Secuencia de Arranque de Disparadores: Manipulación de una Sola Fila	8-14
Secuencia de Arranque de Disparadores: Manipulación de Varias Filas	8-15

Ejemplo de Creación de un Disparador de Sentencia DML: SECURE_EMP	8-16
Prueba del Disparador SECURE_EMP	8-17
Uso de Predicados Condicionales	8-18
Creación de un Disparador de Fila DML	8-19
Uso de los Cualificadores OLD y NEW	8-20
Uso de los Cualificadores OLD y NEW: Ejemplo	8-21
Uso de la Cláusula WHEN para Arrancar un Disparador de Fila Basado en una Condición	8-23
Resumen del Modelo de Ejecución de Disparadores	8-24
Implantación de una Restricción de Integridad con un Disparador After	8-25
Disparadores INSTEAD OF	8-26
Creación de un Disparador INSTEAD OF: Ejemplo	8-27
Creación de un Disparador INSTEAD OF para Realizar DML en Vistas Complejas	8-28
Estado de un Disparador	8-30
Creación de un Disparador Desactivado	8-31
Gestión de Disparadores mediante las Sentencias SQL ALTER y DROP	8-32
Gestión de Disparadores mediante SQL Developer	8-33
Prueba de Disparadores	8-34
Visualización de Información de Disparador	8-35
Uso de USER_TRIGGERS	8-36
Prueba	8-37
Resumen	8-38
Visión General de la Práctica 8: Creación de Disparadores de Sentencia y de Fila	8-39

9 Creación de Disparadores Compuestos, de DDL y de Eventos de Base de Datos

Objetivos	9-2
¿Qué son los Disparadores Compuestos?	9-3
Trabajar con Disparadores Compuestos	9-4
Ventajas del Uso de un Disparador Compuesto	9-5
Secciones de Punto de Temporización de un Disparador Compuesto de Tabla	9-6
Estructura de los Disparadores Compuestos para Tablas	9-7
Estructura de los Disparadores Compuestos para Vistas	9-8
Restricciones de Disparadores Compuestos	9-9
Restricciones de Disparadores en Tablas Mutantes	9-10
Tabla Mutante: Ejemplo	9-11
Uso de un Disparador Compuesto para Resolver el Error en la Tabla Mutante	9-13
Creación de Disparadores en Sentencias DDL	9-15
Creación de Disparadores de Eventos de Base de Datos	9-16
Creación de Disparadores en Eventos de Sistema	9-17
Disparadores LOGON y LOGOFF: Ejemplo	9-18

Sentencias CALL en Disparadores	9-19
Ventajas de los Disparadores de Eventos de Base de Datos	9-20
Privilegios del Sistema Necesarios para Gestionar Disparadores	9-21
Instrucciones para el Diseño de Disparadores	9-22
Prueba	9-23
Resumen	9-24
Práctica 9: Visión General	9-25
10 Uso del Compilador PL/SQL	
Objetivos	10-2
Agenda de la Lección	10-3
Parámetros de Inicialización para la Compilación PL/SQL	10-4
Uso de los Parámetros de Inicialización para la Compilación PL/SQL	10-5
Configuración del Compilador	10-7
Visualización de los Parámetros de Inicialización de PL/SQL Displaying the PL/SQL Initialization Parameters	10-8
Visualización y Definición de los Parámetros de Inicialización de PL/SQL	10-9
Cambio de los Parámetros de Inicialización de PL/SQL: Ejemplo	10-10
Agenda de la Lección	10-11
Visión General de las Advertencias de Tiempo de Compilación PL/SQL para Subprogramas	10-12
Ventajas de las Advertencias del Compilador	10-14
Categorías de Mensajes de Advertencia de Tiempo de Compilación PL/SQL	10-15
Definición de Niveles de Mensajes de Advertencia	10-16
Definición de Niveles de Advertencia del Compilador: mediante PLSQL_WARNINGS	10-17
Definición de Niveles de Advertencia del Compilador: mediante PLSQL_WARNINGS, Ejemplos	10-18
Definición de Niveles de Advertencia del Compilador: mediante PLSQL_WARNINGS en SQL Developer	10-19
Visualización del Valor Actual de PLSQL_WARNINGS	10-20
Visualización de Advertencias del Compilador: mediante SQL Developer, SQL*Plus o las Vistas del Diccionario de Datos	10-21
Mensajes de Advertencia de SQL*Plus: Ejemplo	10-22
Instrucciones para el Uso de PLSQL_WARNINGS	10-23
Agenda de la Lección	10-24
Definición de Niveles de Advertencia del Compilador: mediante el Paquete DBMS_WARNING	10-25
Uso de los Subprogramas del Paquete DBMS_WARNING	10-27
Procedimientos DBMS_WARNING: Sintaxis, Parámetros y Valores Permitidos	10-28

Procedimientos DBMS_WARNING: Ejemplo	10-29
Funciones DBMS_WARNING: Sintaxis, Parámetros y Valores Permitidos	10-30
Funciones DBMS_WARNING: Ejemplo	10-31
Uso de DBMS_WARNING: Ejemplo	10-32
Uso del Mensaje de Advertencia PLW_06009	10-34
Advertencia PLW_06009: Ejemplo	10-35
Prueba	10-36
Resumen	10-37
Práctica 10: Visión General	10-38

11 Gestión del Código PL/SQL

Objetivos	11-2
Agenda de la Lección	11-3
¿Qué es la Compilación Condicional?	11-4
Funcionamiento de la Compilación Condicional	11-5
Uso de Directivas de Selección	11-6
Uso de Directivas de Consulta Predefinidas y Definidas por el Usuario	11-7
Parámetro PLSQL_CCFLAGS y Directiva de Consulta	11-8
Visualización del Valor del Parámetro de Inicialización PLSQL_CCFLAGS	11-9
Parámetro PLSQL_CCFLAGS y Directiva de Consulta: Ejemplo	11-10
Uso de Directivas de Error de Compilación Condicional para Emitir Errores Definidos por el Usuario	11-11
Uso de Expresiones Estáticas con la Compilación Condicional	11-12
Paquete DBMS_DB_VERSION: Constantes Booleanas	11-13
Constantes del Paquete DBMS_DB_VERSION	11-14
Uso de la Compilación Condicional con Versiones de la Base de Datos: Ejemplo	11-15
Uso de Procedimientos DBMS_PREPROCESSOR para Imprimir o Recuperar Texto de Origen	11-17
Agenda de la Lección	11-18
¿Qué es la Ocultación?	11-19
Ventajas de la Ocultación	11-20
Novedades de la Ocultación Dinámica desde Oracle 10g	11-21
Código PL/SQL No Oculto: Ejemplo	11-22
Código PL/SQL Oculto: Ejemplo	11-23
Ocultación Dinámica: Ejemplo	11-24
Utilidad Encapsuladora PL/SQL	11-25
Ejecución de la Utilidad Encapsuladora	11-26
Resultados del Encapsulamiento	11-27
Instrucciones para el Encapsulamiento	11-28
Paquete DBMS_DDL frente a utilidad Wrap	11-29

Prueba 11-30
Resumen. 11-31
Práctica 11: Visión General 11-32

12 Gestión de Dependencias

Objetivos 12-2
Visión General de Dependencias de Objetos de Esquema 12-3
Dependencias 12-4
Dependencias Locales Directas 12-5
Consulta de Dependencias Directas de Objeto: mediante la Vista
 USER_DEPENDENCIES 12-6
Consulta del Estado de un Objeto 12-7
Invalidación de Objetos Dependientes 12-8
Cambio del Objeto de Esquema que Invalida Algunos Dependientes: Ejemplo 12-9
Visualización de Dependencias Directas e Indirectas 12-11
Visualización de Dependencias mediante la Vista DEPTREE 12-12
Metadatos de Dependencia más Precisos en Oracle Database 11g 12-13
Gestión de Dependencias Detalladas 12-14
Gestión de Dependencias Detalladas: Ejemplo 1 12-15
Gestión de Dependencias Detalladas: Ejemplo 2 12-17
Cambios en las Dependencias de Sinónimos 12-18
Mantenimiento de Vistas y Unidades de Programa PL/SQL Válidas 12-19
Otro Supuesto de Dependencias Locales 12-20
Instrucciones para Reducir la Invalidación 12-21
Revalidación de Objetos 12-22
Dependencias Remotas 12-23
Conceptos de Dependencias Remotas 12-24
Definición del Parámetro REMOTE_DEPENDENCIES_MODE 12-25
El Procedimiento Remoto B Se Compila a las 8:00 a.m. 12-26
El Procedimiento Local A Se Compila a las 9:00 a.m. 12-27
Ejecución del Procedimiento A 12-28
Procedimiento Remoto B Recompilado a las 11:00 a.m. 12-29
Ejecución del Procedimiento A 12-30
Modo de Firma 12-31
Recompilación de una Unidad de Programa PL/SQL 12-32
Recompilación Incorrecta 12-33
Recompilación Correcta 12-34
Recompilación de Procedimientos 12-35
Paquetes y Dependencias: El Subprograma Hace Referencia al Paquete 12-36
Paquetes y Dependencias: El Subprograma del Paquete Hace Referencia al
 Procedimiento 12-37

Prueba	12-38
Resumen	12-39
Visión General de la Práctica 12: Gestión de Dependencias en el Esquema	12-40

Apéndice A: Prácticas y Soluciones

Apéndice AP: Adicional: Prácticas y Soluciones

Apéndice B: Descripciones de las Tablas

Apéndice C: Uso de SQL Developer

Objetivos	C-2
¿Qué Es Oracle SQL Developer?	C-3
Especificaciones de SQL Developer	C-4
Interfaz de SQL Developer 1.5	C-5
Creación de una Conexión a la Base Datos	C-7
Exploración de Objetos de Bases de Datos	C-10
Visualización de la Estructura de la Tabla	C-11
Examen de archivos	C-12
Creación de un Objeto de Esquema	C-13
Creación de una Nueva Tabla: Ejemplo	C-14
Uso de la Hoja de Trabajo de SQL	C-15
Ejecución de Sentencias SQL	C-18
Guardado de scripts SQL	C-19
Ejecución de Archivos de Script Guardados: Método 1	C-20
Ejecución de Archivos de Script Guardados: Método 2	C-21
Formato del Código SQL	C-22
Uso de Fragmentos	C-23
Uso de Fragmentos: Ejemplo	C-24
Depuración de Procedimientos y Funciones	C-25
Informes de Bases de Datos	C-26
Creación de un Informe Definido por el Usuario	C-27
Motores de Búsqueda y Herramientas Externas	C-28
Definición de Preferencias	C-29
Restablecimiento del Diseño de SQL Developer	C-30
Resumen	C-31

Apéndice D: Uso de SQL*Plus

Objetivos	D-2
Interacción de SQL y SQL*Plus	D-3
Sentencias SQL frente a Comandos SQL*Plus	D-4

Visión General de SQL*Plus	D-5
Conexión a SQL*Plus	D-6
Visualización de la Estructura de la Tabla	D-7
Comandos de Edición SQL*Plus	D-9
Uso de LIST, n y APPEND	D-11
Uso del Comando CHANGE	D-12
Comandos de Archivos SQL*Plus	D-13
Uso de los Comandos SAVE y START	D-14
Comando SERVEROUTPUT	D-15
Uso del Comando SQL*Plus SPOOL	D-16
Uso del Comando AUTOTRACE	D-17
Resumen	D-18

Apéndice E: Uso de JDeveloper

Objetivos	E-2
Oracle JDeveloper	E-3
Database Navigator	E-4
Creación de una Conexión	E-5
Exploración de Objetos de Bases de Datos	E-6
Ejecución de Sentencias SQL	E-7
Creación de Unidades de Programa	E-8
Compilación	E-9
Ejecución de una Unidad de Programa	E-10
Borrado de una Unidad de Programa	E-11
Ventana Structure	E-12
Ventana Editor	E-13
Application Navigator	E-14
Despliegue de Procedimientos Java Almacenados	E-15
Publicación de Java en PL/SQL	E-16
¿Cómo Puedo Obtener Más Información sobre JDeveloper 11g?	E-17
Resumen	E-18

Apéndice F: Revisión de PL/SQL

Objetivos	F-2
Estructura en Bloque para Bloques PL/SQL Anónimos	F-3
Declaración de Variables PL/SQL	F-4
Declaración de Variables con el Atributo %TYPE: Ejemplos	F-5
Creación de un Registro PL/SQL	F-6
Atributo %ROWTYPE: Ejemplos	F-7
Creación de una Tabla PL/SQL	F-8

Sentencias SELECT en PL/SQL: Ejemplo F-9
Inserción de Datos: Ejemplo F-10
Actualización de Datos: Ejemplo F-11
Supresión de Datos: Ejemplo F-12
Control de Transacciones con las Sentencias COMMIT y ROLLBACK F-13
Sentencias IF, THEN y ELSIF: Ejemplo F-14
Bucle Básico: Ejemplo F-15
Bucle FOR: Ejemplo F-16
Bucle WHILE: Ejemplo F-17
Atributos de Cursor Implícito SQL F-18
Control de Cursos Explícitos F-19
Control de Cursos Explícitos: Declaración del Cursor F-20
Control de Cursos Explícitos: Apertura del Cursor F-21
Control de Cursos Explícitos: Recuperación de Datos del Cursor F-22
Control de Cursos Explícitos: Cierre del Cursor F-23
Atributos de Cursor Explícito F-24
Bucles FOR de Cursor : Ejemplo F-25
Cláusula FOR UPDATE: Ejemplo F-26
Cláusula WHERE CURRENT OF: Ejemplo F-27
Detección de Errores Predefinidos del Servidor de Oracle F-28
Detección de Errores Predefinidos del Servidor de Oracle: Ejemplo F-29
Error No Predefinido F-30
Excepciones Definidas por el Usuario: Ejemplo F-31
Procedimiento RAISE_APPLICATION_ERROR F-32
Resumen F-34

Apéndice G: Estudios para Implantación de Disparadores

Objetivos G-2
Control de la Seguridad en el Servidor G-3
Control de la Seguridad con un Disparador de Base de Datos G-4
Forzado de Integridad de Datos en el Servidor G-5
Protección de la Integridad de los Datos con un Disparador G-6
Forzado de la Integridad Referencial en el Servidor G-7
Protección de la Integridad Referencial con un Disparador G-8
Replicación de Tablas en el Servidor G-9
Replicación de Tablas con un Disparador G-10
Cálculo de Datos Derivados en el Servidor G-11
Cálculo de Valores Derivados con un Disparador G-12
Registro de Eventos con un Disparador G-13
Resumen G-15

Apéndice H: Uso de los Paquetes DBMS_SCHEDULER y HTP

Objetivos	H-2
Generación de Páginas Web con el Paquete HTP	H-3
Uso de los Procedimientos de Paquete HTP	H-4
Creación de un Archivo HTML con SQL*Plus	H-5
Paquete DBMS_SCHEDULER	H-6
Creación de un Trabajo	H-8
Creación de un Trabajo con Parámetros en Línea	H-9
Creación de un Trabajo Utilizando un Programa	H-10
Creación de un Trabajo para un Programa con Argumentos	H-11
Creación de un Trabajo Utilizando una Planificación	H-12
Definición del Intervalo de Repetición para un Trabajo	H-13
Creación de un Trabajo Utilizando un Programa y una Planificación con Nombre	H-14
Gestión de Trabajos	H-15
Vistas del Diccionario de Datos	H-16
Resumen	H-17

I

Introducción

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

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

- Describir los objetivos del curso
- Identificar los entornos disponibles que se pueden utilizar en este curso
- Describir las tablas y el esquema de base de datos que se utilizan en el curso
- Enumerar la documentación y los recursos disponibles



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos de la Lección

PL/SQL soporta muchas construcciones de programa. En esta lección, revisará las unidades de programa como bloques anónimos y se le presentarán 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.

Agenda de la Lección

- Objetivos del Curso y Agenda del Curso
- Esquema y apéndices utilizados en este curso y entornos de desarrollo PL/SQL disponibles en este curso
- Documentación y recursos adicionales de Oracle 11g

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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 y de archivo
- Identificar distintas técnicas que afectan a las consideraciones de diseño del código PL/SQL
- Utilizar el compilador PL/SQL, gestionar código PL/SQL y gestionar dependencias

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 Sugerida del Curso

Primer día:

- Lección 1: Introducción
- Lección 1: Creación de Procedimientos
- Lección 2: Creación de Funciones
- Lección 3: Creación de Paquetes
- Lección 4: Trabajar con Paquetes

Segundo día:

- Lección 5: Uso de Paquetes Proporcionados por Oracle en el Desarrollo de Aplicaciones
- Lección 6: Uso de SQL Dinámico
- Lección 7: Consideraciones de Diseño para Código PL/SQL
- Lección 8: Creación de Disparadores

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Agenda Sugerida del Curso

Tercer día:

- Lección 9: Creación de Disparadores Compuestos, de DDL y de Eventos de Base de Datos
- Lección 10: Uso del Compilador PL/SQL
- Lección 11: Gestión del Código PL/SQL
- Lección 12: Gestión de Dependencias

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

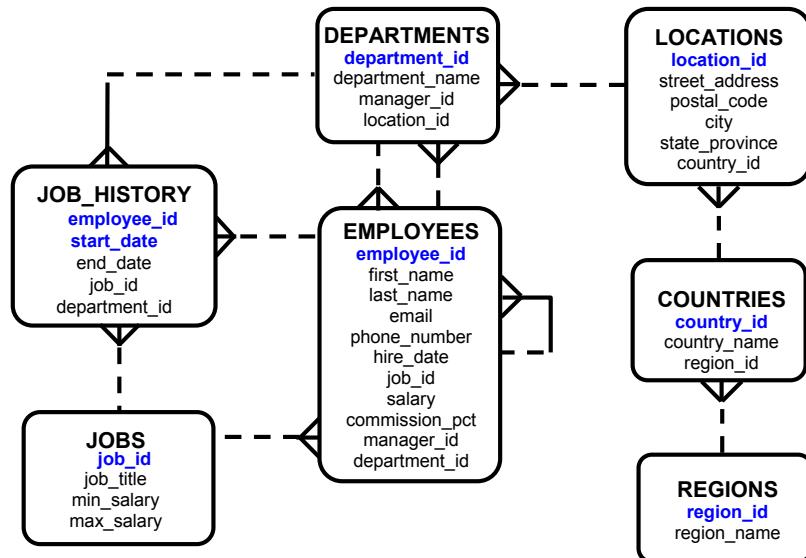
Agenda de la Lección

- Objetivos del curso y agenda del curso
- Esquema y apéndices utilizados en este curso y entornos de desarrollo PL/SQL disponibles en este curso
- Documentación y recursos adicionales de Oracle 11g

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Esquema Human Resources (HR) que Se Utiliza en Este Curso



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Descripción del Esquema Human Resources (HR)

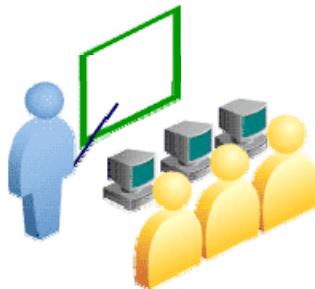
El esquema Human Resources (HR) forma parte de los esquemas de ejemplo de Oracle que se pueden instalar en una base de datos Oracle. Las sesiones prácticas de este curso utilizan datos del esquema HR.

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 superior 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.

Información de las Cuentas de Clase

- Los identificadores de cuenta HR clonados están ya configurados.
- Sus identificadores de cuenta son ora61 u ora62.
- La contraseña coincide con su identificador de cuenta.
- A cada máquina se le asigna una cuenta.
- El instructor tiene un identificador distinto.



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Nota: utilice el identificador de cuenta ora61 u ora62.

Apéndices Utilizados en Este Curso

- Apéndice A: Prácticas y Soluciones
- Apéndice AP: Prácticas Adicionales y Soluciones
- Apéndice B: Descripciones de las Tablas
- Apéndice C: Uso de SQL Developer
- Apéndice D: Uso de SQL*Plus
- Apéndice E: Revisión de JDeveloper
- Apéndice F: Revisión de PL/SQL
- Apéndice G: Estudios para Implantación de Disparadores
- Apéndice H: Uso de los Paquetes DBMS_SCHEDULER y HTP

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Entornos de Desarrollo de PL/SQL

La configuración de este curso proporciona las siguientes herramientas para desarrollar código PL/SQL:

- Oracle SQL Developer (se utiliza en este curso)
- Oracle SQL*Plus
- Oracle JDeveloper IDE



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Entornos de Desarrollo de PL/SQL

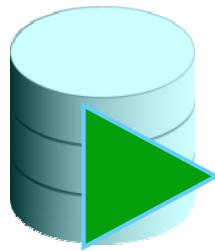
Existen numerosas herramientas que proporcionan entornos de desarrollo de código PL/SQL. Oracle proporciona varias herramientas que se pueden utilizar para escribir código PL/SQL. Algunas de las herramientas de desarrollo disponibles en este curso son:

- **Oracle SQL Developer:** herramienta gráfica
- **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 pantallas que se presentan en las notas del curso se han generado a partir de la salida del entorno SQL Developer.

¿Qué Es Oracle SQL Developer?

- Oracle SQL Developer es una herramienta gráfica gratuita que mejora la productividad y simplifica las tareas de desarrollo de la base de datos.
- Puede conectarse a cualquier esquema de base de datos de destino de Oracle mediante la autenticación estándar de Oracle Database.
- En este curso se utiliza SQL Developer.
- El Apéndice C contiene detalles sobre el uso de SQL Developer.



SQL Developer

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

¿Qué Es Oracle SQL Developer?

Oracle SQL Developer es una herramienta gráfica gratuita diseñada para mejorar la productividad y simplificar el desarrollo de las tareas diarias de la base de datos. Con sólo unos clics, puede crear y mantener fácilmente los procedimientos almacenados, probar sentencias SQL y visualizar los planes del optimizador.

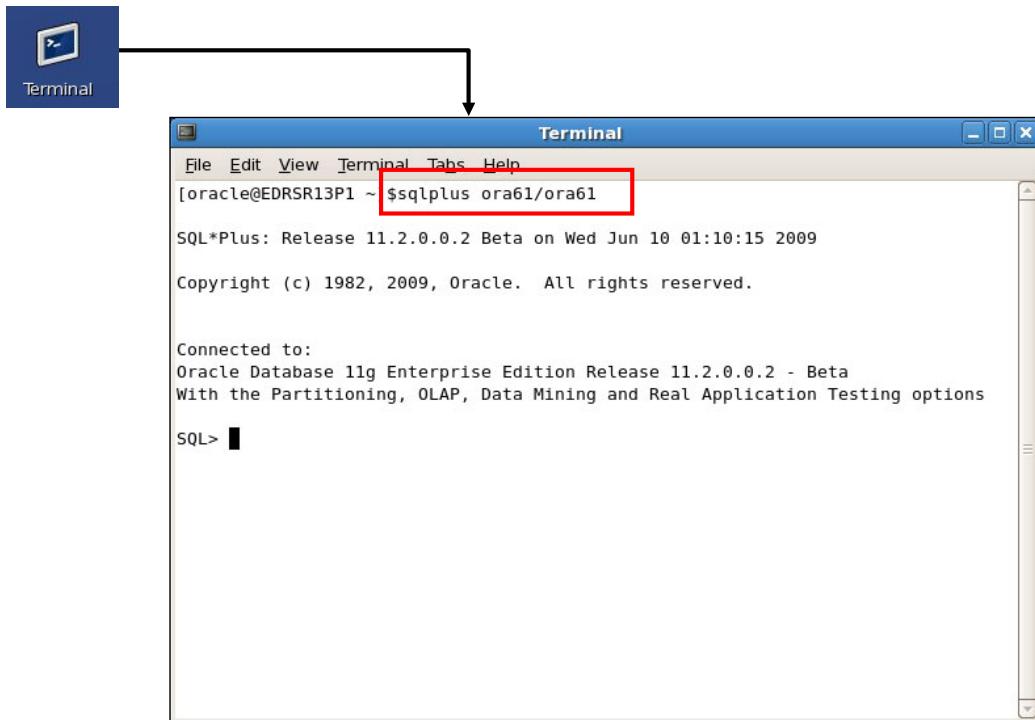
SQL Developer, la herramienta visual para el desarrollo de la base de datos, simplifica las siguientes tareas:

- Exploración y gestión de objetos de la base de datos
- Ejecución de sentencias y scripts SQL
- Edición y depuración de sentencias PL/SQL
- Creación de informes

Puede conectarse a cualquier esquema de base de datos de destino de Oracle mediante la autenticación estándar de Oracle Database. Una vez conectado, puede realizar operaciones en los objetos de la base de datos.

Nota: el Apéndice C de este curso proporciona una introducción al uso de la interfaz de SQL Developer. Consulte el apéndice ahora para obtener información sobre la creación de una conexión de base de datos, así como sobre la interacción con los datos mediante SQL y PL/SQL, entre otros temas.

Codificación de PL/SQL en SQL*Plus



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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:

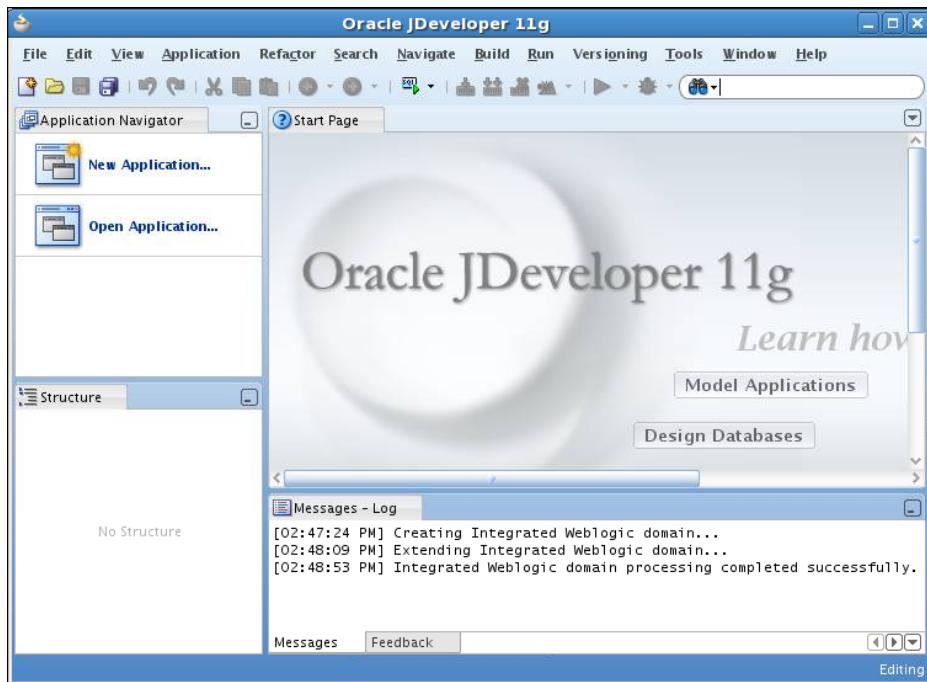
- Se incluye con la base de datos.
- Se instala en un sistema de cliente y servidor de bases de datos.
- Es accesible desde un ícono o la línea de comandos.

Al codificar subprogramas PL/SQL con SQL*Plus, no olvide 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.

Nota: para obtener más información sobre cómo utilizar SQL*Plus, consulte el Apéndice D.

Codificación de PL/SQL en Oracle JDeveloper



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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.
- 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.
- Para obtener más información sobre cómo utilizar JDeveloper, consulte el Apéndice E.

Activación de la Salida de un Bloque PL/SQL

1. Para activar la salida en SQL Developer, ejecute el siguiente comando antes de ejecutar el bloque PL/SQL:

```
SET SERVEROUTPUT ON;
```

2. Utilice el paquete predefinido de Oracle DBMS_OUTPUT y su procedimiento para mostrar la salida, como se indica a continuación:

- DBMS_OUTPUT.PUT_LINE

```
DBMS_OUTPUT.PUT_LINE('The First Name of the
Employee is ' || v_fname);
. . .
```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Activación de la Salida de un Bloque PL/SQL

En el ejemplo mostrado en la diapositiva anterior, se ha almacenado un valor en la variable v_fname. Sin embargo, el valor no se ha impreso.

PL/SQL no tiene la funcionalidad de entrada o salida incorporada. Por tanto, necesita utilizar paquetes predefinidos de Oracle para la entrada y salida. Para generar salida, debe llevar a cabo lo siguiente:

1. Ejecute el siguiente comando SQL:

```
SET SERVEROUTPUT ON
```

Nota: para activar la salida en SQL*Plus, debe emitir de forma explícita el comando SET SERVEROUTPUT ON.

2. En el bloque PL/SQL, utilice el procedimiento PUT_LINE del paquete DBMS_OUTPUT para mostrar la salida. Transfiera el valor que se tiene que imprimir como argumento a este procedimiento (como se muestra en la diapositiva). A continuación, el procedimiento generará el argumento.

Agenda de la Lección

- Objetivos del curso y agenda del curso
- Esquema y apéndices utilizados en este curso y entornos de desarrollo PL/SQL disponibles en este curso
- Documentación y recursos adicionales de Oracle 11g

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Documentación de Oracle 11g SQL y PL/SQL

- *Oracle Database New Features Guide 11g Release 2 (11.2)*
- *Oracle Database Advanced Application Developer's Guide 11g Release 2 (11.2)*
- *Oracle Database PL/SQL Language Reference 11g Release 2 (11.2)*
- *Oracle Database Reference 11g Release 2 (11.2)*
- *Oracle Database SQL Language Reference 11g Release 2 (11.2)*
- *Oracle Database Concepts 11g Release 2 (11.2)*
- *Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (11.2)*
- *Oracle Database SQL Developer User's Guide Release 1.5*

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Documentación de Oracle 11g SQL y PL/SQL

Acceda a <http://www.oracle.com/pls/db111/homepage> y haga clic en el enlace Master Book List del marco izquierdo.

Recursos Adicionales

Para obtener más información sobre las nuevas funciones de Oracle 11g SQL y PL/SQL, consulte los siguientes recursos:

- Oracle Database 11g: New Features eStudies
- Oracle by Example (OBE) series: Oracle Database 11g:
 - http://www.oracle.com/technology/obe/11gr1_db/admin/11gr1db.html
- What's New in PL/SQL in Oracle Database 11g en Oracle Technology Network (OTN):
 - http://www.oracle.com/technology/tech/pl_sql/



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Describir los objetivos del curso
- Identificar los entornos disponibles que se pueden utilizar en este curso
- Describir las tablas y el esquema de base de datos que se utilizan en el curso
- Enumerar la documentación y los recursos disponibles



Copyright © 2010, Oracle y/o sus filiales. 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 nivel medio o de cliente para Oracle Forms y Oracle Reports y proporciona un motor de tiempo de ejecución de PL/SQL dentro de la base de datos Oracle. Los procedimientos y las 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.

Visión General de la Práctica I: Introducción

En esta práctica se abordan los siguientes temas:

- Revisión de los recursos disponibles de SQL Developer
- Inicio de SQL Developer y creación de una nueva conexión de base de datos y exploración de las tablas de esquema
- Definición de algunas preferencias de SQL Developer
- Ejecución de sentencias SQL y un bloque PL/SQL anónimo mediante la hoja de trabajo de SQL
- Acceso y marcado de la documentación de Oracle Database 11g y de otros sitios web útiles

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica I: Visión General

En esta práctica, utilizará SQL Developer para ejecutar sentencias SQL con el fin de examinar los datos del esquema. También creará un bloque anónimo simple. Opcionalmente, puede intentar crear y ejecutar el código PL/SQL en SQL*Plus.

Nota: todas las prácticas escritas utilizan SQL Developer como entorno de desarrollo. Si bien se recomienda utilizar SQL Developer, también puede utilizar los entornos SQL*Plus o JDeveloper disponibles en este curso.

1

Creación de Procedimientos

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

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

- Identificar las ventajas del diseño de subprogramas basados en módulos y capas
- Crear y llamar a procedimientos
- Utilizar parámetros formales y reales
- Utilizar la notación posicional, con nombre o mixta para la transferencia de parámetros
- Identificar los modos de transferencia de parámetros disponibles
- Manejar excepciones en procedimientos
- Eliminar un procedimiento
- Mostrar información de los procedimientos



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos de la Lección

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 a un procedimiento con parámetros OUT.

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

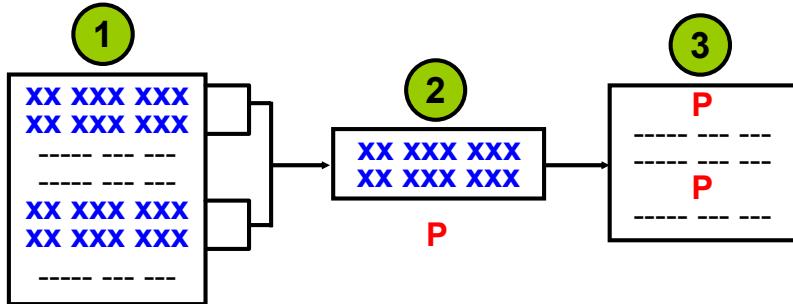
Agenda de la Lección

- Uso de diseños de subprogramas basados en módulos y capas e identificación de ventajas de los subprogramas
- Trabajar con procedimientos:
 - Creación y llamada a procedimientos
 - Identificación de los modos de transferencia de parámetros disponibles
 - Uso de parámetros formales y reales
 - Uso de notación posicional, con nombre y mixta
- Manejo de excepciones en procedimientos, eliminación de procedimientos y visualización de información de los procedimientos

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de Diseños de Subprogramas Basados en Módulos



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.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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.

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 Oracle Database, que puede reutilizar sentencias SQL analizadas para un mejor uso de los recursos del servidor.

Creación de Diseños de Subprogramas Basados en Capas

Crear capas de subprograma para la aplicación.

- Capa de subprograma de acceso a datos con lógica SQL
- Capa de subprograma de lógica de negocio, que puede o no utilizar la capa de acceso a datos



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de Diseños de Subprogramas Basados en Capas

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 implanten las reglas de procesamiento de negocio, que se pueden o no llamar en rutinas de capa de acceso a datos.

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



Copyright © 2010, Oracle y/o sus filiales. 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 solo lugar y 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 titulado *Oracle: Conceptos Fundamentales de PL/SQL*.

Bloques Anónimos: Visión General

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)
```

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Bloques Anónimos: Visión General

Los bloques anónimos se utilizan normalmente para:

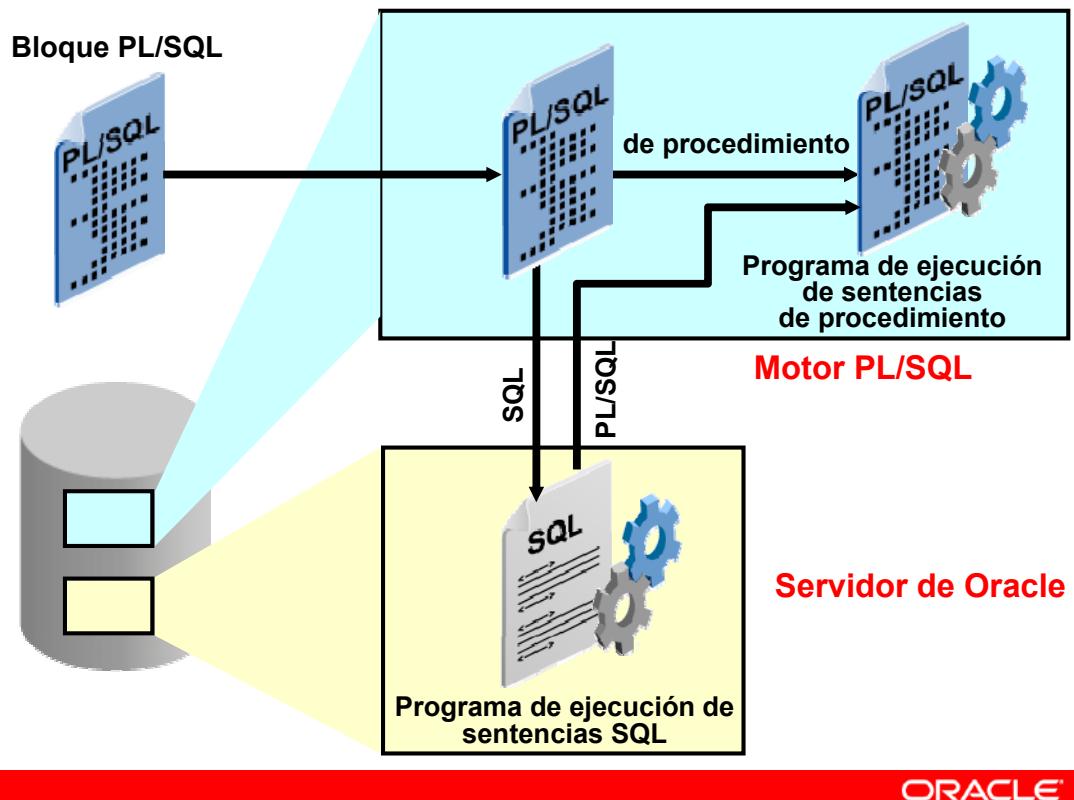
- Escribir código del disparador para componentes de Oracle Forms
- Iniciar llamadas a procedimientos, funciones y construcciones de paquetes
- Aislar el manejo de excepciones dentro de un bloque de código
- Anidar dentro de otros bloques PL/SQL para gestionar el control del flujo de código

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*.

Arquitectura de Tiempo de Ejecución de PL/SQL



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Arquitectura de Tiempo de Ejecución de PL/SQL

El diagrama muestra la ejecución de un bloque PL/SQL por parte de un motor PL/SQL. El motor PL/SQL reside en:

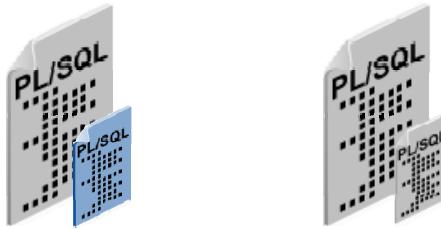
- La base de datos Oracle para ejecutar subprogramas almacenados
- El cliente Oracle Forms al ejecutar aplicaciones de cliente/servidor o en Oracle Application Server al utilizar Oracle Forms Services para ejecutar Forms en la web

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 entorno SQL también puede llamar al entorno PL/SQL, por ejemplo, al utilizar una función en una sentencia SELECT.

El motor PL/SQL es una máquina virtual que reside en memoria y procesa las instrucciones de código m de PL/SQL. Cuando el motor 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 motor 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 motor PL/SQL de Oracle Forms se ejecuta en el cliente para la implantación del cliente o servidor y en el servidor de aplicaciones para la implantación de Forms Services. En cualquier caso, las sentencias SQL se envían normalmente por una red a un servidor de Oracle para su procesamiento.

¿Qué Son los Subprogramas PL/SQL?

- Un subprograma PL/SQL es un bloque PL/SQL con nombre que se puede llamar con un juego de parámetros.
- Puede declarar y definir un subprograma en un bloque PL/SQL o en otro subprograma.
- Un subprograma consta de una especificación y un cuerpo.
- Un subprograma puede ser un procedimiento o una función.
- Normalmente, se utiliza un procedimiento para realizar una acción y una función para calcular y devolver un valor.
- Los subprogramas se pueden agrupar en paquetes PL/SQL.



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

¿Qué Son los Subprogramas PL/SQL?

Un subprograma PL/SQL es un bloque PL/SQL con nombre que se puede llamar con un juego de parámetros. Puede declarar y definir un subprograma en un bloque PL/SQL o en otro subprograma.

Partes de los Subprogramas

Un subprograma consta de una especificación y un cuerpo. Para declarar un programa, debe proporcionar la especificación, que incluye descripciones de los parámetros. Para definir un subprograma, debe proporcionar tanto la especificación como el cuerpo. Puede declarar un subprograma en primer lugar y, posteriormente, definirlo en el mismo bloque o subprograma o declararlo y definirlo al mismo tiempo.

Tipos de Subprograma

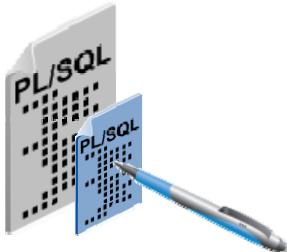
PL/SQL tiene dos tipos de subprogramas: procedimientos y funciones. Normalmente, se utiliza un procedimiento para realizar una acción y una función para calcular y devolver un valor.

Un procedimiento y una función tienen la misma estructura, excepto que sólo la función tiene elementos adicionales como la cláusula RETURN o la sentencia RETURN.

La cláusula RETURN especifica el tipo de dato del valor de retorno (necesario). Una sentencia RETURN especifica el valor de retorno (necesario). Las funciones se tratan más detalladamente en la siguiente lección titulada “Creación de Funciones y Depuración de Subprogramas”.

Los subprogramas se pueden agrupar en paquetes PL/SQL, lo que aumenta cada vez más la capacidad de reutilización y mantenimiento del código. Los paquetes se tratan en las lecciones sobre paquetes: lecciones 3 y 4.

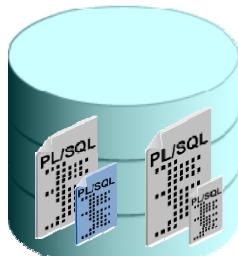
Ventajas del Uso de Subprogramas PL/SQL



Mantenimiento sencillo



Claridad de código mejorada



Subprogramas: procedimientos y funciones almacenados



Integridad y seguridad de datos mejoradas



Rendimiento mejorado

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 se ejecutan por defecto con 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.
- 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, reduciendo de esta forma la necesidad de comentarios y mejorando la claridad del código.

Diferencias entre Bloques Anónimos y Subprogramas

Bloques Anónimos	Subprogramas
Bloques PL/SQL sin nombre	Bloques PL/SQL con nombre
Se compilan en cada ocasión	Se compilan sólo una vez
No se almacenan en la base de datos	Se almacenan en la base de datos
Otras aplicaciones no los pueden llamar	Tienen nombre y, por tanto, otras aplicaciones los pueden llamar
No devuelven valores	Los subprogramas denominados funciones deben devolver valores
No pueden utilizar parámetros	Pueden utilizar parámetros

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Diferencias entre Bloques Anónimos y Subprogramas

En la tabla de la diapositiva no sólo se muestran las diferencias entre los bloques anónimos y los subprogramas, sino que también se resaltan las ventajas generales de los subprogramas.

Los bloques anónimos no son objetos de base de datos persistente. Se compilan y se ejecutan sólo una vez. No se almacenan en la base de datos para volver a utilizarlos. Si desea volver a utilizarlos, debe volver a ejecutar el script que crea el bloque anónimo, que produce la recompilación y la ejecución.

Los procedimientos y las funciones se compilan y se almacenan en la base de datos en forma compilada.

Sólo se vuelven a compilar cuando se modifican. Puesto que se almacenan en la base de datos, cualquier aplicación puede utilizar estos subprogramas basándose en los permisos necesarios. La aplicación que realiza la llamada puede transferir los parámetros a los procedimientos si el procedimiento está diseñado para aceptar parámetros. Asimismo, una aplicación que realiza la llamada puede recuperar un valor si llama a una función o a un procedimiento.

Agenda de la Lección

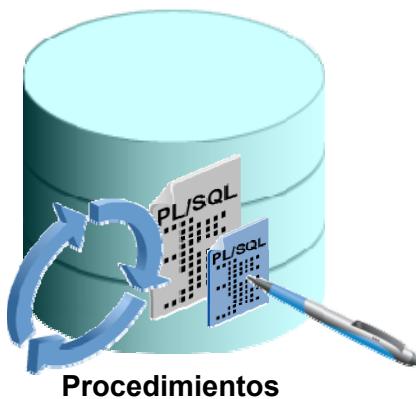
- Uso de diseños de subprogramas basados en módulos y capas e identificación de ventajas de los subprogramas
- Trabajar con procedimientos:
 - Creación y llamada a procedimientos
 - Identificación de los modos de transferencia de parámetros disponibles
 - Uso de parámetros formales y reales
 - Uso de notación posicional, con nombre y mixta
- Manejo de excepciones en procedimientos, eliminación de procedimientos y visualización de información de los procedimientos

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

¿Qué Son los Procedimientos?

- Son un tipo de subprograma que realiza una acción
- Se pueden almacenar en la base de datos como objetos de esquema
- Fomentan la capacidad de reutilización y mantenimiento



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

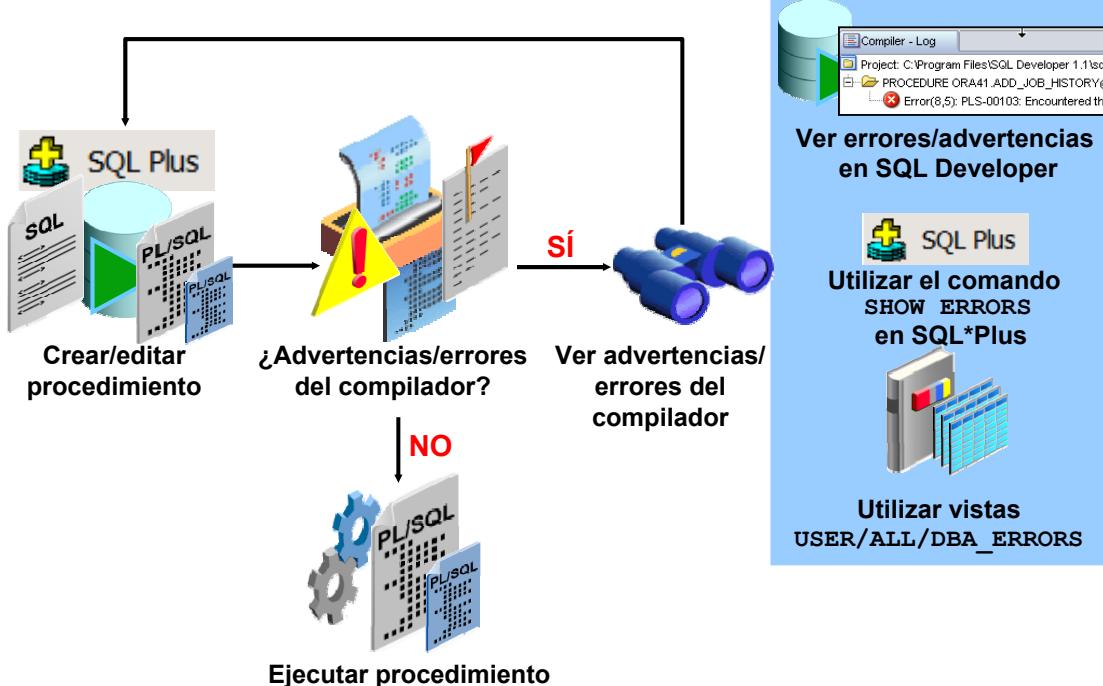
Definición de un Procedimiento

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. Cuando se validan, se pueden utilizar en cualquier número de aplicaciones. Si los requisitos cambian, sólo es necesario actualizar el procedimiento.

Creación de Procedimientos: Visión General



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de Procedimientos: Visión General

Para desarrollar un procedimiento mediante una herramienta como SQL Developer, realice los siguientes pasos:

1. Cree el procedimiento mediante el árbol Object Navigator de SQL Developer en el área SQL Worksheet.
2. Compile el procedimiento. El procedimiento se crea en la base de datos y se compila. La sentencia `CREATE PROCEDURE` crea y almacena el código fuente y el valor de *m-code* compilado en la base de datos. Para compilar el procedimiento, haga clic con el botón derecho en el nombre del procedimiento en el árbol Object Navigator y, a continuación, haga clic en **Compile**.
3. Si se produce un error de compilación, el valor de *m-code* no se almacena y debe editar el código fuente para realizar las correcciones. No se puede llamar a un procedimiento que contenga errores de compilación. Puede ver los errores de compilación en SQL Developer, SQL*Plus o en las vistas del diccionario de datos adecuadas, como se muestra en la diapositiva.
4. Después de terminar la compilación correctamente, ejecute el procedimiento para realizar la acción deseada. Puede ejecutar el procedimiento mediante SQL Developer o utilizar el comando `EXECUTE` en SQL*Plus.

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 (mediante `DROP`) y, a continuación, ejecute la sentencia `CREATE PROCEDURE`.

Creación de Procedimientos con la Sentencia SQL

CREATE OR REPLACE

- Utilice la cláusula CREATE para crear un procedimiento autónomo que se almacene en Oracle Database.
- Utilice la opción OR REPLACE para sustituir un procedimiento existente.

```
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

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de Procedimientos con la Sentencia SQL CREATE OR REPLACE

Puede utilizar la sentencia SQL CREATE PROCEDURE para crear procedimientos autónomos que se almacenen en Oracle Database. Un procedimiento es similar a un programa en miniatura: realiza una acción concreta. Especifique el nombre del procedimiento, sus parámetros, sus variables locales y el bloque BEGIN-END que contiene su código y maneja cualquier excepción.

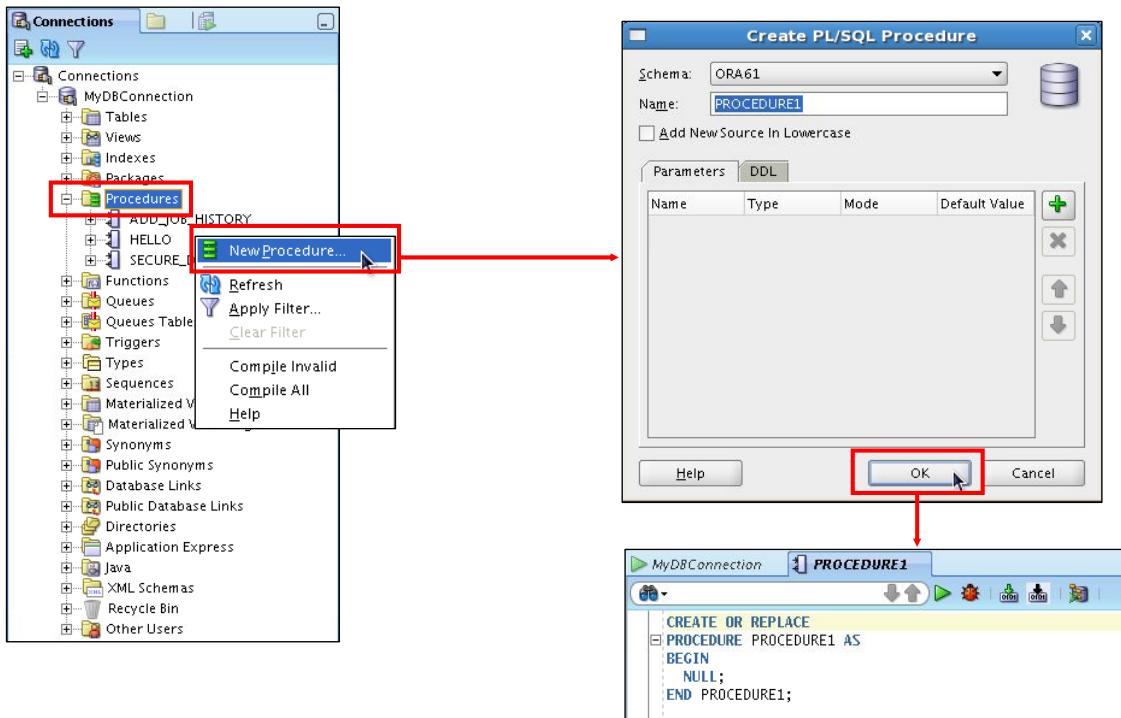
- Los bloques PL/SQL empiezan por BEGIN, precedido opcionalmente por la declaración de variables locales. Los bloques PL/SQL terminan en END o END *procedure_name*.
- La opción REPLACE indica que si el procedimiento existe, se borra y se sustituye por la nueva versión creada por la sentencia. La opción REPLACE no borra ninguno de los privilegios asociados al procedimiento.

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 (de enlace) 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.

Creación de Procedimientos mediante SQL Developer



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

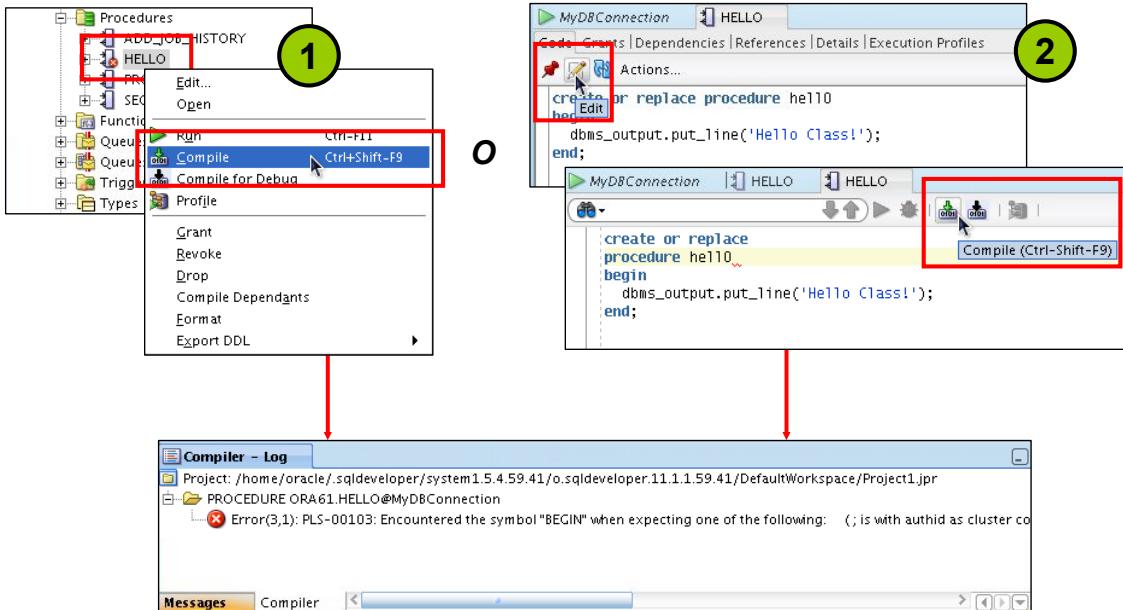
Creación de Procedimientos mediante SQL Developer

1. Haga clic con el botón derecho en el nodo **Procedures** en la página con separadores **Connections**.
2. Seleccione **New Procedure** en el menú de acceso directo. Se muestra el cuadro de diálogo **Create PL/SQL Procedure**. Especifique la información para el nuevo procedimiento y, a continuación, haga clic en OK para crear el subprograma y que se muestre en la ventana Editor, donde puede introducir los detalles.

Los componentes del cuadro de diálogo **Create PL/SQL Procedure** son los siguientes:

- **Schema:** esquema de base de datos en el que se creará el subprograma PL/SQL.
- **Name:** nombre del subprograma que debe ser único en un esquema.
- **Add New Source in Lowercase:** si se selecciona esta opción, el nuevo texto aparece en minúsculas independientemente de si lo introduce en mayúsculas o minúsculas. Esta opción sólo afecta a la apariencia del código, porque PL/SQL no distingue entre mayúsculas y minúsculas en su ejecución.
- **Separador Parameters:** para agregar un parámetro, haga clic en el icono Add (+). Para cada parámetro del procedimiento que se vaya a crear, especifique el nombre de parámetro, el tipo de dato, el modo y, opcionalmente, el valor por defecto. Utilice el icono Remove (X) y los iconos de flecha para suprimir y mover un parámetro hacia arriba o hacia abajo en la lista, respectivamente.
- **Separador DDL:** este separador contiene una visualización de sólo lectura de una sentencia SQL, que refleja la definición actual del subprograma.

Compilación de Procedimientos y Visualización de Errores de Compilación en SQL Developer



ORACLE

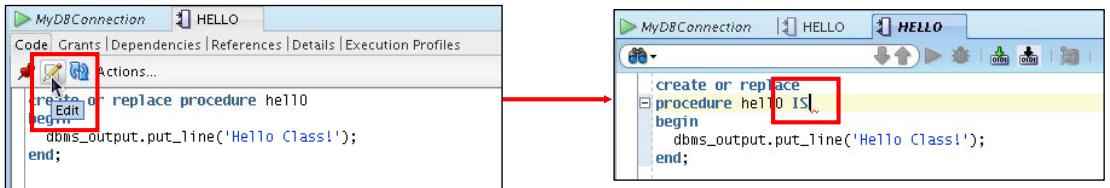
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Compilación de Procedimientos y Visualización de Errores de Compilación en SQL Developer

Puede compilar procedimientos mediante uno de los dos métodos siguientes:

- Acceda al nodo Procedures del árbol Object Navigator. Haga clic con el botón derecho en el nombre del procedimiento y seleccione Compile en el menú de acceso directo. Para ver los mensajes de compilación, visualice el subseparador Messages en el separador **Compiler – Log**.
- Edite el procedimiento mediante el ícono Edit de la barra de herramientas de código del procedimiento. Realice las ediciones necesarias y, a continuación, haga clic en el ícono Compile de la barra de herramientas de código. Para ver los mensajes de compilación, visualice el subseparador Messages en el separador **Compiler – Log**.

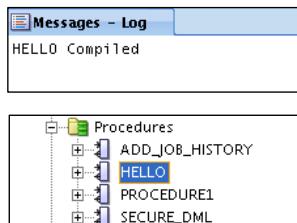
Corrección de Errores de Compilación en SQL Developer



1. Editar procedimiento

The screenshot shows the same SQL Developer interface. The code editor now displays the error message 'ORA-01017: invalid character' at the beginning of the line where 'IS' was typed. A red box highlights the error message.

2. Corregir el error (agregar la palabra clave IS)



4. Recompilación correcta



3. Recompilar procedimiento

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Corrección de Errores de Compilación en SQL Developer

1. Edite el procedimiento mediante el ícono Edit de la barra de herramientas de código del procedimiento. Se abre un nuevo separador de código del procedimiento en modo de lectura/escritura.
2. Realice las correcciones necesarias.
3. Haga clic en el ícono Compile de la barra de herramientas de código.
4. Para ver los mensajes de compilación, visualice el subseparador Messages en el separador **Compiler – Log**. Además, si el procedimiento se compila correctamente, se elimina la X roja del nombre del procedimiento del árbol Object Navigator.

Reglas de Nomenclatura de Estructuras PL/SQL Utilizadas en este Curso

Estructura PL/SQL	Regla	Ejemplo
Variable	v_variable_name	v_rate
Constante	c_constant_name	c_rate
Parámetro de subprograma	p_parameter_name	p_id
Variable de enlace (host)	b_bind_name	b_salary
Cursor	cur_cursor_name	cur_emp
Registro	rec_record_name	rec_emp
Tipo	type_name_type	ename_table_type
Excepción	e_exception_name	e_products_invalid
Manejo de archivos	f_file_handle_name	f_file

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Reglas de Nomenclatura de Estructuras PL/SQL Utilizadas en este Curso

En la tabla de la diapositiva se muestran algunos ejemplos de las reglas de nomenclatura de las estructuras PL/SQL utilizadas en este curso.

¿Qué Son los Parámetros y los Modos de Parámetros?

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

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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
- Parámetros (o argumentos) reales: valores literales, variables o expresiones utilizadas en la lista de parámetros del subprograma de llamada

```
-- Procedure definition, Formal parameters
CREATE PROCEDURE raise_sal(p_id NUMBER, p_sal NUMBER) IS
BEGIN
    . . .
END raise_sal;

-- Procedure calling, Actual parameters (arguments)
v_emp_id := 100;
raise_sal(v_emp_id, 2000)
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 de la variable `p_id` y `p_sal` 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 de llamada. En el segundo ejemplo, se realiza una llamada a `raise_sal`, donde la variable `v_emp_id` proporciona el valor de parámetro real para el parámetro formal `p_id`, mientras que 2000 se proporciona como valor real para `p_sal`.

Parámetros reales:

- Se asocian a parámetros formales durante la llamada al subprograma.
- Pueden ser también expresiones, como en el siguiente ejemplo:
`raise_sal(v_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.

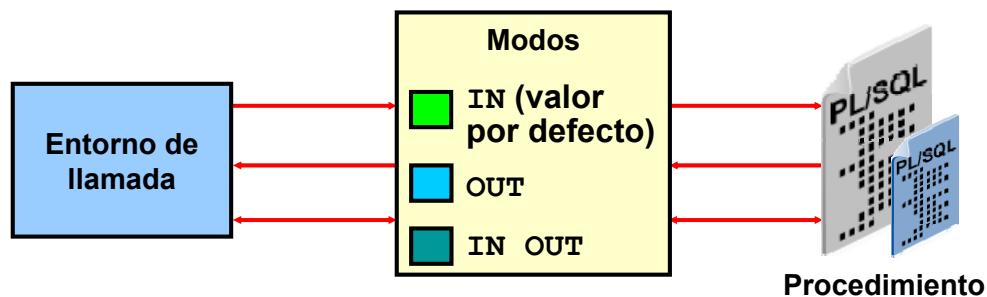
Nota: a los parámetros reales también se les denomina *argumentos reales*.

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 proc_name(param_name [mode] datatype)
...

```



ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 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 especifica ningún modo con una declaración de parámetro, se considera que el parámetro es 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.

Comparación de Modos de Parámetro

IN	OUT	IN OUT
Modo por defecto	Se debe especificar	Se debe especificar
El valor se transfiere al subprograma	El valor se devuelve al entorno de llamada	Valor transferido al subprograma; valor devuelto 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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Comparación de Modos de Parámetro

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 le 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.

El parámetro **OUT** o **IN OUT** debe 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.

Uso del Modo de Parámetro IN: Ejemplo

```

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

```



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

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Parámetros IN: Ejemplo

En el ejemplo de la diapositiva se muestra un procedimiento con dos parámetros IN. Al ejecutar el primer ejemplo de la diapositiva se crea un procedimiento `raise_salary` en la base de datos. El segundo ejemplo de la diapositiva llama a `raise_salary` y proporciona el primer valor de parámetro 176 para el identificador de empleado y un aumento del salario del 10 por ciento para el segundo valor de parámetro.

Para llamar a un procedimiento mediante la función SQL Worksheet de SQL Developer o mediante SQL*Plus, utilice el siguiente comando `EXECUTE`, mostrado en el segundo ejemplo de código de la diapositiva.

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 del Modo de Parámetro OUT: Ejemplo

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

```
SET SERVEROUTPUT ON
DECLARE
  v_emp_name employees.last_name%TYPE;
  v_emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(171, v_emp_name, v_emp_sal);
  DBMS_OUTPUT.PUT_LINE(v_emp_name||' earns '|| 
    to_char(v_emp_sal, '$999,999.00'));
END;
/
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Parámetros OUT: Ejemplo

En el ejemplo de la diapositiva, 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 el 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 segundo cuadro de código de la diapositiva. El procedimiento acepta un valor de identificador de empleado a través del parámetro `p_id`. Las variables `v_emp_name` y `v_emp_salary` se llenan con la información recuperada de la consulta en los dos parámetros OUT correspondientes. A continuación, se muestra el resultado de la ejecución del código del segundo ejemplo de código de la diapositiva. `v_emp_name` contiene el valor Smith y `v_emp_salary` contiene el valor 7400:

The screenshot shows the Oracle SQL Developer interface with the following details:

- Toolbar:** Results, Script Output, Explain, Autotrace, DBMS Output, OWA Output.
- Buttons:** Undo, Redo, Save, Print.
- Output Window:**

```
anonymous block completed
Smith earns $7,400.00
```

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.

Uso del Modo de Parámetro IN OUT: Ejemplo

Entorno de Llamada

p_phone_no (antes de la llamada) p_phone_no (después de la llamada)

'8006330575'

'(800) 633-0575'

```
CREATE OR REPLACE PROCEDURE format_phone
  (p_phone_no IN OUT VARCHAR2) IS
BEGIN
  p_phone_no := '(' || SUBSTR(p_phone_no,1,3) ||
                 ')' || SUBSTR(p_phone_no,4,3) ||
                 '-' || SUBSTR(p_phone_no,7);
END format_phone;
/
```

```
anonymous block completed
b_phone_no
-----
8006330575
anonymous block completed
b_phone_no
-----
(800) 633-0575
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 puede devolver el valor original sin cambiar o un nuevo valor definido dentro del procedimiento.

Nota: un parámetro IN OUT funciona como variable inicializada.

En el ejemplo de la diapositiva se 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.

En el siguiente código se utiliza la variable de host b_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 b_phone_no. La salida del siguiente código se muestra en la diapositiva anterior:

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

Visualización de los Parámetros OUT: Uso de la Subrutina DBMS_OUTPUT.PUT_LINE

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

```
SET SERVEROUTPUT ON

DECLARE
    v_emp_name employees.last_name%TYPE;
    v_emp_sal   employees.salary%TYPE;
BEGIN
    query_emp(171, v_emp_name, v_emp_sal);
    DBMS_OUTPUT.PUT_LINE('Name: ' || v_emp_name);
    DBMS_OUTPUT.PUT_LINE('Salary: ' || v_emp_sal);
END;
```

```
anonymous block completed
Name: Smith
Salary: 7400
```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización de los Parámetros OUT: Uso de la Subrutina DBMS_OUTPUT

En el ejemplo de la diapositiva se ilustra la forma de visualizar los valores devueltos de los parámetros OUT en SQL*Plus o en la hoja de trabajo de SQL Developer.

Puede utilizar 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. El valor de SET SERVEROUTPUT debe ser ON.

Visualización de los Parámetros OUT: Uso de Variables de Host SQL*Plus

1. Utilizar variables de host SQL*Plus.
2. Ejecutar `QUERY_EMP` mediante variables de host.
3. Imprimir las variables de host.

```
VARIABLE b_name    VARCHAR2 (25)
VARIABLE b_sal     NUMBER
EXECUTE query_emp(171, :b_name, :b_sal)
PRINT b_name b_sal
```

```
anonymous block completed
b_name
-----
Smith

b_sal
-----
7400
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización de los Parámetros OUT: Uso de Variables de Host SQL*Plus

En el ejemplo de la diapositiva se demuestra cómo utilizar variables de host SQL*Plus creadas mediante el comando `VARIABLE`. Las variables SQL*Plus son externas al bloque PL/SQL y se conocen como variables de enlace 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 SQL*Plus seguido del nombre de la variable SQL*Plus (sin los dos puntos, ya que no se trata de un comando PL/SQL o un contexto).

Nota: para obtener más información sobre el comando `VARIABLE`, consulte `SQL*Plus Command Reference` (`Referencia de Comandos SQL*Plus`).

Notificaciones Disponibles para la Transferencia de Parámetros Reales

- Al llamar a un subprograma, puede escribir los parámetros reales mediante las siguientes notaciones:
 - 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
 - Mixta: muestra algunos de los parámetros reales como posicionales y otros como con nombre
- Antes de Oracle Database 11g, sólo se soporta la notación posicional en llamadas de SQL
- A partir de Oracle Database 11g, las notaciones con nombre y mixtas se pueden utilizar para especificar argumentos en llamadas a subrutinas PL/SQL desde sentencias SQL

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

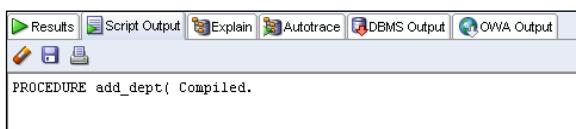
Sintaxis de Transferencia de Parámetros

Al llamar a un subprograma, puede escribir los parámetros reales mediante las siguientes notaciones:

- **Posicional:** muestra los valores de parámetros reales en el mismo orden en el que se declaran los parámetros formales. Esta notación es compacta, pero si especifica los parámetros (especialmente literales) en orden incorrecto, el error puede ser difícil de detectar. Debe cambiar el código si cambia la lista de parámetros del procedimiento.
- **Con nombre:** 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: $=>$. No importa el orden de los parámetros. Esta notación es más detallada, pero facilita la lectura y el mantenimiento de código. En ocasiones, puede evitar cambiar el código cuando cambia la lista de parámetros del procedimiento, por ejemplo, si se reordenan los parámetros o se agregan nuevos parámetros adicionales.
- **Mixta:** muestra los primeros valores del parámetro por posición y el resto mediante la sintaxis especial del método con nombre. Puede utilizar esta notación para llamar a procedimientos con parámetros necesarios, seguidos de parámetros opcionales.

Transferencia de Parámetros Reales: Creación del Procedimiento add_dept

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



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Transferencia de Parámetros: Ejemplos

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

Transferencia de Parámetros Reales: Ejemplos

```
-- Passing parameters using the positional notation.
EXECUTE add_dept ('TRAINING', 2500)
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	TRAINING		2500

1 rows selected

```
-- Passing parameters using the named notation.
EXECUTE add_dept (p_loc=>2400, p_name=>'EDUCATION')
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
290	EDUCATION		2400

1 rows selected

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Transferencia de Parámetros Reales: Ejemplos

La transferencia de parámetros reales por posición se muestra en la primera llamada para ejecutar add_dept en el primer ejemplo de código de la diapositiva. El primer parámetro real proporciona el valor TRAINING para el parámetro formal name. El segundo valor de parámetro real de 2500 se asigna por posición al parámetro formal loc.

La transferencia de parámetros mediante la notación con nombre se muestra en el segundo ejemplo de código de la diapositiva. Se hace referencia al parámetro real loc, que se declara como el segundo parámetro formal, según el nombre en la llamada, donde se asocia al valor real de 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 los Parámetros

- Define valores por defecto para parámetros
- Proporciona flexibilidad combinando la sintaxis de transferencia de parámetros posicional y con nombre

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

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

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de la Opción DEFAULT para los Parámetros

Puede asignar un valor por defecto a un parámetro IN de la siguiente forma:

- Operador de asignación (:=), como se muestra para el parámetro name de la diapositiva
- Opción DEFAULT, como se muestra para el parámetro p_loc en la diapositiva

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 los parámetros OUT e IN OUT.

En el segundo cuadro de código de la diapositiva se muestran tres formas de llamar al procedimiento add_dept:

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

Uso de la Opción DEFAULT para los Parámetros (continuación)

A continuación, se muestra el resultado del segundo ejemplo de código de la diapositiva anterior:

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	TRAINING	2500	
290	EDUCATION	2400	
300	Unknown	1700	
310	ADVERTISING	1200	
320	Unknown	1200	
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		1700
140	Control And Credit		1700
150	Shareholder Services		1700
160	Benefits		1700
170	Manufacturing		1700
180	Construction		1700
190	Contracting		1700
200	Operations		1700
210	IT Support		1700
220	NOC		1700
230	IT Helpdesk		1700
240	Government Sales		1700
250	Retail Sales		1700
260	Recruiting		1700
270	Payroll		1700

32 rows selected

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(p_name=>'new dept', 'new location')
```

```
Error starting at line 1 in command:
EXECUTE add_dept(p_name=>'new dept', 'new location')
Error report:
ORA-06550: line 1, column 36:
PLS-00312: a positional parameter association may not follow a named association
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
06550. 00000 - "line %s, column %s:\n%s"
*Cause: Usually a PL/SQL compilation error.
*Action:
```

Llamada a Procedimientos

- Puede llamar a procedimientos mediante bloques anónimos, otro procedimiento o paquetes.
- El usuario debe ser propietario del procedimiento o tener el privilegio EXECUTE.

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

PROCEDURE process_employees Compiled.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Llamada a Procedimientos

Puede llamar a los procedimientos mediante:

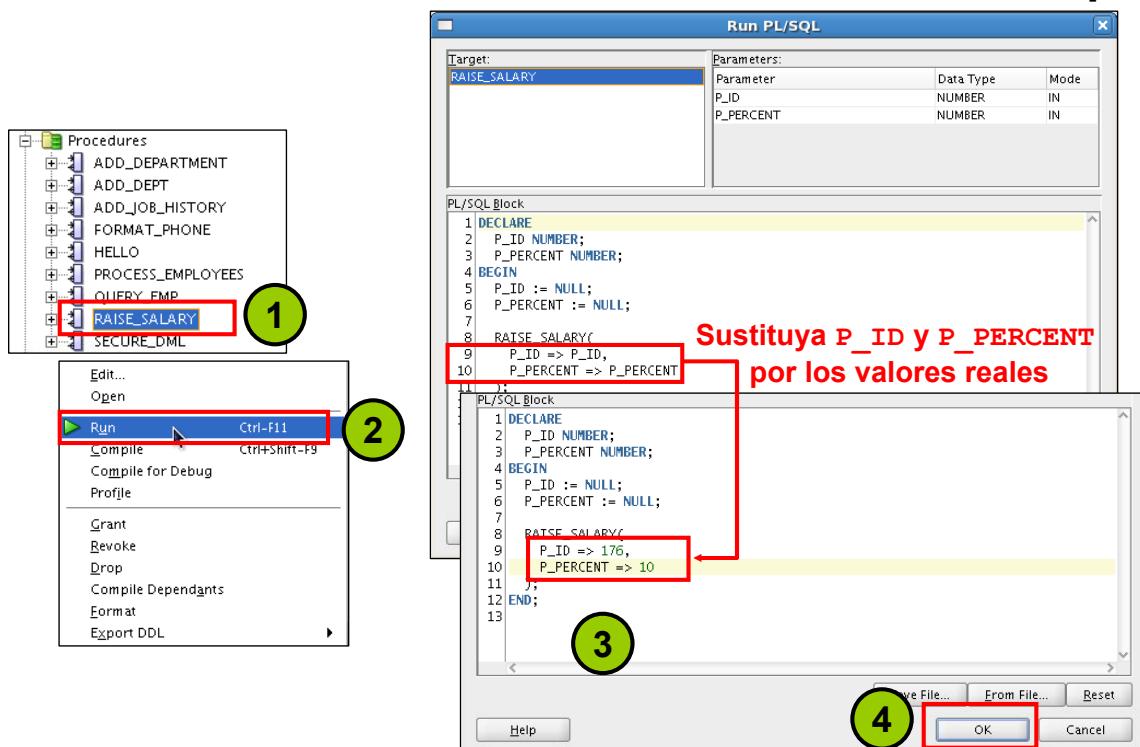
- Bloques anónimos
- Otro procedimiento o subprograma PL/SQL

En algunos ejemplos de las páginas anteriores se ilustra cómo utilizar bloques anónimos (o el comando EXECUTE en SQL Developer o SQL*Plus).

En el ejemplo de la diapositiva se muestra cómo llamar a un procedimiento desde otro procedimiento almacenado. El procedimiento almacenado PROCESS_EMPLOYEES 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.

Nota: el usuario debe ser propietario del procedimiento o tener el privilegio EXECUTE.

Llamada a Procedimientos mediante SQL Developer



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Llamada a Procedimientos mediante SQL Developer

En el ejemplo de la diapositiva, se llama al procedimiento `raise_salary` para incrementar el salario actual del empleado 176 (8.600 dólares) en un 10 por ciento, de la siguiente forma:

1. Haga clic con el botón derecho en el nombre del procedimiento en el nodo **Procedures** y, a continuación, haga clic en **Run**. Se muestra el cuadro de diálogo **Run PL/SQL**.
2. En la sección **PL/SQL Block**, cambie las especificaciones de los parámetros formales **IN** e **IN/OUT** mostrados que aparecen después del operador de asociación, “=>” a los valores *reales* que desea utilizar para ejecutar o depurar la función o el procedimiento. Por ejemplo, para incrementar el salario actual del empleado 176 de 8.600 en un 10 por ciento, puede llamar al procedimiento `raise_salary`, como se muestra en la diapositiva. Proporcione los valores para los parámetros de entrada `ID` y `PERCENT` que se especifican como 176 y 10, respectivamente. Para ello, cambie el valor `ID => ID` mostrado por `ID => 176` y `PERCENT => PERCENT` por `PERCENT => 10`.
3. Haga clic en **OK**. SQL Developer ejecuta el procedimiento. El salario actualizado de 9.460 se muestra a continuación:

Results	Script Output	Explain	Autotrace	DBMS Output	OWA Output
Results:					
	EMPLOYEE_ID	SALARY			
1	176	9460			

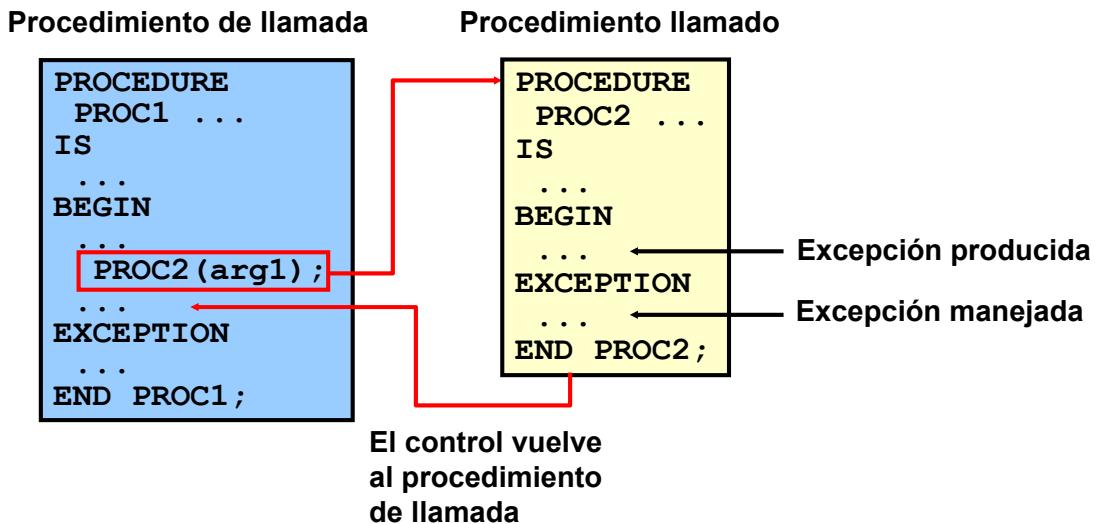
Agenda de la Lección

- Uso de diseños de subprogramas basados en módulos y capas e identificación de ventajas de los subprogramas
- Trabajar con procedimientos:
 - Creación y llamada a procedimientos
 - Identificación de los modos de transferencia de parámetros disponibles
 - Uso de parámetros formales y reales
 - Uso de notación posicional, con nombre y mixta
- Manejo de excepciones en procedimientos, eliminación de procedimientos y visualización de información de los procedimientos

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Excepciones Manejadas



ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
BEGIN
    INSERT INTO DEPARTMENTS (department_id,
        department_name, manager_id, location_id)
    VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
    DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_name);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Err: adding dept: '|| p_name);
END;

```

```

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

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Excepciones Manejadas: Ejemplo

Los dos procedimientos de la diapositiva son los siguientes:

- El procedimiento `add_department` crea un nuevo registro de departamento mediante la asignación de un nuevo número de departamento de una secuencia de Oracle y define los valores de columna `department_name`, `manager_id` y `location_id` mediante los parámetros `p_name`, `p_mgr` y `p_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:

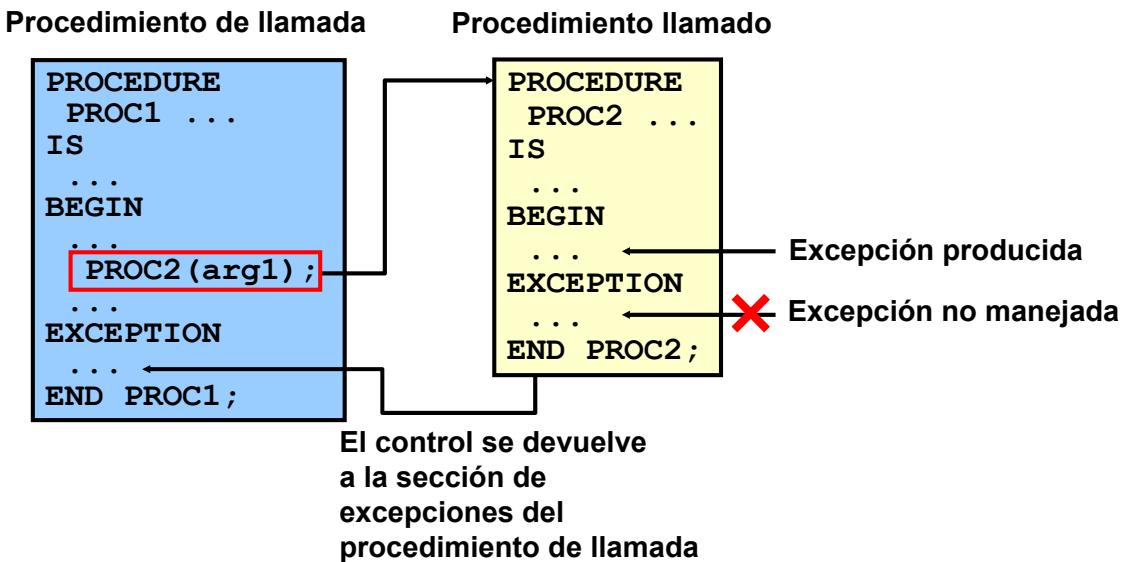
```

Added Dept: Media
Err: adding dept: Editing
Added Dept: Advertising

```

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

Excepciones No Manejadas



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Excepciones No Manejadas

Tal como se ha descrito anteriormente, 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 la hay.

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

```
SET SERVEROUTPUT ON
CREATE PROCEDURE add_department_noex(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
BEGIN
    INSERT INTO DEPARTMENTS (department_id,
        department_name, manager_id, location_id)
    VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
    DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_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); XX
END;
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Excepciones no Manejadas: Ejemplo

En el ejemplo de código de la diapositiva se 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:

```
Error starting at line 8 in command:
EXECUTE create_departments_noex
Error report:
ORA-02291: integrity constraint (ORA62.DEPT_MGR_FK) violated - parent key not found
ORA-06512: at "ORA62.ADD_DEPARTMENT_NOEX", line 4
ORA-06512: at "ORA62.CREATE_DEPARTMENTS_NOEX", line 4
ORA-06512: at line 1
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause: A foreign key value has no matching primary key value.
*Action: Delete the foreign key or add a matching primary key.
```

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: Uso de la Sentencia SQL DROP o SQL Developer

- Uso de la sentencia DROP:

```
DROP PROCEDURE raise_salary;
```

- Uso de SQL Developer:



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Eliminación de Procedimientos: Uso de la Sentencia SQL DROP o SQL Developer

Cuando un procedimiento almacenado ya no es necesario, puede utilizar la sentencia SQL `DROP PROCEDURE` seguida del nombre del procedimiento para eliminarlo, de la siguiente forma:

```
DROP PROCEDURE procedure_name
```

También puede utilizar SQL Developer para borrar un procedimiento almacenado, de la siguiente forma:

- Haga clic con el botón derecho en el nombre del procedimiento en el nodo **Procedures** y, a continuación, haga clic en **Drop**. Aparece el cuadro de diálogo **Drop**.
- Haga clic en **Apply** para borrar el procedimiento.

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.
- Puede que tenga que refrescar el nodo **Procedures** antes de poder ver los resultados de la operación de borrado. Para refrescar el nodo **Procedures**, haga clic con el botón derecho en el nombre del procedimiento en el nodo **Procedures** y, a continuación, haga clic en **Refresh**.

Visualización de Información de Procedimientos mediante Vistas de Diccionario de Datos

```
DESCRIBE user_source
```

Name	Null	Type
NAME		VARCHAR2(30)
TYPE		VARCHAR2(12)
LINE		NUMBER
TEXT		VARCHAR2(4000)

4 rows selected

```
SELECT text
  FROM user_source
 WHERE name = 'ADD_DEPT' AND type = 'PROCEDURE'
 ORDER BY line;
```

TEXT
1 PROCEDURE add_dept(
2 p_name IN departments.department_name%TYPE,
3 p_loc IN departments.location_id%TYPE) IS
4
5 BEGIN
6 INSERT INTO departments(department_id, department_name, location_id)
7 VALUES (departments_seq.NEXTVAL, p_name, p_loc);
8 END add_dept;

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización del Procedimiento en el Diccionario de Datos

El código fuente 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 fuente. Para ver el código fuente 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

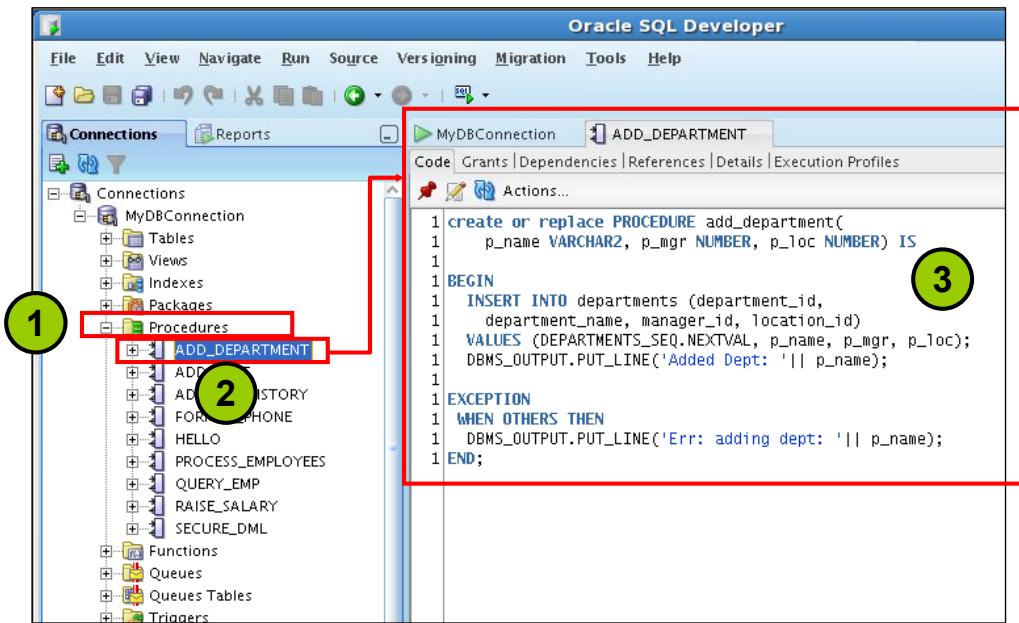
En el ejemplo de consulta se muestran todas las columnas que proporciona la tabla USER_SOURCE:

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

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

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

Visualización de Información de Procedimientos mediante SQL Developer



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización de Información de Procedimientos mediante SQL Developer

Para ver un código de procedimiento en SQL Developer, realice los siguientes pasos:

1. Haga clic en el nodo **Procedures** en el separador **Connections**.
2. Haga clic en el nombre del procedimiento.
3. El código del procedimiento se muestra en el separador **Code**, como se muestra en la diapositiva.

Prueba

Los parámetros formales son valores literales, variables y expresiones utilizadas en la lista de parámetros del subprograma de llamada

1. Verdadero
2. Falso



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: 2

Parámetros Formales o Reales (o argumentos)

- **Parámetros formales:** variables locales que se declaran en la lista de parámetros de una especificación de subprograma.
- **Parámetros reales:** valores literales, variables o expresiones utilizadas en la lista de parámetros del subprograma de llamada.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Identificar las ventajas del diseño de subprogramas basados en módulos y capas
- Crear y llamar a procedimientos
- Utilizar parámetros formales y reales
- Utilizar la notación posicional, con nombre o mixta para la transferencia de parámetros
- Identificar los modos de transferencia de parámetros disponibles
- Manejar excepciones en procedimientos
- Eliminar un procedimiento
- Mostrar información de los procedimientos



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visión General de la Práctica 1: Creación, Compilación y Llamada de Procedimientos

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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica 1: Visión General

En esta práctica, creará, compilará y llamará a procedimientos que emiten comandos DML y de consulta. También aprenderá a manejar excepciones en procedimientos.

Si se producen errores de compilación durante la ejecución de procedimientos, puede utilizar el separador Compiler-Log de SQL Developer.

Nota: se recomienda utilizar SQL Developer para esta práctica.

Importante

En todas las prácticas de este curso y las soluciones de las mismas se supone que los objetos como procedimientos, funciones, etc. se crean mediante el área SQL Worksheet de SQL Developer. Al crear un objeto en el área SQL Worksheet, debe refrescar el nodo de objetos para que el nuevo objeto se muestre en el árbol Navigator. Para compilar el objeto recién creado, puede hacer clic con el botón derecho en el nombre del objeto en el árbol Navigator y, a continuación, seleccionar Compile en el menú de acceso directo. Por ejemplo, después de introducir el código para crear un procedimiento en el área SQL Worksheet, puede hacer clic en el icono Run Script (o pulsar F5) para ejecutar el código. De esta forma, se crea y compila el procedimiento.

También puede crear objetos como procedimientos con el nodo PROCEDURES en el árbol Navigator y, a continuación, compilar el procedimiento. Al crear objetos mediante el árbol Navigator, se muestra automáticamente el objeto recién creado.

Creación de Funciones y Depuración de Subprogramas

2

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

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

- Diferenciar entre un procedimiento y una función
- Describir los usos de las funciones
- Crear funciones almacenadas
- Llamar a una función
- Eliminar una función
- Comprender la funcionalidad básica del depurador de SQL Developer



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos de la Lección

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

Agenda de la Lección

- Trabajar con funciones:
 - Diferencia entre un procedimiento y una función
 - Descripción de los usos de las funciones
 - Creación, llamada y eliminación de funciones almacenadas
- Introducción sobre el depurador de SQL Developer

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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 para otro subprograma
- Se puede agrupar en paquetes PL/SQL



Copyright © 2010, Oracle y/o sus filiales. 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 puede 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.

Las funciones (y procedimientos) se pueden agrupar en paquetes PL/SQL. Los paquetes aumentan cada vez más la capacidad de reutilización y mantenimiento del código. Los paquetes se tratan en las lecciones tituladas “Creación de Paquetes” y “Trabajar con Paquetes”.

Creación de 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 © 2010, Oracle y/o sus filiales. 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 de enlace 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.

Diferencia entre Procedimientos y 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 transferir 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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Diferencia 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 modo 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.

Creación y Ejecución de Funciones: Visión General



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación y Ejecución de Funciones: Visión General

En el diagrama de la diapositiva se ilustran los pasos básicos de creación y ejecución de una función:

1. Cree la función mediante el árbol Object Navigator de SQL Developer en el área SQL Worksheet.
2. Compile la función. La función se crea en la base de datos. La sentencia `CREATE FUNCTION` crea y almacena el código fuente y el valor *m-code* compilado en la base de datos. Para compilar la función, haga clic con el botón derecho en el nombre de la función en el árbol Object Navigator y, a continuación, haga clic en Compile.
3. Si hay errores o advertencias de compilación, puede ver (y, a continuación, corregir) las advertencias o errores mediante uno de los siguientes métodos:
 - a. Uso de la interfaz de SQL Developer (separador Compiler – Log)
 - b. Uso del comando de SQL*Plus `SHOW ERRORS`
 - c. Uso de las vistas `USER/ALL/DBA_ERRORS`
4. Después de terminar la compilación correctamente, llame a la función para volver al valor deseado.

Creación y Llamada a Funciones Almacenadas mediante la Sentencia CREATE FUNCTION: Ejemplo

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

FUNCTION get_sal Compiled.

```
-- Invoke the function as an expression or as
-- a parameter value.
```

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

anonymous block completed
24000

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 script 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 cuadro de código utiliza el comando `EXECUTE` de 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 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. En el ejemplo no se devuelve un valor si no se encuentra una fila para un `id` determinado. Lo ideal es crear un manejador de excepciones para que también devuelva un valor.

Uso de Diferentes Métodos para Ejecutar Funciones

```
-- As a PL/SQL expression, get the results using host variables
```

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

```
anonymous block completed
b_salary
-----
24000
```

```
-- As a PL/SQL expression, get the results using a local
-- variable
SET SERVEROUTPUT ON
DECLARE
    sal employees.salary%type;
BEGIN
    sal := get_sal(100);
    DBMS_OUTPUT.PUT_LINE('The salary is: '|| sal);
END ;
/
```

```
anonymous block completed
The salary is: 24000
```

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Diferentes Métodos para Ejecutar Funciones

Si las funciones están bien diseñadas, se pueden convertir en construcciones potentes. Se puede llamar a las funciones de la siguiente forma:

- **Como parte de expresiones PL/SQL:** puede utilizar variables de host o locales para que contengan el valor devuelto de una función. El primer ejemplo de la diapositiva utiliza una variable de host y el segundo ejemplo utiliza una variable local en un bloque anónimo.

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

Uso de Diferentes Métodos para Ejecutar Funciones

```
-- Use as a parameter to another subprogram

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

```
anonymous block completed
24000
```

```
-- Use in a SQL statement (subject to restrictions)

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

JOB_ID	GET_SAL(EMPLOYEE_ID)
SH_CLERK	2600
SH_CLERK	2600
AD_ASST	4400
MK_MAN	13000

```
...
SH_CLERK 3100
SH_CLERK 3000

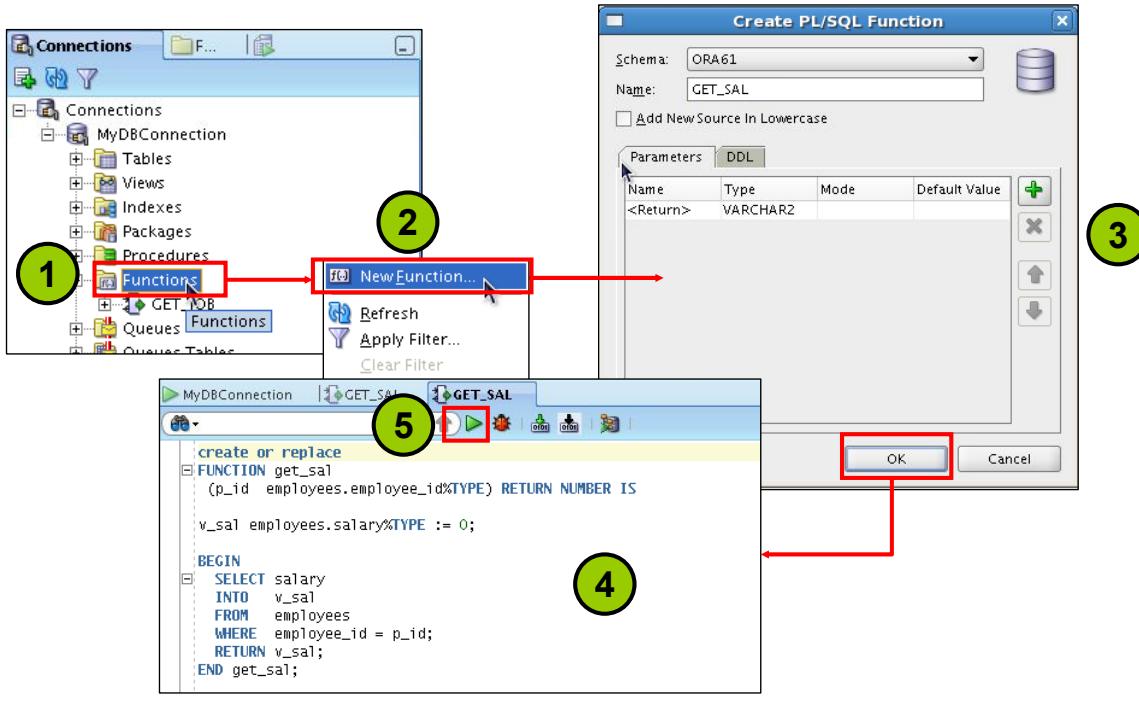
107 rows selected
```

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Diferentes Métodos para Ejecutar Funciones (continuación)

- **Como parámetro de otro subprograma:** en el primer ejemplo de la diapositiva se 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 *Oracle Database: Conceptos Fundamentales de SQL I*.
- **Como expresión de una sentencia SQL:** en el segundo ejemplo se muestra el modo en que una función se puede utilizar como función de una sola fila en una sentencia SQL.

Creación y Compilación de Funciones mediante SQL Developer



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación y Compilación de Funciones mediante SQL Developer

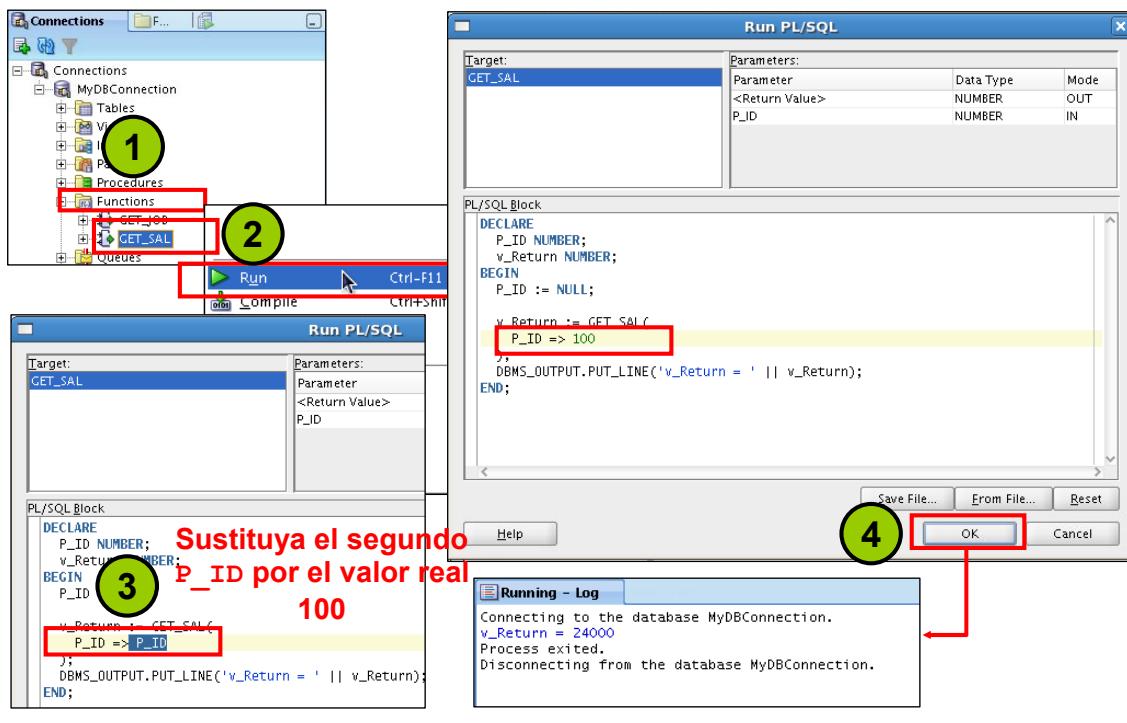
Puede crear una nueva función en SQL Developer realizando los siguientes pasos:

1. Haga clic con el botón derecho en el nodo **Functions**.
2. Seleccione **New Function** en el menú de acceso directo. Se muestra el cuadro de diálogo **Create PL/SQL Function**.
3. Seleccione el esquema, el nombre de la función y la lista de parámetros (con el ícono +) y, a continuación, haga clic en **OK**. Se muestra el editor de códigos de la función.
4. Introduzca el código de la función.
5. Para compilar la función, haga clic en el ícono **Compile**.

Nota

- Para crear una nueva función en SQL Developer, también puede introducir el código en SQL Worksheet y, a continuación, hacer clic en el ícono Run Script.
- Para obtener más información sobre la creación de funciones en SQL Developer, consulte el tema de la ayuda en pantalla correspondiente sobre la creación de procedimientos o funciones de subprograma PL/SQL.

Ejecución de Funciones mediante SQL Developer



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Ejecución de Funciones mediante SQL Developer

Puede ejecutar una función en SQL Developer realizando los siguientes pasos:

1. Haga clic en el nodo **Functions**.
2. Haga clic con el botón derecho en el nombre de la función y, a continuación, seleccione **Run**. Se muestra el cuadro de diálogo **Run PL/SQL**.
3. Sustituya el segundo nombre del parámetro por el valor de parámetro real, como se muestra en el ejemplo de la diapositiva.
4. Haga clic en **OK**.

Nota: para tener más información sobre la ejecución de funciones en SQL Developer, consulte el tema de la ayuda en pantalla sobre la ejecución y depuración de funciones y procedimientos.

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



Copyright © 2010, Oracle y/o sus filiales. 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. Las funciones aumentan 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.

Uso de Funciones en Expresiones SQL: Ejemplo

```

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

```

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

6 rows selected

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Funciones en Expresiones SQL: Ejemplo

En el ejemplo de la diapositiva se 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%.

Para ejecutar el código que se muestra en el ejemplo de la diapositiva en SQL Developer, introduzca el código en la hoja de trabajo de SQL y, a continuación, haga clic en el icono **Run Script**. 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.

Llamada a Funciones Definidas por el Usuario en Sentencias SQL

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`
- 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`

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Llamada a Funciones Definidas por el Usuario en Sentencias SQL

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, como se muestra en el 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
205	960
147	960
108	960
168	920
10 rows selected	

Restricciones al Llamar a Funciones desde Expresiones SQL

- Las funciones definidas por el usuario 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:
 - El usuario debe ser propietario de la función o tener el privilegio `EXECUTE`
 - Puede que necesite activar la palabra clave `PARALLEL_ENABLE` para permitir una ejecución en paralelo de la sentencia SQL



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Restricciones al 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 e `IN`.
- 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.

Las siguientes restricciones se aplican 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`.
- Puede que necesite activar la palabra clave `PARALLEL_ENABLE` para permitir una ejecución en paralelo de la sentencia SQL mediante la función. Cada esclavo en paralelo tendrá copias privadas de las variables locales de la función.

Otras restricciones aplicables a las funciones definidas por el usuario son las siguientes: no se puede llamar desde la cláusula de restricción `CHECK` de una sentencia `CREATE TABLE` o `ALTER TABLE`. Además, no se pueden utilizar para especificar un valor por defecto para una columna. 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 puede contener sentencias DML
- Una sentencia UPDATE o DELETE de una tabla T no puede realizar consultas ni contener DML en la misma tabla T
- Las sentencias SQL no pueden terminar transacciones (es decir, no pueden ejecutar operaciones COMMIT ni ROLLBACK)

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



Copyright © 2010, Oracle y/o sus filiales. 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:

- Cuando se llama a una función desde una sentencia SELECT o una sentencia paralela UPDATE o DELETE, la función no puede modificar las tablas de base de datos.
- Cuando se llama a una función desde una sentencia UPDATE o DELETE, la función no puede consultar ni modificar las tablas de base de datos modificadas por dicha sentencia.
- Cuando se llama a una función desde una sentencia SELECT, INSERT, UPDATE o DELETE, la función no 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 va seguida de una confirmación automática.

Restricciones al Llamar a Funciones desde SQL: Ejemplo

```
CREATE OR REPLACE FUNCTION dml_call_sql(p_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', p_sal);
  RETURN (p_sal + 100);
END;
```

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

```
FUNCTION dml_call_sql(p_sal Compiled.

Error starting at line 1 in command:
UPDATE employees
  SET salary = dml_call_sql(2000)
WHERE employee_id = 170
Error report:
SQL Error: ORA-04091: table ORA62.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA62.DML_CALL_SQL", line 4
04091. 00000 - "Table %s.%s is mutating, trigger/function may not see it"
*Cause: A trigger (or a user defined plsql function that is referenced in
this statement) attempted to look at (or modify) a table that was
in the middle of being modified by the statement which fired it.
*Action: Rewrite the trigger (or function) so it does not read that table.
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Restricciones al Llamar a Funciones desde SQL: Ejemplo

La función `dml_call_sql` de la diapositiva 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(p_a NUMBER)
  RETURN NUMBER IS
  v_s NUMBER;
BEGIN
  SELECT salary INTO v_s FROM employees
  WHERE employee_id = 170;
  RETURN (v_s + p_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 diapositiva:

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

Notación con Nombre y Mixta desde SQL

- PL/SQL permite que se especifiquen argumentos en una llamada de subrutina mediante una notación posicional, con nombre o mixta.
- Antes de Oracle Database 11g, sólo se soporta la notación posicional en llamadas de SQL.
- A partir de Oracle Database 11g, las notaciones con nombre y mixtas se pueden utilizar para especificar argumentos en llamadas a subrutinas PL/SQL desde sentencias SQL.
- Para listas de parámetros largas, la mayoría con valores por defecto, puede omitir valores de los parámetros opcionales.
- Puede evitar la duplicación del valor por defecto del parámetro opcional en cada sitio de llamada.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Notación con Nombre y Mixta desde SQL: Ejemplo

```
CREATE OR REPLACE FUNCTION f(
    p_parameter_1 IN NUMBER DEFAULT 1,
    p_parameter_5 IN NUMBER DEFAULT 5)
RETURN NUMBER
IS
    v_var number;
BEGIN
    v_var := p_parameter_1 + (p_parameter_5 * 2);
    RETURN v_var;
END f;
/
```

FUNCTION f(Compiled.

```
SELECT f(p_parameter_5 => 10) FROM DUAL;
```

```
F(P_PARAMETER_5=>10)
-----
21
1 rows selected
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Ejemplo de Uso de Notación con Nombre y Mixta desde una Sentencia SQL

En el ejemplo de la diapositiva, la llamada a la función `f` en la sentencia SQL `SELECT` utiliza la notación con nombre. Antes de Oracle Database 11g, no se podía utilizar la notación con nombre o mixta al transferir parámetros a una función desde una sentencia SQL. Antes de Oracle Database 11g, recibía el siguiente error:

```
SELECT f(p_parameter_5 => 10) FROM DUAL;
```

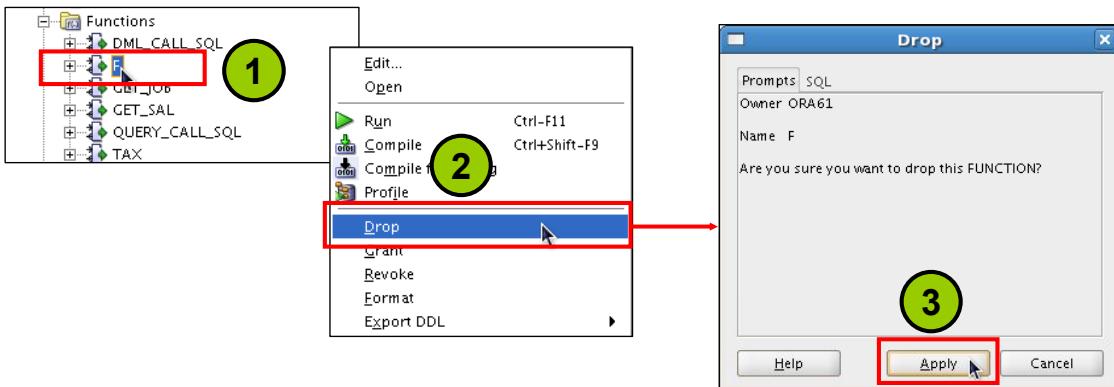
ORA-00907: missing right parenthesis

Eliminación de Funciones: Uso de la Sentencia SQL DROP o SQL Developer

- **Uso de la sentencia DROP:**

```
DROP FUNCTION f;
```

- **Uso de SQL Developer:**



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Eliminación de Funciones

Uso de la sentencia DROP

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

Uso de **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 utiliza **DROP** para borrar una función y, a continuación, vuelve a crearla, todos los privilegios otorgados para esta función se revocan automáticamente.

Uso de SQL Developer

Para borrar una función en SQL Developer, haga clic con el botón derecho en el nombre de la función en el nodo **Functions** y, a continuación, seleccione **Drop**. Aparece el cuadro de diálogo **Drop**. Para borrar la función, haga clic en **Apply**.

Visualización de Funciones mediante Vistas de Diccionario de Datos

```
DESCRIBE USER_SOURCE
```

Name	Null	Type
NAME		VARCHAR2(30)
TYPE		VARCHAR2(12)
LINE		NUMBER
TEXT		VARCHAR2(4000)

4 rows selected

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

Results:	
TEXT	
1	FUNCTION tax(p_value IN NUMBER)
2	FUNCTION query_call_sql(p_a NUMBER) RETURN NUMBER IS
3	FUNCTION get_sal
4	FUNCTION dml_call_sql(p_sal NUMBER)
5	RETURN NUMBER IS
6	RETURN NUMBER IS
7	(p_id employees.employee_id%TYPE) RETURN NUMBER IS
8	v_s NUMBER;

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización de Funciones mediante Vistas de Diccionario de Datos

El código fuente 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 fuente. 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:

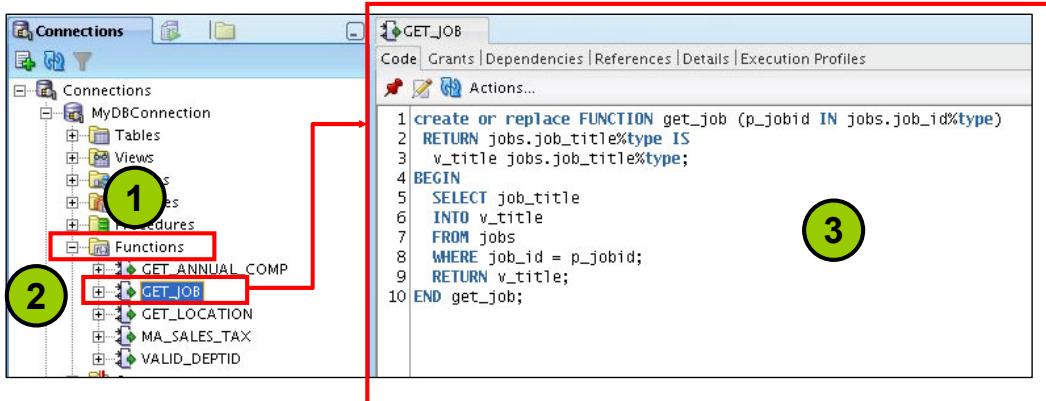
- En la tabla USER_SOURCE para mostrar el código PL/SQL del que es propietario
- En la tabla ALL_SOURCE para mostrar el código PL/SQL para el que el propietario de dicho código de subprograma le ha otorgado el derecho EXECUTE

En el segundo ejemplo de la diapositiva se utiliza la tabla USER_SOURCE para mostrar el código fuente para todas las funciones del esquema.

También puede utilizar la vista de diccionario de datos USER_OBJECTS para mostrar una lista de los nombres de función.

Nota: la salida del segundo ejemplo de código de la diapositiva se ha generado mediante el ícono Execute Statement (F9) de la barra de herramientas para proporcionar una salida con mejor formato.

Visualización de Información de Funciones mediante SQL Developer



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización de Información de Funciones mediante SQL Developer

Para ver un código de función en SQL Developer, realice los siguientes pasos:

1. Haga clic en el nodo **Functions** del separador **Connections**.
2. Haga clic en el nombre de la función.
3. El código de la función se muestra en el separador **Code**, como aparece en la diapositiva.

Prueba

Una función PL/SQL almacenada:

1. Se llama como parte de una expresión
2. Debe contener una cláusula RETURN en la cabecera
3. Debe devolver un único valor
4. Debe contener al menos una sentencia RETURN
5. No contiene una cláusula RETURN en la cabecera



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: 1, 2, 3, 4

Práctica 2-1: 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 utilizarlas 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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica 2-1: Visión General

Se recomienda utilizar SQL Developer para esta práctica.

Agenda de la Lección

- Trabajar con funciones:
 - Diferencia entre un procedimiento y una función
 - Descripción de los usos de las funciones
 - Creación, llamada y eliminación de funciones almacenadas
- Introducción sobre el depurador de SQL Developer

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Depuración de Subprogramas PL/SQL mediante el Depurador de SQL Developer

- Puede utilizar el depurador para controlar la ejecución del programa PL/SQL.
- Para depurar un subprograma PL/SQL, un *administrador de seguridad* debe otorgar los siguientes privilegios al desarrollador de aplicaciones:
 - DEBUG ANY PROCEDURE
 - DEBUG CONNECT SESSION

```
GRANT DEBUG ANY PROCEDURE TO ora61;
GRANT DEBUG CONNECT SESSION TO ora61;
```

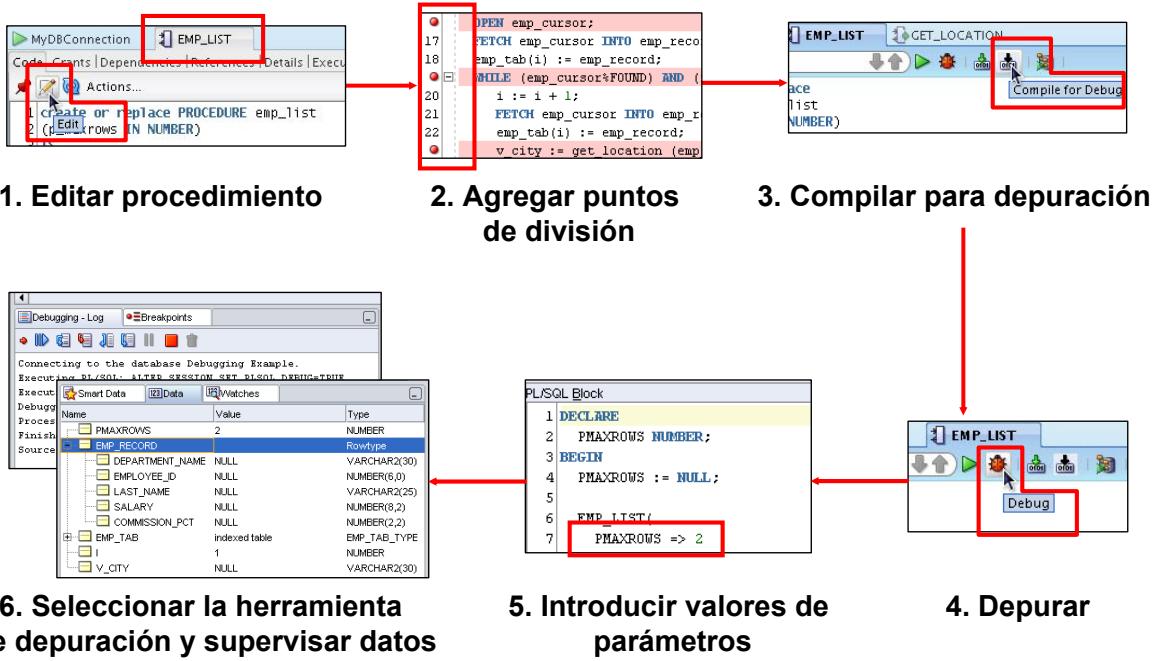


Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Desplazamiento por el Código al Depurar

El depurador de SQL Developer permite controlar la ejecución del programa. Puede controlar si el programa ejecuta una única línea de código, un subprograma completo (procedimiento o función) o un bloque de programa completo. Controlando de forma manual cuándo se debe ejecutar el programa y cuándo se debe hacer una pausa, podrá moverse rápidamente por las secciones que sabe que funcionan correctamente y concentrarse en las secciones que están causando problemas.

Depuración de Subprogramas: Visión General



Separador de Edición de Código de Función o Procedimiento

```

1  create or replace
2  PROCEDURE emp_list
3  (p_maxrows IN NUMBER)
4  IS
5  CURSOR cur_emp IS
6      SELECT d.department_name, e.employee_id, e.last_name,
7          e.salary, e.commission_pct
8      FROM departments d, employees e
9      WHERE d.department_id = e.department_id;
10     rec_emp cur_emp%ROWTYPE;
11     TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY BINARY_INTEGER;
12     emp_tab emp_tab_type;
13     i NUMBER := 1;
14     v_city VARCHAR2(30);
15 BEGIN
16     OPEN cur_emp;
17     FETCH cur_emp INTO rec_emp;
18     emp_tab(i) := rec_emp;
19     WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
20         i := i + 1;
21         FETCH cur_emp INTO rec_emp;
22         emp_tab(i) := rec_emp;
23         v_city := get_location (rec_emp.department_name);
24         dbms_output.put_line('Employee ' || rec_emp.last_name ||
25             ' works in ' || v_city );
26     END LOOP;
27     CLOSE cur_emp;
28     FOR j IN REVERSE 1..i LOOP
29         DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
30     END LOOP;
31 END emp_list;
32

```

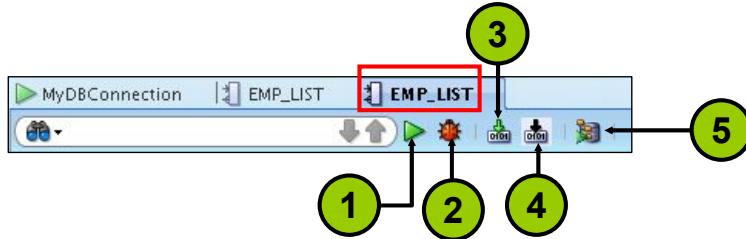
ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Separador de Código de Función o Procedimiento

Este separador muestra una barra de herramientas y el texto del subprograma, que puede editar. Para definir y anular la definición de puntos de división de la depuración, haga clic a la izquierda de la delgada línea vertical situada junto a cada sentencia con la que desee asociar un punto de división. (Al definir un punto de división, aparece un círculo rojo.)

Barra de Herramientas del Separador de Función o Procedimiento

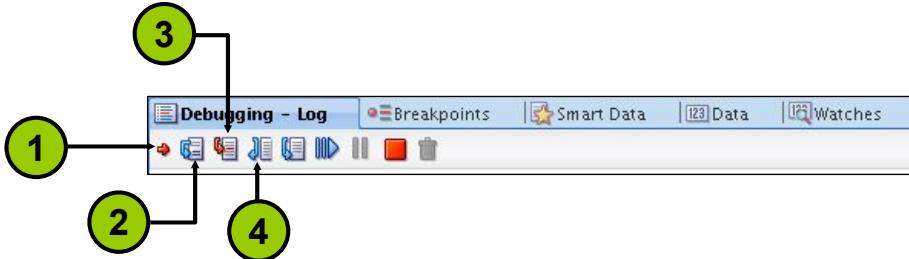


Icono	Descripción
1. Run	Inicia una ejecución normal de la función o procedimiento y muestra los resultados en el separador Running - Log
2. Debug	Ejecuta el subprograma en modo de depuración y muestra el separador Debugging - Log, que incluye la barra de herramientas de depuración para controlar la ejecución
3. Compile	Compila el subprograma
4. Compile for Debug	Compila el subprograma para que se pueda depurar
5. Profile	Muestra la ventana Profile que se utiliza para especificar los valores de parámetros para ejecutar, depurar o crear perfiles de un procedimiento o función PL/SQL

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Barra de Herramientas del Separador Debugging – Log



Icono	Descripción
1. Find Execution Point	Va al siguiente punto de ejecución
2. Step Over	Salta el siguiente subprograma y va a la siguiente sentencia después del subprograma
3. Step Into	Ejecuta una única sentencia de programa al mismo tiempo. Si el punto de ejecución se ubica en una llamada a un subprograma, va a la primera línea ejecutable en la primera sentencia de dicho subprograma
4. Step Out	Deja el subprograma actual y va a la siguiente sentencia con un punto de división

ORACLE

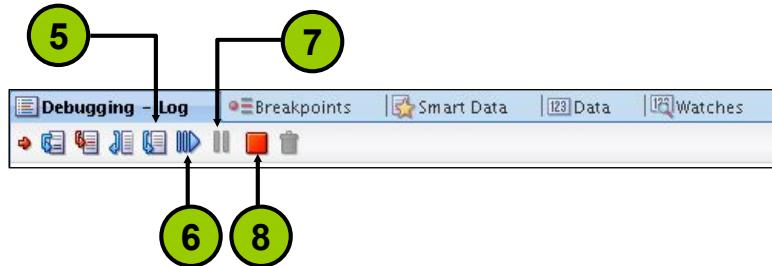
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Barra de Herramientas del Separador Debugging – Log

El separador Debugging - Log contiene la barra de herramientas de depuración y mensajes informativos.

1. **Find Execution Point:** va al punto de ejecución (la siguiente línea del código fuente que ejecutará el depurador)
2. **Step Over:** salta el siguiente subprograma (a menos que el subprograma tenga un punto de división) y va a la siguiente sentencia después del subprograma. Si el punto de ejecución se ubica en una llamada de subprograma, ejecuta dicho subprograma sin parar (en lugar de ir a la primera línea ejecutable del mismo) y, a continuación, posiciona el punto de ejecución en la sentencia que sigue a la llamada. Si el punto de ejecución se ubica en la última sentencia de un subprograma, Step Over vuelve del subprograma, colocando el punto de ejecución en la línea del código que sigue a la llamada al subprograma desde el que vuelve.
3. **Step Into:** ejecuta una única sentencia de programa al mismo tiempo. Si el punto de ejecución se ubica en una llamada a un subprograma, Step Into a la primera línea ejecutable de dicho subprograma y coloca el punto de ejecución en su primera sentencia. Si el punto de ejecución se ubica en la última sentencia de un subprograma, Step Into vuelve del subprograma, colocando el punto de ejecución en la línea del código que sigue a la llamada al subprograma desde el que vuelve.
4. **Step Out:** deja el subprograma actual y va a la siguiente sentencia

Barra de Herramientas del Separador Debugging – Log



Icono	Descripción
5. Step to End of Method	Va a la última sentencia del subprograma actual
6. Resume	Continúa con la ejecución
7. Pause	Para la ejecución, pero no sale
8. Terminate	Para y sale de la ejecución

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Barra de Herramientas del Separador Debugging – Log (continuación)

5. **Step to End of Method:** va a la última sentencia del subprograma actual
6. **Resume:** continúa con la ejecución
7. **Pause:** para la ejecución pero no sale, lo que permite reanudar la ejecución
8. **Terminate:** para y sale de la ejecución. No puede reanudar la ejecución desde este punto; en su lugar, para iniciar la ejecución o depuración desde comienzo del subprograma, haga clic en el ícono Run o Debug de la barra de herramientas del separador Source.

Separadores Adicionales

Debugging - Log	Breakpoints	Smart Data	Data	Watches
Name	Value	Type		
P_MAXROWS	100	NUMBER		
REC_EMP		Rowtype		
EMP_TAB	indexed table	EMP_TAB_TYPE		
I	1	NUMBER		
V_CITY	NULL	VARCHAR2(30)		

Tabulador	Descripción
Breakpoints	Muestra los puntos de división, definidos por el sistema y por el usuario.
Smart Data:	Muestra información sobre variables. Puede especificar estas preferencias haciendo clic con el botón derecho en la ventana Smart Data y seleccionando Preferences.
Data	Ubicado en el área de texto de código; muestra información sobre todas las variables.
Watches	Ubicado en el área de texto de código; muestra información sobre todas las comprobaciones.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Separadores Adicionales

Definición de Comprobaciones de Expresión

Una comprobación permite supervisar los valores que cambian de las variables o expresiones durante la ejecución del programa. Al introducir una expresión de comprobación, la ventana Watches muestra el valor actual de la expresión. Durante la ejecución del programa, el valor de la comprobación cambia a medida que el programa actualiza los valores de las variables en la expresión de comprobación.

Una comprobación evalúa una expresión según el contexto actual, que se controla mediante la selección en la ventana Stack. Si se mueve a un nuevo contexto, la expresión se vuelve a evaluar para el nuevo contexto. Si el punto de ejecución se mueve a una ubicación en la que algunas de las variables de la expresión de comprobación no están definidas, la expresión de comprobación completa se convierte en no definida. Si el punto de ejecución vuelve a una ubicación en la que se puede evaluar la expresión de comprobación, la ventana Watches vuelve a mostrar el valor de la expresión de comprobación.

Para abrir la ventana Watches, haga clic en View, Debugger y, a continuación, Watches.

Para agregar una comprobación, haga clic con el botón derecho en la ventana Watches y seleccione Add Watch. Para editar una comprobación, haga clic con el botón derecho en la ventana Watches y seleccione Edit Watch.

Nota: si no puede ver algunos de los separadores de depuración descritos en esta lección, puede volver a mostrar dichos separadores mediante la opción de menú View > Debugger.

Ejemplo de Depuración de Procedimiento: Creación de un Nuevo Procedimiento emp_list

```

1 CREATE OR REPLACE PROCEDURE emp_list(pmaxrows IN NUMBER) AS
2 CURSOR emp_cursor IS
3 SELECT d.department_name,
4       e.employee_id,
5       e.last_name,
6       e.salary,
7       e.commission_pct
8 FROM departments d,
9      employees e
10 WHERE d.department_id = e.department_id;
11 emp_record emp_cursor % rowtype;
12 type emp_tab_type IS TABLE OF emp_cursor % rowtype INDEX BY binary_integer;
13 emp_tab emp_tab_type;
14 i NUMBER := 1;
15 v_city VARCHAR2(30);
16 BEGIN
17
18   OPEN emp_cursor;
19   FETCH emp_cursor
20     INTO emp_record;
21   emp_tab(i) := emp_record;
22   WHILE(emp_cursor % FOUND)
23     AND(i <= pmaxrows)
24   LOOP
25     i := i + 1;
26     FETCH emp_cursor
27       INTO emp_record;
28     emp_tab(i) := emp_record;
29     v_city := get_location(emp_record.department_name);
30     DBMS_OUTPUT.PUT_LINE('Employee ' || emp_record.last_name || ' works in ' || v_city);
31   END LOOP;
32
33   CLOSE emp_cursor;
34   FOR j IN REVERSE 1 .. i
35   LOOP
36     DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
37   END LOOP;
38 END emp_list;

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Ejemplo de Depuración de Procedimiento: Creación de un Nuevo Procedimiento emp_list

El procedimiento emp_list recopila información de empleado como el nombre de departamento, identificador, nombre, salario y porcentaje de comisión del empleado. El procedimiento crea un registro para almacenar la información de empleado. El procedimiento también crea una tabla que puede contener varios registros de empleados. “i” es un contador.

El código abre el cursor y recupera los registros del empleado. El código también comprueba si hay más registros que recuperar o si el número de registros recuperados es menor que el número de registros especificados. El código imprime finalmente la información de empleado. El procedimiento también llama a la función get_location, que devuelve el nombre de la ciudad en la que trabaja el empleado.

Nota: asegúrese de que muestra el código de procedimiento en modo de edición. Para editar el código de procedimiento, haga clic en Edit en la barra de herramientas del procedimiento.

Ejemplo de Depuración de Procedimiento: Creación de una Nueva Función get_location

```
1  CREATE OR REPLACE FUNCTION get_location(p_deptname IN VARCHAR2) RETURN VARCHAR2 AS
2    v_loc_id NUMBER;
3    v_city VARCHAR2(30);
4  BEGIN
5    SELECT d.location_id,
6      l.city
7    INTO v_loc_id,
8      v_city
9    FROM departments d,
10      locations l
11   WHERE UPPER(department_name) = UPPER(p_deptname)
12   AND d.location_id = l.location_id;
13   RETURN v_city;
14 END get_location;
```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Ejemplo de Depuración de Procedimiento: Creación de un Nuevo Procedimiento

Esta función devuelve la ciudad en la que trabaja un empleado. Se llama desde el procedimiento emp_list .

Definición de Puntos de División y Compilación de emp_list para Modo de Depuración

```

1 create or replace
2 PROCEDURE emp_list
3 (p_maxrows IN NUMBER)
4 IS
5 CURSOR cur_emp IS
6   SELECT d.department_name, e.employee_id, e.last_name,
7         e.salary, e.commission_pct
8   FROM departments d, employees e
9   WHERE d.department_id = e.department_id;
10  rec_emp cur_emp%ROWTYPE;
11  TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY BINARY_INTEGER;
12  emp_tab emp_tab_type;
13  i NUMBER := 1;
14  v_city VARCHAR2(30);
15  BEGIN
16    OPEN cur_emp;
17    FETCH cur_emp INTO rec_emp;
18    emp_tab(i) := rec_emp;
19    WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
20      i := i + 1;
21      FETCH cur_emp INTO rec_emp;
22      emp_tab(i) := rec_emp;
23      v_city := get_location (rec_emp.department_name);
24      dbms_output.put_line('Employee ' || rec_emp.last_name || 
25                           ' works in ' || v_city );
26    END LOOP;
27    CLOSE cur_emp;
28    FOR j IN REVERSE 1..i LOOP
29      DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
30    END LOOP;
31  END emp_list;
32

```

Messages - Log

EMP_LIST Compiled

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Definición de Puntos de División y Compilación de emp_list para Modo de Depuración

En el ejemplo de la diapositiva, el procedimiento emp_list se muestra en modo de edición y se agregan cuatro puntos de división a varias ubicaciones en el código. Para compilar el procedimiento para la depuración, haga clic con el botón derecho en el código y, a continuación, seleccione Compile for Debug desde el menú de acceso directo. El separador Messages – Log muestra el mensaje de que se ha compilado el procedimiento.

Compilación de la Función `get_location` para el Modo de Depuración

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, there are tabs for 'MyDBConnection', 'EMP_LIST', 'GET_LOCATION', and 'GET_LOCATION'. The 'GET_LOCATION' tab is active. A red box highlights the 'Compile for Debug' button in the toolbar, which is located next to the 'Run' and 'Stop' buttons. Another red box highlights the message 'GET_LOCATION Compiled' in the 'Messages - Log' window at the bottom left.

```

1 create or replace
2 FUNCTION get_location
3 (p_deptname IN VARCHAR2) RETURN VARCHAR2
4 AS
5   v_loc_id NUMBER;
6   v_city  VARCHAR2(30);
7 BEGIN
8   SELECT d.location_id, l.city INTO v_loc_id, v_city
9   FROM departments d, locations l
10  WHERE upper(department_name) = upper(p_deptname)
11  and d.location_id = l.location_id;
12  RETURN v_city;
13 END GET_LOCATION;
14

```

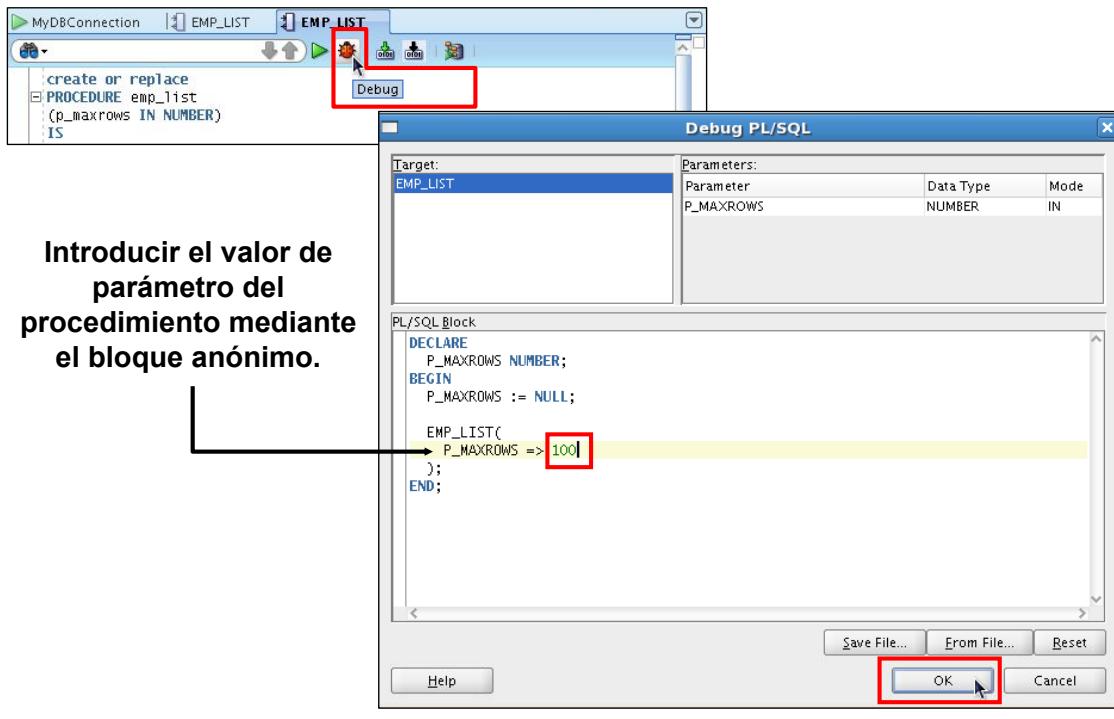
ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Compilación de la Función `get_location` para el Modo de Depuración

En el ejemplo de la diapositiva, se muestra la función `get_location` en modo de edición. Para compilar la función para la depuración, haga clic con el botón derecho en el código y, a continuación, seleccione `Compile for Debug` en el menú de acceso directo. El separador `Messages – Log` muestra el mensaje de que se ha compilado la función.

Depuración de emp_list e Introducción de Valores para el Parámetro PMAXROWS



Introducir el valor de parámetro del procedimiento mediante el bloque anónimo.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

ORACLE

Depuración de emp_list e Introducción de Valores para el Parámetro PMAXROWS

El siguiente paso en el proceso de depuración consiste en depurar el procedimiento mediante cualquiera de los diferentes métodos disponibles mencionados anteriormente, por ejemplo, hacer clic en el ícono Debug en la barra de herramientas del procedimiento. Se muestra un bloque anónimo, donde se le ha solicitado introducir los parámetros para este procedimiento. emp_list tiene un parámetro, PMAXROWS, que especifica el número de registros que devolver. Sustituya el segundo PMAXROWS por un número, por ejemplo 100 y, a continuación, haga clic en OK.

Depuración de emp_list: Ejecución de Step Into (F7) en el Código

El control de programa se para en el primer punto de división.

```

1 CREATE OR REPLACE PROCEDURE emp_list
2  (p_maxrows IN NUMBER)
3  IS
4  CURSOR cur_emp IS
5   SELECT d.department_name, e.employee_id, e.last_name,
6    e.salary, e.commission_pct
7   FROM departments d, employees e
8   WHERE d.department_id = e.department_id;
9   rec_emp cur_emp%ROWTYPE;
10  TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY BINARY_INTEGER;
11  emp_tab emp_tab_type;
12  i NUMBER := 1;
13  v_city VARCHAR2(30);
14  BEGIN
15  OPEN cur_emp;
16  FETCH cur_emp INTO rec_emp;
17  emp_tab(i) := rec_emp;
18  WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19    i := i + 1;
20    FETCH cur_emp INTO rec_emp;
21    emp_tab(i) := rec_emp;
22    v_city := get_location (rec_emp.department_name);
23    dbms_output.put_line('Employee ' || rec_emp.last_name ||
24      ' works in ' || v_city );
25  END LOOP;
26  CLOSE cur_emp;
27  FOR j IN REVERSE 1..i LOOP
28    DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
29  END LOOP;
30 END emp_list;

```

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Herramienta de Depuración Step Into

El comando Step Into ejecuta una única sentencia de programa al mismo tiempo. Si el punto de ejecución se ubica en una llamada a un subprograma, el comando Step Into va a la primera línea ejecutable de dicho subprograma y coloca el punto de ejecución en su primera sentencia del subprograma. Si el punto de ejecución se ubica en la última sentencia de un subprograma, la selección de Step Into provoca que el depurador vuelva del subprograma, colocando el punto de ejecución en la línea del código que sigue a la llamada al subprograma desde el que vuelve. El término *pasos únicos* hace referencia al uso de Step Into para la ejecución sucesiva mediante las sentencias del código de programa. Puede ir a la primera línea ejecutable en un subprograma de cualquiera de las siguientes formas: seleccione Debug > Step Into, pulse F7 o haga clic en el icono Step Into en la barra de herramientas Debugging – Log.

En el ejemplo de la diapositiva, el control de programa se para en el primer punto de división en el código. La flecha junto al punto de división indica que es la línea del código que se ejecutará a continuación. Observe los diferentes separadores en la parte inferior de la ventana.

Nota: los comandos Step Into y Step Over proporcionan la forma más simple de desplazarse por el código de programa. Aunque los dos comandos son muy similares, cada uno proporciona una forma diferente de controlar la ejecución del código.

Depuración de emp_list: Ejecución de Step Into (F7) en el Código

```

14 BEGIN
15   OPEN cur_emp;
16   FETCH cur_emp INTO rec_emp;
17   emp_tab(i) := rec_emp;
18   WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19     i := i + 1;
20     FETCH cur_emp INTO rec_emp;
21     emp_tab(i) := rec_emp;
22     v_city := get_location(rec_emp.department_id);
23     dbms_output.put_line('Employee ' || rec_emp.last_name || ' works in ' || v_city );
24   END LOOP;
25   CLOSE cur_emp;
26   FOR j IN REVERSE 1..i LOOP
27     DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
28   END LOOP;
29   END emp_list;
30
31

```

Step Into (F7): va a la primera línea ejecutable y ejecuta el código del cursor. El control se transfiere a la definición del cursor.

Name	Type
P_MAXROWS	NUMBER
REC_EMP	Rowtype
DEPARTMENT_NAME	VARCHAR2(30)
EMPLOYEE_ID	NUMBER(6,0)
LAST_NAME	VARCHAR2(25)
SALARY	NUMBER(8,2)
COMMISSION_PCT	NUMBER(2,2)
EMP_TAB	indexed table
I	NUMBER
V_CITY	VARCHAR2(30)

```

6   SELECT d.department_name, e.employee_id,
7        e.salary, e.commission_pct
8   FROM departments d, employees e
9  WHERE d.department_id = e.department_id;
rec_emp cur_emp%ROWTYPE;

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Depuración de emp_list: Ejecución de Step Into en el Código

Al seleccionar el comando Step Into, se ejecuta una única sentencia de programa al mismo tiempo. Si el punto de ejecución se ubica en una llamada a un subprograma, el comando Step Into va hasta la primera línea ejecutable en dicho subprograma y coloca la ejecución en la primera sentencia del subprograma.

En el ejemplo de la diapositiva, al pulsar F7 se ejecuta la línea del código en el primer punto de división. En este caso, el control de programa se transfiere a la sección en la que se ha definido el cursor.

Visualización de Datos

The screenshot shows two code snippets and two corresponding Data windows from the Oracle Database Developer Tools.

Code Snippet 1:

```

18 OPEN emp_cursor;
19
20  FETCH emp_cursor

```

Code Snippet 2:

```

16 OPEN cur_emp;
17   FETCH cur_emp INTO rec_emp;
18   emp_tab(1) := rec_emp;
19   WHILE (cur_emp%FOUND) AND (1 <= p_maxrows) LOOP

```

Data Window 1 (Line 18 execution):

Name	Value	Type
P_MAXROWS	100	NUMBER
REC_EMP	Rowtype	
DEPARTMENT_NAME	NULL	VARCHAR2(30)
EMPLOYEE_ID	NULL	NUMBER(6,0)
LAST_NAME	NULL	VARCHAR2(25)
SALARY	NULL	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)
EMP_TAB	Indexed table	EMP_TAB_TYPE
_values		EMP_TAB_TYPE elem...
I	1	NUMBER
V_CITY	NULL	VARCHAR2(30)

Data Window 2 (Line 19 execution):

Name	Value	Type
P_MAXROWS	100	NUMBER
REC_EMP	Rowtype	
DEPARTMENT_NAME	'Executive'	VARCHAR2(30)
EMPLOYEE_ID	100	NUMBER(6,0)
LAST_NAME	'King'	VARCHAR2(25)
SALARY	26400	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)
EMP_TAB	Indexed table	EMP_TAB_TYPE
_values		EMP_TAB_TYPE elem...
I	1	NUMBER
V_CITY	NULL	VARCHAR2(30)

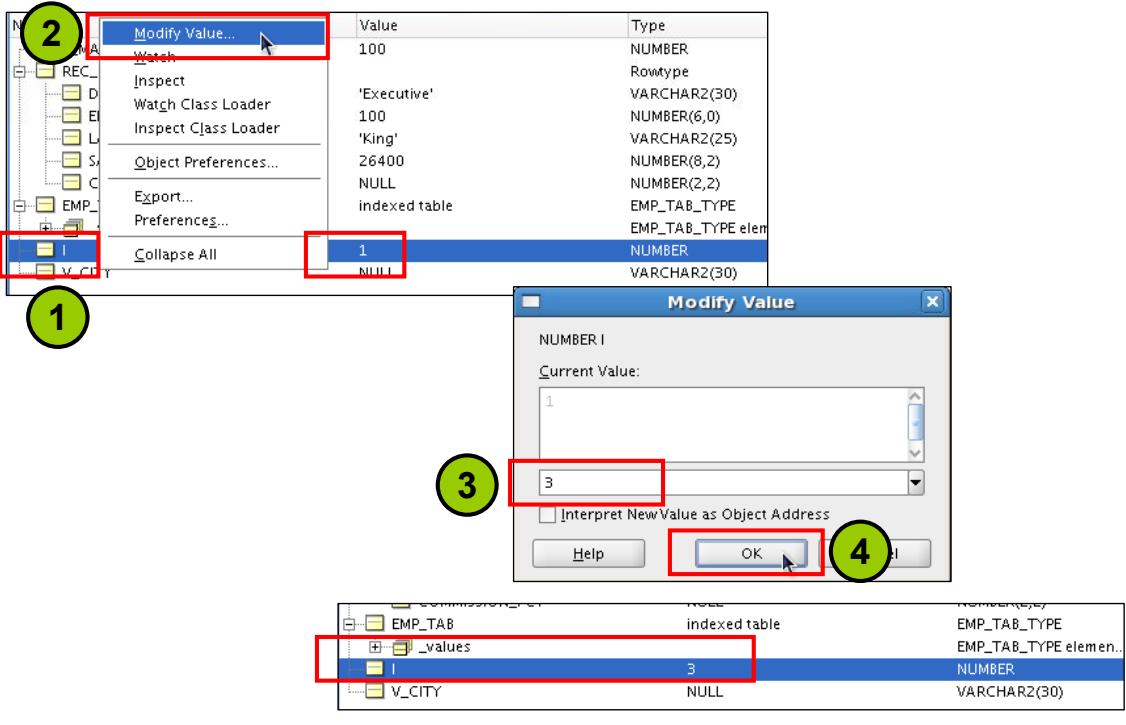
ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización de Datos

Mientras depura el código, puede utilizar el separador Data para mostrar y modificar las variables. También puede definir comprobaciones para supervisar un subjuego de variables mostradas en el separador Data. Para mostrar u ocultar los separadores Data, Smart Data y Watch: seleccione View > Debugger y, a continuación, seleccione los separadores que desee mostrar u ocultar.

Modificación de Variables al Depurar el Código



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Modificación de Variables al Depurar el Código

Para modificar el valor de una variable en el separador Data, haga clic con el botón derecho en el nombre de la variable y, a continuación, seleccione Modify Value en el menú de acceso directo. Aparece la ventana Modify Value. Aparece el valor actual de la variable. Puede introducir un nuevo valor en el segundo cuadro de texto y, a continuación, haga clic en OK.

Depuración de emp_list: Ejecución de Step Over en el Código

```

14 BEGIN
15   OPEN cur_emp;
16   FETCH cur_emp INTO rec_emp;
17   emp_tab(i) := rec_emp;
18   WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19     i := i + 1;
20     FETCH cur_emp INTO rec_emp;
21     emp_tab(i) := rec_emp;
22     v_city := get_location (rec_emp.department_name);
23     dbms_output.put_line('Employee ' || rec_emp.last_name ||
24                           ' works in ' || v_city );

```

1 F8

Step Over (F8): ejecuta el cursor (lo mismo que F7), pero no se transfiere el control para abrir el código del cursor

```

14 BEGIN
15   OPEN cur_emp;
16   FETCH cur_emp INTO rec_emp;
17   emp_tab(i) := rec_emp;
18   WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19     i := i + 1;
20     FETCH cur_emp INTO rec_emp;
21     emp_tab(i) := rec_emp;
22     v_city := get_location (rec_emp.department_name);
23     dbms_output.put_line('Employee ' || rec_emp.last_name ||
24                           ' works in ' || v_city );

```

2 F8

```

14 BEGIN
15   OPEN cur_emp;
16   FETCH cur_emp INTO rec_emp;
17   emp_tab(i) := rec_emp;
18   WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19     i := i + 1;
20     FETCH cur_emp INTO rec_emp;
21     emp_tab(i) := rec_emp;
22     v_city := get_location (rec_emp.department_name);
23     dbms_output.put_line('Employee ' || rec_emp.last_name ||
24                           ' works in ' || v_city );

```

3 F8

ORACLE

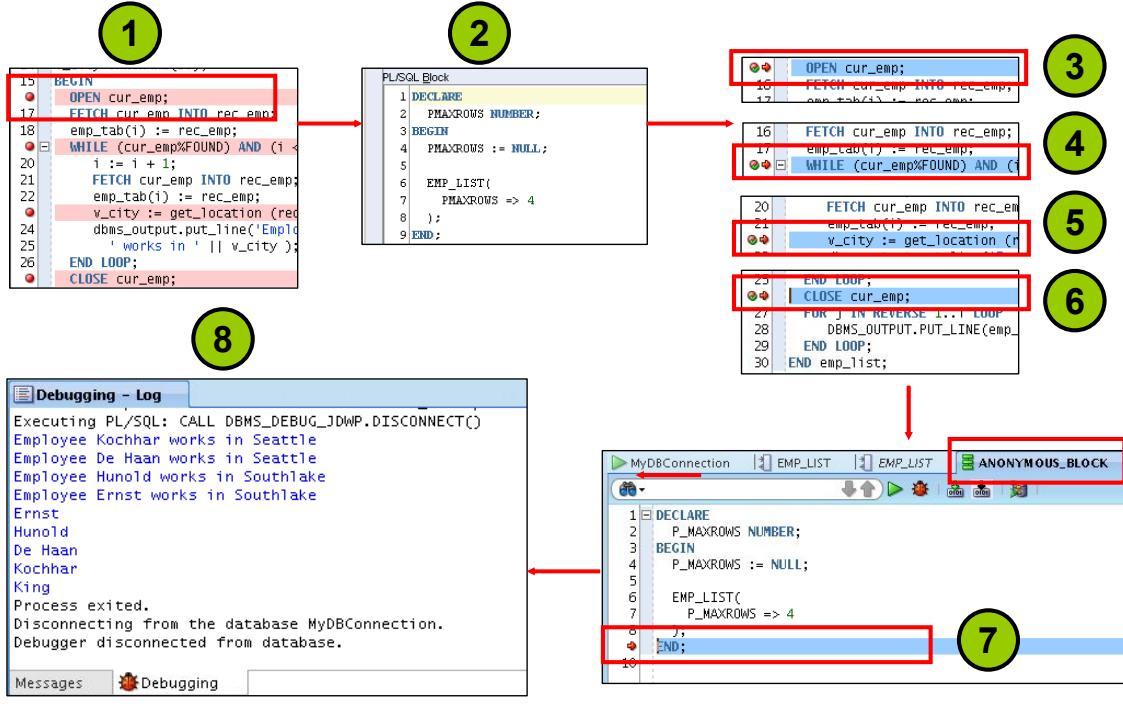
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Herramienta de Depuración Step Over

El comando Step Over, al igual que Step Into, permite ejecutar sentencias del programa a la misma vez. Sin embargo, si ejecuta el comando Step Over cuando el punto de ejecución se ubica en una llamada de subprograma, el depurador ejecuta dicho subprograma sin parar (en lugar de ir a la primera línea ejecutable del mismo) y, a continuación, posiciona el punto de ejecución en la sentencia que sigue a la llamada de subprograma. Si el punto de ejecución se ubica en la última sentencia de un subprograma, la selección de Step Over provoca que el depurador vuelva del subprograma, colocando el punto de ejecución en la línea del código que sigue a la llamada al subprograma desde el que vuelve. Puede ir a la siguiente línea de código del subprograma de cualquiera de las siguientes formas: seleccione Debug > Step Over, pulse F8 o haga clic en el icono Step Over en la barra de herramientas Debugging – Log.

En el ejemplo de la diapositiva, el comando Step Over ejecutará la línea del cursor abierta sin transferir el control de programa a la definición del cursor, como ocurría en el ejemplo de la opción Step Into.

Depuración de emp_list: Ejecución Step Out del Código (Mayús + F7)



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Depuración de emp_list: Ejecución de Step Out del Código (Mayús + F7)

La opción Step Out del código deja el subprograma actual y va al subprograma de la siguiente sentencia. En el ejemplo de la diapositiva, empieza en el paso 3, si pulsa Mayús + F7, el control de programa deja el procedimiento y va a la siguiente sentencia en el bloque anónimo. Si continúa pulsando Mayús + F7, le llevará al siguiente bloque anónimo que imprime el contenido del buffer SQL.

Depuración de emp_list: Run to Cursor (F4)

The screenshot shows the Oracle SQL Developer interface. On the left is the PL/SQL editor with the code for the emp_list procedure. A red box highlights the line 'OPEN emp_cursor;' where a breakpoint is set. Another red box highlights the line 'DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);'. The Debug menu is open, and the 'Run to Cursor F4' option is selected, also highlighted with a red box. To the right is the Watch window showing variables like PMAXROWS, EMP_RECORD, and EMP_TAB.

```

10  emp_record emp_cursor%ROWTYPE;
11  TYPE emp_tab_type IS TABLE OF emp_cursor%ROWTYPE INDEX BY BINARY_INTEGER;
12  emp_tab emp_tab_type;
13  i NUMBER := 1;
14  v_city VARCHAR2(30);
15  BEGIN
16      OPEN emp_cursor;
17      FETCH emp_cursor INTO emp_record;
18      emp_tab(i) := emp_record;
19      WHILE (emp_cursor%FOUND) AND (i <= pMaxRows) LOOP
20          i := i + 1;
21          FETCH emp_cursor INTO emp_record;
22          emp_tab(i) := emp_record;
23          v_city := get_location (emp_record.department_name);
24          dbms_output.put_line('Employee ' || emp_record.last_name ||
25                                ' works in ' || v_city );
26      END LOOP;
27      CLOSE emp_cursor;
28
29      FOR j IN REVERSE 1..i LOOP
30          DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
31      END LOOP;
32  END emp_list;

```

Name	Value	Type
PMAXROWS	5	NUMBER
EMP_RECORD	Rowtype	
DEPARTMENT_NAME	'Administration'	VARCHAR2
EMPLOYEE_ID	200	NUMBER(6,0)
LAST_NAME	'Whealen'	VARCHAR2
SALARY	4400	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)
EMP_TAB	indexed table	EMP_TAB...
I	1	NUMBER
V_CITY	NULL	VARCHAR2

Run to Cursor F4:
ir a la posición del cursor
sin necesidad de ir una a una
o definir un punto de división.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

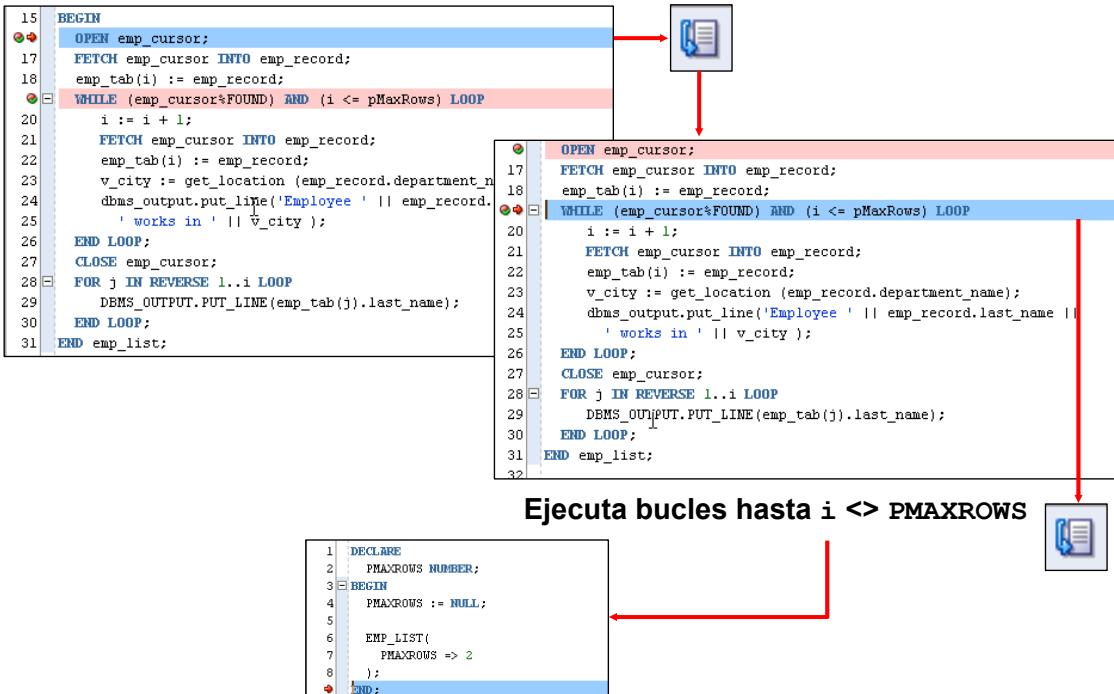
Desplazamiento a la Posición del Cursor

Si se desplaza por el código de aplicación en el depurador, puede ir a una ubicación determinada sin necesidad de ir una a una o definir un punto de división. Para ir a una ubicación de programa específica: en un editor de subprogramas, coloque el cursor de texto en la línea del código en la que deseé que se pare el depurador. Puede ir a la ubicación del cursor mediante uno de los siguientes procedimientos: en el editor de procedimientos, haga clic con el botón derecho y seleccione Run to Cursor y, a continuación, la opción Debug > Run to Cursor del menú principal o pulse F4.

Puede dar lugar a cualquiera de las siguientes condiciones:

- Al ir a la posición del cursor, el programa se ejecuta sin parar, hasta que la ejecución alcanza la ubicación marcada por el cursor de texto en el editor de origen.
- Si el programa nunca ejecuta realmente la línea del código en el que está el cursor de texto, el comando Run to Cursor hará que el programa se ejecute hasta que encuentre un punto de división o hasta que finalice.

Depuración de emp_list: Step to End of Method



ORACLE

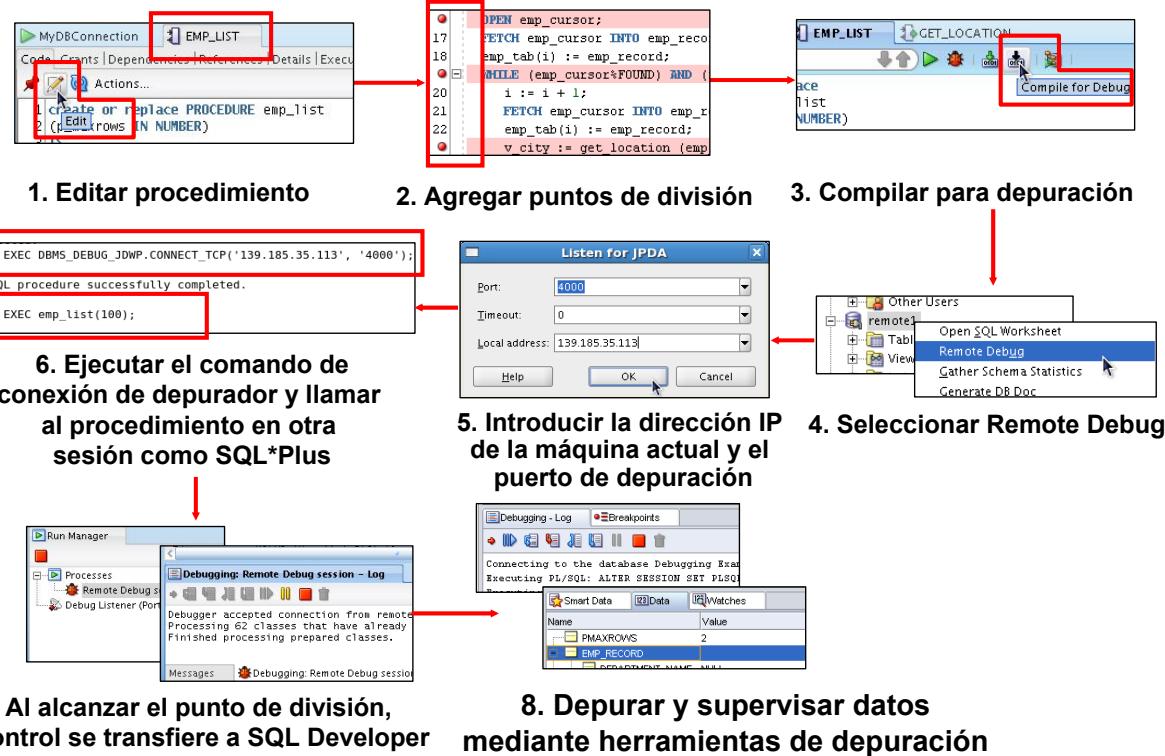
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Depuración de emp_list: Step to End of Method

Step to End of Method va a la última sentencia en el subprograma actual o al siguiente punto de división si lo hay en el subprograma actual.

En el ejemplo de la diapositiva, se utiliza la herramienta de depuración Step to End of Method. Debido a que hay un segundo punto de división, al seleccionar Step to End of Method se transfiere el control a dicho punto de división. Si vuelve a seleccionar Step to End of Method, se revisan las iteraciones del bucle while en primer lugar y, a continuación, se transfiere el control de programa a la siguiente sentencia ejecutable en el bloque anónimo.

Depuración de Subprogramas Remotamente: Visión General



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Depuración Remota

Puede utilizar la depuración remota para conectarse a cualquier código PL/SQL en una base de datos y depurarlo, independientemente de dónde se ubique la base de datos, siempre y cuando tenga acceso a la misma y haya creado una conexión a la misma. La depuración remota es cuando algo, distinto del usuario como desarrollador, inicia un procedimiento que puede realizar la depuración. La depuración remota y local tiene muchos pasos en común. Para depurar de forma remota, realice los siguientes pasos:

1. Edite el subprograma que desee depurar.
2. Agregue el punto de división en el subprograma.
3. Haga clic en el icono **Compile for Debug** en la barra de herramientas.
4. Haga clic con el botón derecho en la conexión para la base de datos remota y seleccione **Remote Debug**.
5. En el cuadro de diálogo **Debugger - Attach to JPDA**, introduzca la dirección IP de la máquina local y el número de puerto, 4000 y, a continuación, haga clic en **OK**.
6. Ejecute el comando de conexión de depurador mediante una sesión diferente como SQL*Plus y, a continuación, llame al procedimiento en dicha sesión.
7. Al alcanzar un punto de división, el control se vuelve a transferir a la sesión de SQL Developer original y aparece la barra de herramientas del depurador.
8. Depure el subprograma y supervise los datos mediante las herramientas de depuración tratadas anteriormente.

Visión General de la Práctica 2-2: Introducción sobre el Depurador de SQL Developer

En esta práctica se abordan los siguientes temas:

- Creación de un procedimiento y una función
- Inserción de puntos de división en el procedimiento
- Compilación del procedimiento y de la función para el modo de depuración
- Depuración del procedimiento y desplazamiento a la primera línea ejecutable en el código
- Visualización y modificación de las variables del subprograma



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica 2-2: Visión General

Debe utilizar SQL Developer para esta práctica. Esta práctica ofrece una introducción a la funcionalidad básica del depurador de SQL Developer.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Diferenciar entre un procedimiento y una función
- Describir los usos de las funciones
- Crear funciones almacenadas
- Llamar a una función
- Eliminar una función
- Comprender la funcionalidad básica del depurador de SQL Developer



Copyright © 2010, Oracle y/o sus filiales. 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.

3 Creación de Paquetes

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos de la Lección

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

Agenda de la Lección

- Identificación de ventajas y componentes de paquetes
- Trabajar con paquetes:
 - Creación del cuerpo y especificación del paquete
 - Llamada a subprogramas del paquete
 - Eliminación de paquetes
 - Visualización de la información del paquete

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

¿Que Son los Paquetes PL/SQL?

- Un paquete es un objeto de esquema que agrupa tipos PL/SQL, variables y subprogramas relacionados de forma lógica.
- Normalmente, los paquetes constan de dos partes:
 - Una especificación
 - Un cuerpo
- La especificación es la interfaz para el paquete. Declara los tipos, variables, constantes, excepciones, cursos y subprogramas a los que se puede hacer referencia desde fuera del paquete.
- El cuerpo define las consultas para los cursos y el código para los subprogramas.
- Permiten que el servidor de Oracle lea varios objetos en memoria a la vez.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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.

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



Copyright © 2010, Oracle y/o sus filiales. 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 programación relacionadas 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. En lo sucesivo, 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 implantación sin tener que recompilar los programas de llamada. Además, ocultando los detalles de la implantación a los usuarios, se protege la integridad del paquete.

Ventajas del Uso de Paquetes

- Funcionalidad adicional: persistencia de variables y cursores públicos
- 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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Ventajas del Uso de Paquetes (continuación)

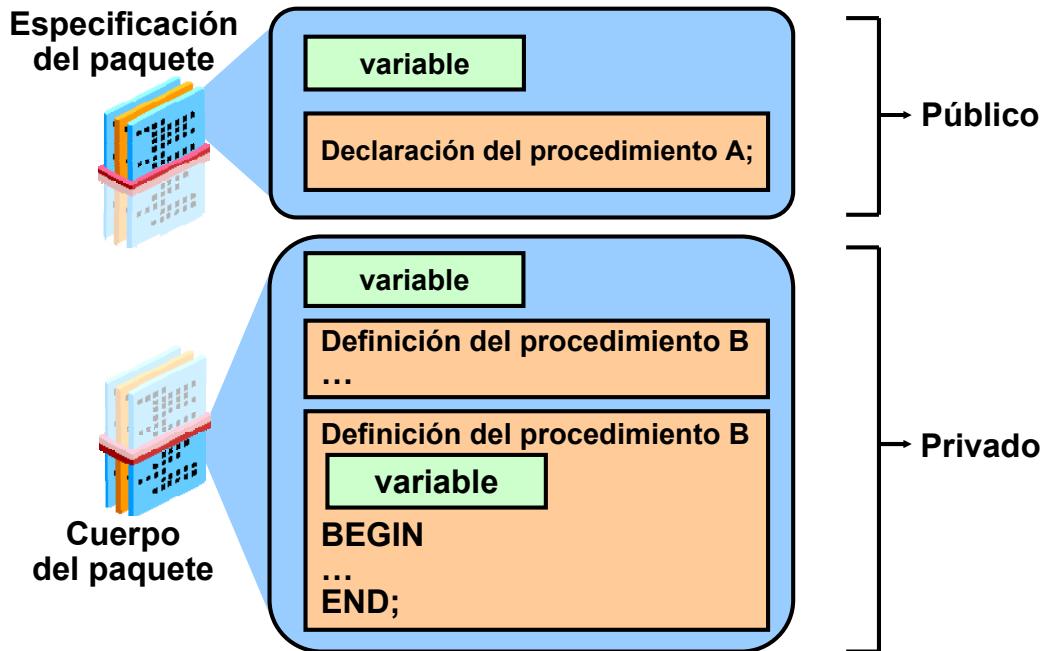
Funcionalidad adicional: los cursores y variables públicos empaquetados se mantienen durante una sesión. Por lo tanto, los pueden compartir todos los subprogramas que se ejecutan en el entorno. También le permiten mantener datos entre transacciones sin tener que almacenarlos en la base de datos. Las 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 operaciones de E/S del disco. Los subprogramas empaquetados también pueden parar dependencias en cascada y, de esta forma, 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”.

Componentes de un Paquete PL/SQL



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Componentes de un Paquete PL/SQL

Un paquete tiene dos partes:

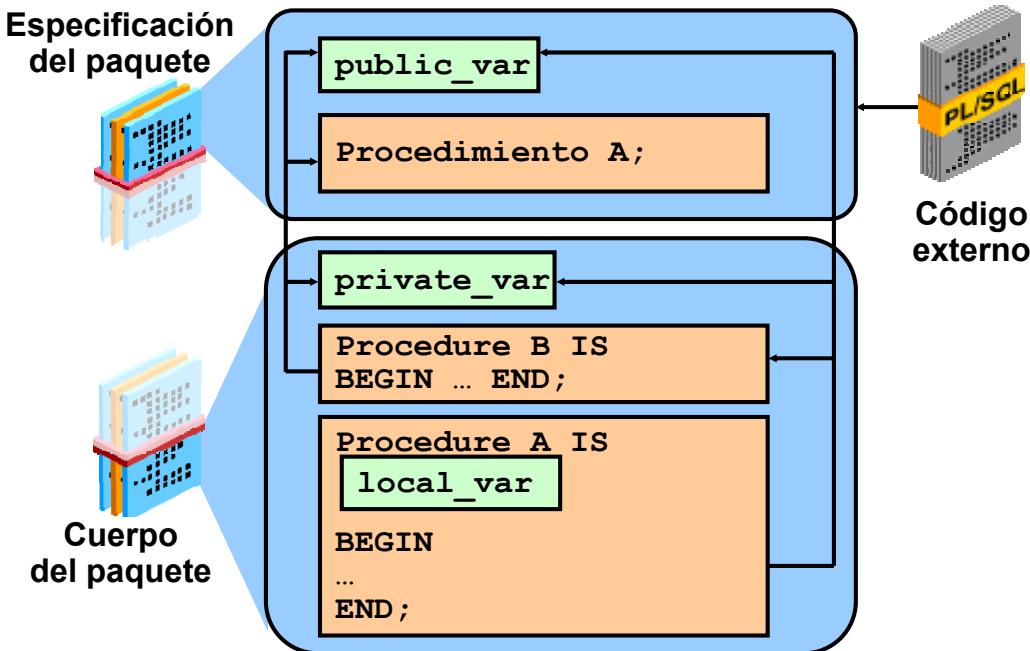
- La **especificación del paquete** es la interfaz para las aplicaciones. Declara los tipos, variables, constantes, excepciones, cursoras y subprogramas públicos disponibles para su uso. La especificación del paquete puede incluir también valores PRAGMA, que son directivas para el compilador.
- El **cuerpo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

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 Interna y Externa de los Componentes de un Paquete



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visibilidad Interna y Externa de los Componentes de un Paquete

La visibilidad de un componente describe si éste se puede ver, es decir, si otros componentes u objetos pueden hacer referencia a él y utilizarlo. 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.

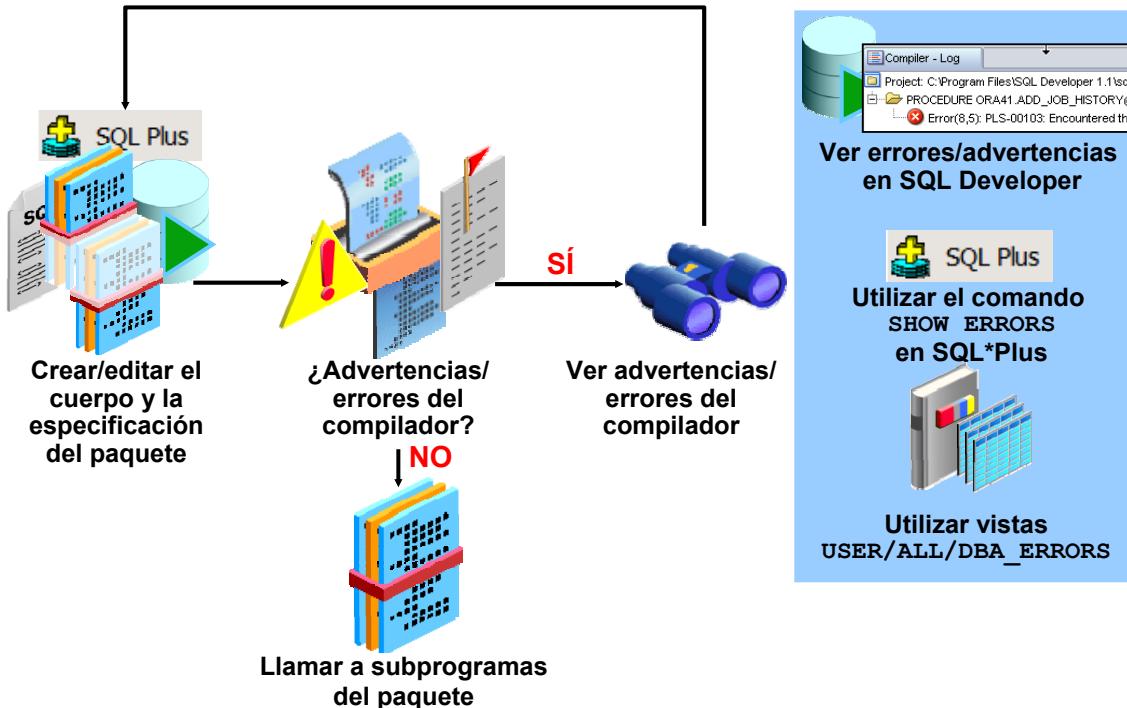
Los componentes que se declaran globalmente son visibles interna y externamente en el paquete, de esta manera:

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).

- 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`, o bien con otras construcciones de paquete privadas. Una variable pública declarada en la especificación del paquete es una variable global.

Desarrollo de Paquetes PL/SQL: Visión General



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Desarrollo de Paquetes PL/SQL

En el gráfico de la diapositiva se ilustran los pasos básicos del desarrollo y uso de un paquete:

1. Cree el procedimiento mediante el árbol Object Navigator de SQL Developer en el área SQL Worksheet.
2. Compile el paquete. El paquete se crea en la base de datos. La sentencia CREATE PACKAGE crea y almacena el código fuente y el valor *m-code* compilado en la base de datos. Para compilar el paquete, haga clic con el botón derecho en el nombre del paquete en el árbol Object Navigator y, a continuación, haga clic en Compile.
3. Si no hay ningún error o advertencia de compilación, ejecute cualquier construcción pública dentro de la especificación de paquete desde un entorno de servidor de Oracle.
4. Si hay errores o advertencias de compilación, puede ver (y, a continuación, corregir) las advertencias o errores mediante uno de los siguientes métodos:
 - Uso de la interfaz de SQL Developer (separador Compiler – Log)
 - Uso del comando de SQL*Plus SHOW ERRORS
 - Uso de las vistas USER/ALL/DBA_ERRORS

Agenda de la Lección

- Identificación de ventajas y componentes de paquetes
- Trabajar con paquetes:
 - Creación del cuerpo y de la especificación del paquete
 - Llamada a subprogramas del paquete
 - Eliminación de paquetes
 - Visualización de la información del paquete

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de Especificación del Paquete: Mediante la Sentencia CREATE PACKAGE

```
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 del paquete son visibles para los usuarios con privilegios en el paquete.



Copyright © 2010, Oracle y/o sus filiales. 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.

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 implantación de un procedimiento o de una función declarados en la especificación de un paquete se realiza en el cuerpo del paquete.

Oracle Database almacena la especificación y el cuerpo de un paquete por separado. Esto permite cambiar la implantación 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: mediante SQL Developer



ORACLE

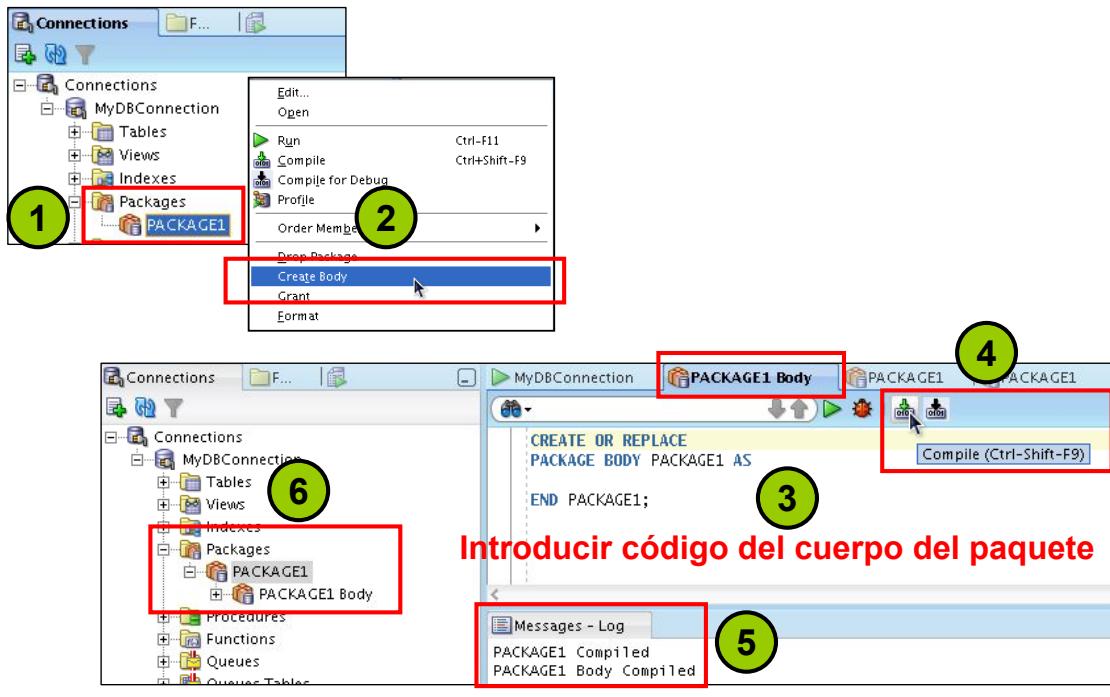
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de la Especificación del Paquete: mediante SQL Developer

Puede utilizar SQL Developer para crear la especificación del paquete de la forma siguiente:

1. Haga clic con el botón derecho en el nodo **Packages** del árbol de navegación **Connections**.
2. Seleccione **New Package** en el menú de acceso directo.
3. En la ventana **Create PL/SQL Package**, seleccione el nombre del esquema, introduzca el nombre del nuevo paquete y, a continuación, haga clic en **OK**. Aparece un separador para el nuevo paquete junto al shell del nuevo paquete.
4. Introduzca el código para el nuevo paquete.
5. Compile o guarde (mediante el icono Save de la barra de herramientas principal) el nuevo paquete.
6. El separador **Messages – Log** muestra si la compilación se ha realizado correctamente.
7. El paquete recién creado se muestra en el nodo **Packages** del árbol de navegación **Connections**.

Creación del Cuerpo del Paquete: mediante SQL Developer



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

ORACLE

Creación del Cuerpo del Paquete: mediante SQL Developer

Puede utilizar SQL Developer para crear el cuerpo del paquete, de la forma siguiente:

1. Haga clic con el botón derecho en el nombre del paquete para el que está creando un cuerpo en el nodo **Packages** del árbol de navegación **Connections**.
2. Seleccione **Create Body** en el menú de acceso directo. Aparece un separador para el nuevo cuerpo del paquete, junto al shell del nuevo cuerpo.
3. Introduzca el código para el nuevo cuerpo.
4. Compile o guarde el cuerpo del paquete.
5. El separador **Messages – Log** muestra si la compilación se ha realizado correctamente.
6. El cuerpo del paquete recién creado se muestra en el nodo **Packages** del árbol de navegación **Connections**.

Ejemplo de Especificación de un Paquete: comm_pkg

```
-- The package spec with a public variable and a
-- public procedure that are accessible from
-- outside the package.

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

- `V_STD_COMM` es una variable *pública* 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 implanta en el cuerpo del paquete.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Ejemplo de Especificación de un Paquete: comm_pkg

En el ejemplo de la diapositiva se crea un paquete llamado `comm_pkg`, que se utiliza para gestionar reglas de procesamiento de negocios para el cálculo de comisiones.

La variable `v_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

```
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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación del Cuerpo del Paquete

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

- 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 implantación de todos los procedimientos o funciones declarados 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.
- **private type and variable declarations**: declara tipos y subtipos definidos por el usuario, excepciones, cursor, constantes y variables privadas.
- **subprogram specifications**: especifica la implantación 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(p_comm NUMBER) RETURN BOOLEAN IS
        v_max_comm    employees.commission_pct%type;
    BEGIN
        SELECT MAX(commission_pct) INTO v_max_comm
        FROM   employees;
        RETURN (p_comm BETWEEN 0.0 AND v_max_comm);
    END validate;

    PROCEDURE reset_comm (p_new_comm NUMBER) IS
    BEGIN
        IF validate(p_new_comm) THEN
            v_std_comm := p_new_comm; -- reset public var
        ELSE
            RAISE_APPLICATION_ERROR(
                -20210, 'Bad Commission');
        END IF;
    END reset_comm;
END comm_pkg;

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Ejemplo del Cuerpo del Paquete: comm_pkg

En la diapositiva se 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 menor 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 v_std_comm. En el ejemplo, tenga en cuenta lo siguiente:

- La variable v_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 v_std_comm, se les puede hacer referencia directamente sin cualificación.
- El procedimiento reset_comm implanta 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 que el procedimiento reset_comm, porque 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, 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: Ejemplos

```
-- Invoke a function within the same packages:
CREATE OR REPLACE PACKAGE BODY comm_pkg IS ...
  PROCEDURE reset_comm(p_new_comm NUMBER) IS
  BEGIN
    IF validate(p_new_comm) THEN
      v_std_comm := p_new_comm;
    ELSE ...
    END IF;
  END reset_comm;
END comm_pkg;
```

```
-- Invoke a package procedure from SQL*Plus:
EXECUTE comm_pkg.reset_comm(0.15)
```

```
-- Invoke a package procedure in a different schema:
EXECUTE scott.comm_pkg.reset_comm(0.15)
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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, sino opcional.

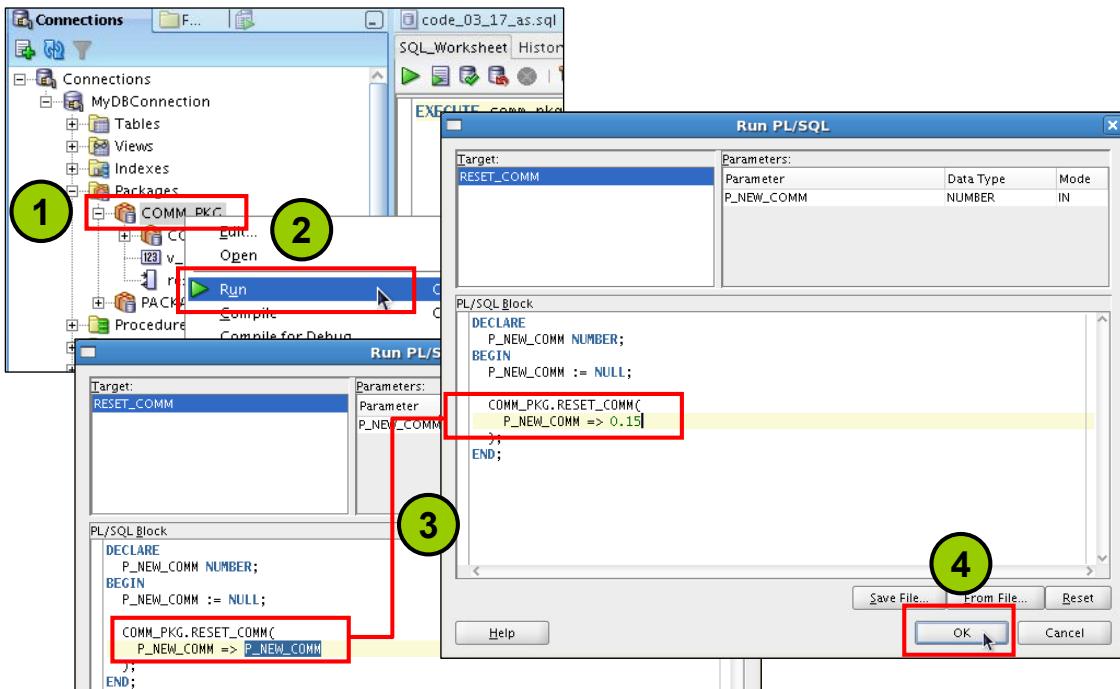
Ejemplo 2: llama al procedimiento `reset_comm` desde SQL*Plus (un entorno que está fuera del paquete) para restablecer la comisión predominante en 0,15 para la sesión de usuario.

Ejemplo 3: llama al procedimiento `reset_comm` que es propiedad de un usuario de esquema llamado SCOTT. Con SQL*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)
```

Llamada a Subprogramas de Paquete: mediante SQL Developer



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Llamada a Subprogramas de Paquete: mediante SQL Developer

Puede utilizar SQL Developer para llamar al subprograma del paquete de la forma siguiente:

1. Haga clic con el botón derecho en el nodo Packages del árbol de navegación.
2. Seleccione **Run** en el menú flotante. Se muestra la ventana **Run PL/SQL**. Puede utilizar la ventana **Run PL/SQL** para especificar los valores del parámetro para ejecutar un procedimiento o función PL/SQL. (Si especifica un paquete, seleccione una función o procedimiento del paquete.) Especifique lo siguiente:
 - a. **Target:** seleccione el nombre de la función o procedimiento que deseé ejecutar.
 - b. **Parameters:** esta sección muestra cada uno de los parámetros del destino especificado. El modo de cada parámetro puede ser **IN** (se transfiere el valor), **OUT** (se devuelve el valor) o **IN/OUT** (se transfiere el valor y el resultado de la acción de la función o del procedimiento se almacena en el parámetro).
3. En la sección **PL/SQL Block**, cambie las especificaciones de los parámetros formales **IN** e **IN/OUT** en este bloque a los valores reales que desea utilizar para ejecutar la función o el procedimiento. Por ejemplo, para especificar 0.15 como el valor para un parámetro de entrada denominado **P_NEW_COMM**, cambie **P_NEW_COMM => P_NEW_COMM** a **P_NEW_COMM => 0.15**.
4. Haga clic en **OK**. SQL Developer ejecuta la función o el procedimiento.

Creación y Uso de Paquetes sin Cuerpo

```
CREATE OR REPLACE PACKAGE global_consts IS
    c_mile_2_kilo CONSTANT NUMBER := 1.6093;
    c_kilo_2_mile CONSTANT NUMBER := 0.6214;
    c_yard_2_meter CONSTANT NUMBER := 0.9144;
    c_meter_2_yard CONSTANT NUMBER := 1.0936;
END global_consts;
```

```
SET SERVEROUTPUT ON
BEGIN
    DBMS_OUTPUT.PUT_LINE('20 miles = ' ||
        20 * global_consts.c_mile_2_kilo || ' km');
END;
```

```
SET SERVEROUTPUT ON
CREATE FUNCTION mtr2yrd(p_m NUMBER) RETURN NUMBER IS
BEGIN
    RETURN (p_m * global_consts.c_meter_2_yard);
END mtr2yrd;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(mtr2yrd(1))
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 del paquete sin un cuerpo de paquete, a lo que se conoce como *paquete sin cuerpo*. Tal como se ha descrito anteriormente en esta lección, si una especificación sólo declara tipos, constantes, variables y excepciones, el cuerpo del paquete no es necesario.

Ejemplos

En el primer cuadro de código de la diapositiva se 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. Se asume que se ha ejecutado la sentencia `SET SERVEROUTPUT ON` antes de ejecutar los ejemplos de código de la diapositiva.

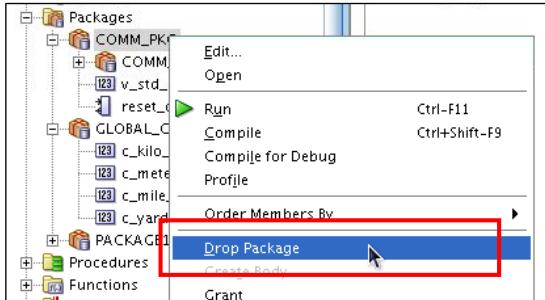
En el segundo cuadro de código se hace referencia a la constante `c_mile_2_kilo` en el paquete `global_consts` agregando el nombre del paquete como prefijo al identificador de la constante.

En el tercer ejemplo se crea una función autónoma `c_mtr2yrd` para convertir metros en yardas y se utiliza el ratio de conversión de constantes `c_meter_2_yard` declarado en el paquete `global_consts`. La función se llama en un parámetro `DBMS_OUTPUT.PUT_LINE`.

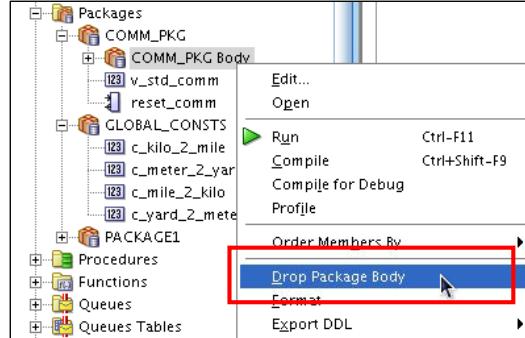
Regla que se debe seguir: cuando se hace referencia a una variable, cursor, constante o excepción desde fuera de un paquete, debe cualificarlos con el nombre del paquete.

Eliminación de Paquetes: mediante la sentencia SQL DROP o SQL Developer

Borrar especificación y cuerpo del paquete



Borrar sólo cuerpo del paquete



```
-- Remove the package specification and body
DROP PACKAGE package_name;
```

```
-- Remove the package body only
DROP PACKAGE BODY package_name;
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Eliminación de Paquetes

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

Visualización de Paquetes mediante Diccionario

```
-- View the package specification.
SELECT text
FROM user_source
WHERE name = 'COMM_PKG' AND type = 'PACKAGE'
ORDER BY LINE;
```

TEXT
1 PACKAGE comm_pkg IS
2 std_comm NUMBER := 0.10; --initialized to 0.10
3 PROCEDURE reset_comm(new_comm NUMBER);
4 END comm_pkg;

```
-- View the package body.
SELECT text
FROM user_source
WHERE name = 'COMM_PKG' AND type = 'PACKAGE BODY'
ORDER BY LINE;
```

TEXT
1 PACKAGE BODY comm_pkg IS
2 FUNCTION validate(comm NUMBER) RETURN BOOLEAN IS
3 max_comm employees.commission_pct%type;
4 BEGIN
5 SELECT MAX(commission_pct) INTO max_comm
6 FROM employees;
7 RETURN (comm BETWEEN 0.0 AND max_comm);

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización de Paquetes en el Diccionario de Datos

El código fuente de los paquetes PL/SQL también se almacena en las vistas de diccionario de datos USER_SOURCE y ALL_SOURCE. La tabla USER_SOURCE se utiliza para mostrar el código PL/SQL del que es propietario. La tabla ALL_SOURCE se utiliza para mostrar el código PL/SQL para el que el propietario de dicho código de subprograma le ha otorgado el derecho EXECUTE por el propietario de dicho código del subprograma y proporciona la columna OWNER además de las columnas anteriores.

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

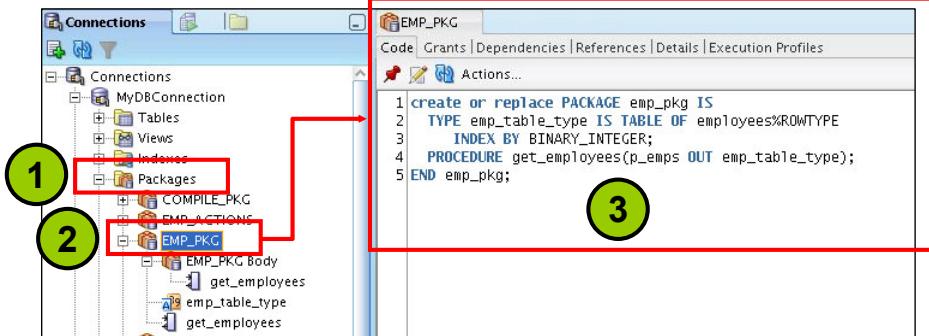
- Igual a 'PACKAGE' para mostrar el código fuente de la especificación del paquete.
- Igual a 'PACKAGE BODY' para mostrar el código fuente del cuerpo del paquete.

También puede ver la especificación y el cuerpo del paquete en SQL Developer mediante el nombre del paquete y el nodo Packages.

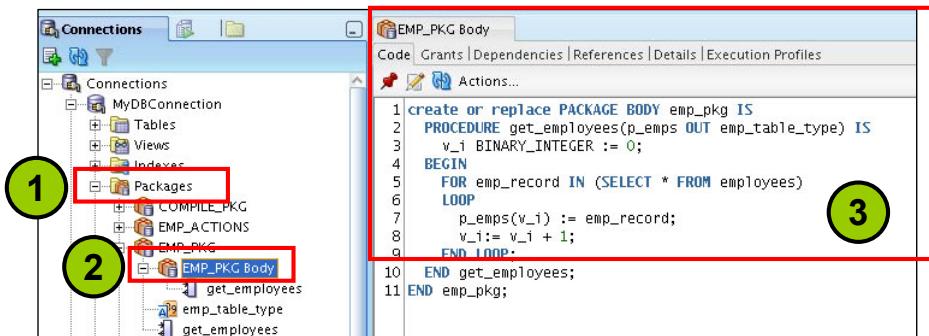
Nota: no puede mostrar el código fuente de los paquetes integrados PL/SQL de Oracle ni PL/SQL cuyo código fuente se ha ocultado mediante una utilidad WRAP o una ocultación. La ocultación y encapsulamiento del código fuente PL/SQL se tratará en una lección posterior. Al hacer clic en el icono Execute Statement (F9) (en lugar de en el icono Run Script) en la barra de herramientas de SQL Worksheet, a veces se muestra una salida con un formato mejor, como se muestra en los ejemplos de la diapositiva.

Visualización de Paquetes mediante SQL Developer

Para ver la especificación del paquete, haga clic en el nombre del paquete



Para ver el cuerpo del paquete, haga clic en el cuerpo del paquete



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización de Paquetes mediante SQL Developer

Para ver la especificación del paquete en SQL Developer, realice los siguientes pasos:

1. Haga clic en el nodo **Packages** del separador **Connections**.
2. Haga clic en el nombre del paquete.
3. El código de especificación del paquete se muestra en el separador **Code**, como aparece en la diapositiva.

Para ver el cuerpo del paquete en SQL Developer, realice los siguientes pasos:

1. Haga clic en el nodo **Packages** del separador **Connections**.
2. Haga clic en el cuerpo del paquete.
3. El código del cuerpo del paquete se muestra en el separador **Code**, como aparece en la diapositiva.

Instrucciones para la Escritura de Paquetes

- Desarrollar 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.
- Coloque elementos en la parte de la declaración del cuerpo del paquete cuando deba mantenerlos a lo largo de una sesión o entre transacciones.
- La gestión de dependencias detalladas reduce la necesidad de recompilar subprogramas de referencia si la especificación del paquete cambia.
- La especificación del paquete debe contener tan pocas construcciones como sea posible.



Copyright © 2010, Oracle y/o sus filiales. 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; defínalas 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 lo 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.

Antes de Oracle Database 11g, 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. Oracle Database 11g reduce esta dependencia. El seguimiento de las dependencias ahora se realiza a nivel de elemento dentro de una unidad. La gestión de dependencias detalladas se tratará en una lección posterior.

Prueba

La especificación del paquete es la interfaz para las aplicaciones. Declara los tipos, variables, constantes, excepciones, cursores y subprogramas públicos disponibles para su uso. La especificación del paquete puede incluir también PRAGMAS, que son directivas para el compilador.

1. Verdadero
2. Falso

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: 1

Resumen

En esta lección, debe haber aprendido 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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Resumen

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

Un paquete está compuesto por una especificación de paquete y un cuerpo de 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 fuente. Al llamar por primera vez a un paquete, el paquete completo se carga en memoria. Esto reduce el acceso al disco para llamadas posteriores.

Visión General de la Práctica 3: Creación y Uso de Paquetes

En esta práctica se abordan los siguientes temas:

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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica 3: Visión General

En esta práctica, creará cuerpos y especificaciones de paquetes. A continuación, llamará a las construcciones de los paquetes con datos de ejemplo sencillos.

Nota: si utiliza SQL Developer, los errores de tiempo de compilación se muestran en el separador Message Log. Si utiliza SQL*Plus para crear el código almacenado, utilice SHOW ERRORS para ver los errores de compilación.

4

Trabajar con Paquetes

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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
- Utilizar matrices asociativas (tablas de índice) y registros en paquetes



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos de la Lección

En esta lección se presentan las funciones más avanzadas de PL/SQL, incluida la sobrecarga, la referencia anticipada, los procedimientos que se realizan sólo una vez y la persistencia de las variables, constantes, excepciones y cursos. También se explica el efecto de las funciones de empaquetado que se utilizan en sentencias SQL.

Agenda de la Lección

- Sobrecarga de subprogramas de paquetes mediante declaraciones anticipadas y creación de bloques de inicialización en un cuerpo del paquete
- Gestión de estados de datos de paquete persistentes mientras dure una sesión y uso de matrices asociativas (tablas de índice) y registros en paquetes

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Sobrecarga de Subprogramas 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 familia de tipos 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; es decir, agregando nuevos parámetros a los subprogramas existentes
- Proporciona una forma de sobrecargar subprogramas locales, subprogramas de paquete y métodos de tipo, pero no subprogramas autónomos



Copyright © 2010, Oracle y/o sus filiales. 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 desee que un subprograma acepte juegos similares de parámetros con diferentes tipos de dato. Por ejemplo, la función TO_CHAR se puede llamar de más de una forma, lo que permite 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 familia de tipos de dato*.

Plantéese 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, es posible que desee buscar empleados por su identificador de empleado, así como proporcionar una forma de buscar empleados por apellido. La lógica es esencialmente la misma, pero los parámetros o criterios de búsqueda varían.
- 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.

Sobrecarga de Subprogramas (continuación)

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 aunque los tipos sean 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 los parámetros, 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.

Ejemplo de Sobrecarga de Procedimientos: Creación de la Especificación del Paquete

```

CREATE OR REPLACE PACKAGE dept_pkg IS
  PROCEDURE add_department
    (p_deptno departmentsdepartment_id%TYPE,
     p_name departmentsdepartment_name%TYPE := 'unknown',
     p_loc   departmentslocation_id%TYPE := 1700);

  PROCEDURE add_department
    (p_name departmentsdepartment_name%TYPE := 'unknown',
     p_loc   departmentslocation_id%TYPE := 1700);
END dept_pkg;
/

```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Sobrecarga: Ejemplo

En esta diapositiva se 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.

Es mejor especificar tipos de dato utilizando el atributo `%TYPE` para variables que se utilizan con el fin de llenar columnas en las tablas de base de datos, como se muestra en el ejemplo de la diapositiva; sin embargo, también puede especificar los tipos de dato de la siguiente forma:

```

CREATE OR REPLACE PACKAGE dept_pkg_method2 IS
  PROCEDURE add_department (p_deptno NUMBER,
                            p_name VARCHAR2 := 'unknown', p_loc NUMBER := 1700);

```

Ejemplo de Sobrecarga de Procedimientos: Creación del Cuerpo del Paquete

```
-- Package body of package defined on previous slide
CREATE OR REPLACE PACKAGE BODY dept_pkg IS
PROCEDURE add_department -- First procedure's declaration
(p_deptno departmentsdepartment_id%TYPE,
 p_name   departmentsdepartment_name%TYPE := 'unknown',
 p_loc    departmentslocation_id%TYPE := 1700) IS
BEGIN
  INSERT INTO departments(department_id,
    department_name, location_id)
  VALUES (p_deptno, p_name, p_loc);
END add_department;
PROCEDURE add_department -- Second procedure's declaration
(p_name   departmentsdepartment_name%TYPE := 'unknown',
 p_loc    departmentslocation_id%TYPE := 1700) IS
BEGIN
  INSERT INTO departments (department_id,
    department_name, location_id)
  VALUES (departments_seqNEXTVAL, p_name, p_loc);
END add_department;
END dept_pkg; /
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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. Observe el siguiente ejemplo

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

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
980	Education		2500
1 rows selected			

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

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

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	Training		2400
1 rows selected			

Sobrecarga y 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 un subprograma incorporado sustituye la declaración estándar en el contexto local, a menos que se cualifique mediante el nombre del paquete

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Sobrecarga y Paquete STANDARD

Un paquete denominado STANDARD define el entorno PL/SQL y declara globalmente los tipos, las excepciones y los subprogramas que están disponibles automáticamente para los programas PL/SQL. La mayoría de las funciones incorporadas del paquete STANDARD están sobrecargadas. Por ejemplo, la función TO_CHAR tiene cuatro declaraciones diferentes, tal y como se muestra en la diapositiva. 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 STANDARDTO_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, SCOTTTO_CHAR.

En el ejemplo de la diapositiva, PL/SQL resuelve una llamada a TO_CHAR haciendo coincidir el número y los tipos de dato de los parámetros formales y reales.

Referencia a Procedimiento No Válido

- 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
    calc_rating;
  END;
END forward_pkg;
/

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 la diapositiva, donde 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 que los subprogramas se declaren en 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 una declaración anticipada y el compilador puede resolver su referencia.

Uso de Declaraciones Anticipadas para Solucionar una Referencia a Procedimiento No Válido

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 © 2010, Oracle y/o sus filiales. 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 recursivos. 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 de la especificación del paquete no necesitan declaraciones anticipadas.

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
    v_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 v_tax
    FROM      tax_rates
    WHERE     rate_name = 'TAX';
END taxes;
/

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Bloque de Inicialización de Paquetes

La primera vez que se hace referencia a un componente del paquete, el paquete 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.

En el ejemplo de la diapositiva se muestra la variable pública v_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

- Las funciones de paquete se utilizan en sentencias SQL
- Para ejecutar una sentencia SQL que llame a una función de miembro, Oracle Database debe conocer el nivel de pureza de la función
- El nivel de pureza es el grado en el que la función está libre de efectos secundarios, es decir, el acceso a tablas de base de datos, variables de paquete, etc. para su lectura o escritura
- Es importante controlar los efectos secundarios porque pueden:
 - Impedir el correcto paralelismo de una consulta
 - Producir resultados dependientes del orden y, por lo tanto, indeterminados
 - Necesar acciones no permitidas, como el mantenimiento del estado de los paquetes en las sesiones de usuario



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Funciones de Paquete en SQL y Restricciones

Para ejecutar una sentencia SQL que llame a una función almacenada, el servidor de Oracle debe conocer el nivel de pureza de la función o el grado en que ésta se encuentra libre de efectos secundarios. El término *efecto secundario* hace referencia al acceso a tablas de base de datos, variables de paquete, etc. para su lectura o escritura. Es importante controlar los efectos secundarios, porque pueden impedir el correcto paralelismo de una consulta, producir resultados dependientes del orden y, por lo tanto, indeterminados o necesitar acciones no permitidas, como el mantenimiento del estado del paquete en las sesiones de usuario.

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.

Control de Efectos Secundarios de Subprogramas PL/SQL

Para que se pueda llamar desde sentencias SQL, una función almacenada debe cumplir las siguientes reglas de pureza para controlar los efectos secundarios:

- Cuando se llama desde una sentencia SELECT o DML paralela, la función no puede modificar ninguna tabla de la base de datos
- Cuando se llamada desde una sentencia DML, la función no puede realizar consultas ni modificar ninguna tabla de la base de datos modificada por dicha sentencia
- Cuando se llama desde una sentencia SELECT o DML, la función no puede ejecutar el control de transacciones SQL, el control de sesiones ni las sentencias de control del sistema



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Control de Efectos Secundarios de Subprogramas PL/SQL

Cuantos menos efectos secundarios tenga una función, mejor se podrá optimizar en una consulta, especialmente cuando se utilizan las indicaciones PARALLEL_ENABLE o DETERMINISTIC.

Para que se pueda llamar desde sentencias SQL, una función almacenada (y los subprogramas que llamen) debe cumplir las siguientes reglas de pureza que se enumeran en la diapositiva. El objetivo de estas reglas es controlar los efectos secundarios.

Si alguna sentencia SQL del cuerpo de la función viola una regla, recibirá un mensaje de error en tiempo de ejecución (al analizar la sentencia).

Para comprobar las violaciones de las reglas de pureza en el momento de la compilación, utilice el pragma RESTRICT_REFERENCES para confirmar que una función no lee ni escribe en las tablas de bases de datos o las variables de paquetes

Nota

- En la diapositiva, una sentencia DML hace referencia a una sentencia INSERT, UPDATE o DELETE.
- Para obtener más información sobre el uso del pragma RESTRICT_REFERENCES, consulte *Oracle Database PL/SQL Language Reference* (Referencia del Lenguaje PL/SQL de Oracle Database).

Función de Paquete en SQL: Ejemplo

```

CREATE OR REPLACE PACKAGE taxes_pkg IS
    FUNCTION tax (p_value IN NUMBER) RETURN NUMBER;
END taxes_pkg;
/
CREATE OR REPLACE PACKAGE BODY taxes_pkg IS
    FUNCTION tax (p_value IN NUMBER) RETURN NUMBER IS
        v_rate NUMBER := 008;
    BEGIN
        RETURN (p_value * v_rate);
    END tax;
END taxes_pkg;
/

```

```

SELECT taxes_pkgtax(salary), salary, last_name
FROM employees;

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Función de Paquete en SQL: Ejemplo

En el primer ejemplo de código de la diapositiva se muestra cómo crear la especificación de paquete y el cuerpo que incluye la función tax en el paquete taxes_pkg. En el segundo ejemplo de código se muestra cómo llamar a la función empaquetada tax en la sentencia SELECT. Los resultados son los siguientes:

TAXES_PKG.TAX(SALARY)	SALARY	LAST_NAME
1920	24000	King
1360	17000	Kochhar
1360	17000	De Haan
720	9000	Hunold
480	6000	Ernst
384	4800	Austin
384	4800	Pataballa
336	4200	Lorentz
960	12000	Greenberg

107 rows selected

Agenda de la Lección

- Sobrecarga de subprogramas de paquetes mediante declaraciones anticipadas y creación de bloques de inicialización en un cuerpo del paquete
- Gestión de estados de datos de paquete persistentes mientras dure una sesión y uso de matrices asociativas (tablas de índice) y registros en paquetes

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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_REUSABLE en la especificación del paquete



Copyright © 2010, Oracle y/o sus filiales. 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 del área global de usuario (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.

PRAGMA significa que la sentencia es una directiva de compilador. Los PRAGMA se procesan en tiempo de compilación, no en tiempo de ejecución. No afectan al significado de un programa; simplemente transmiten información al compilador. Si agrega PRAGMA SERIALLY_RESUABLE a la especificación del paquete, la base de datos almacena las variables de paquete en el área global del sistema (SGA) compartida entre 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_REUSABLE es útil si desea conservar la memoria y si el estado del paquete no necesita persistir durante cada sesión de usuario.

Estado Persistente de Paquetes (continuación)

Este PRAGMA es adecuado para paquetes que declaran áreas de trabajo temporales de gran volumen que se utilizan una vez y que no son necesarias durante las llamadas posteriores a la base de datos en la misma sesión.

Puede marcar un paquete sin cuerpo como reutilizable en serie. Si un paquete tiene una especificación y un cuerpo, debe marcar ambos. No puede marcar sólo el cuerpo.

La memoria global para los paquetes reutilizables en serie se agrupa en el área global del sistema (SGA), no se asigna a los distintos usuarios en el área global de usuario (UGA). De esta forma, el área de trabajo de paquetes se puede reutilizar. Cuando finaliza la llamada al servidor, la memoria se devuelve al pool. Cada vez que se reutiliza el paquete, sus variables públicas se inicializan en sus valores por defecto o NULL.

Nota: no se puede acceder a los paquetes reutilizables en serie desde disparadores de base de datos u otros subprogramas PL/SQL llamados desde sentencias SQL. Si lo intenta, el servidor de Oracle genera un error.

Estado Persistente de Variables de Paquetes: Ejemplo

Hora	Eventos	v_std_comm [variable]	MAX (commission_pct) [column]	v_std_comm [variable]	MAX (commission_pct) [Column]
9:00	<i>Scott> EXECUTE comm_pkgreset_comm(025)</i>	010 025	04	-	04
9:30	<i>Jones> INSERT INTO employees(last_name, commission_pct) VALUES('Madonna', 08);</i>	025	04		08
9:35	<i>Jones> EXECUTE comm_pkgreset_comm(05)</i>	025	04	010 05	08
10:00	<i>Scott> EXECUTE comm_pkgreset_comm(06)</i> <i>Err -20210 'Bad Commission'</i>	025	04	05	08
11:00 11:01 12:00	<i>Jones> ROLLBACK;</i> <i>EXIT</i> <i>EXEC comm_pkgreset_comm(02)</i>	025 025 025	04 04 04	05 - 02	04 04 04

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Estado Persistente de Variables de Paquetes: Ejemplo

La secuencia de la diapositiva está basada en dos usuarios distintos: 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. En el ejemplo se muestra cómo el estado persistente de la variable de paquete v_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 025. El estado del paquete de v_std_comm se inicializa en 010 y, a continuación, se define en 025, el cual se valida, ya que es menor que el valor máximo de la base de datos, que es 04.

A las 09:30: Jones inserta una nueva fila en la tabla EMPLOYEES con un nuevo valor máximo de v_commission_pct de 08. Esto no se confirma, por lo que sólo Jones puede visualizarlo. El estado de Scott no se ve afectado.

A las 09:35: Jones llama a reset_comm con un nuevo valor de comisión de 05. El estado de v_std_comm de Jones se inicializa por primera vez en 010 y, a continuación, se define en el nuevo valor 05 que es válido para esta sesión con el valor máximo de la base de datos, 08.

A las 10:00: Scott llama a reset_comm con un nuevo valor de comisión de 06, que es mayor que la comisión máxima de la base de datos visible para esta sesión, 04 (Jones no ha confirmado el valor 08).

Entre las 11:00 y las 12:00: Jones realiza un rollback en la transacción (sentencia INSERT) y sale de la sesión. Jones se conecta a las 11:45 y ejecuta correctamente el procedimiento, definiendo su estado en 02.

Estado Persistente de un Cursor de Paquete: Ejemplo

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

CREATE OR REPLACE PACKAGE BODY curs_pkg IS
  -- Package body
  CURSOR cur_c IS
    SELECT employee_id FROM employees;
  PROCEDURE open IS
  BEGIN
    IF NOT cur_c%ISOPEN THEN
      OPEN cur_c;
    END IF;
  END open;
  -- code continued on next slide
```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Estado Persistente de un Cursor de Paquete: Ejemplo

En el ejemplo de la diapositiva se muestra la especificación y el cuerpo del paquete CURS_PKG. La declaración del cuerpo continúa en la siguiente diapositiva.

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

- 1 Llame al procedimiento open para abrir el cursor.

Estado Persistente de un Cursor de Paquete: Ejemplo

```

FUNCTION next(p_n NUMBER := 1) RETURN BOOLEAN IS
    v_emp_id employees.employee_id%TYPE;
BEGIN
    FOR count IN 1..p_n LOOP
        FETCH cur_c INTO v_emp_id;
        EXIT WHEN cur_c%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Id: ' || (v_emp_id));
    END LOOP;
    RETURN cur_c%FOUND;
END next;
PROCEDURE close IS
BEGIN
    IF cur_c%ISOPEN THEN
        CLOSE cur_c;
    END IF;
END close;
END curs_pkg;

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Estado Persistente de un Cursor de Paquete: Ejemplo (continuación)

- 2 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 necesita procesar más filas; de lo contrario, devuelve FALSE.
- 3 Llame al procedimiento `close` para cerrar el cursor, antes o al final del procesamiento de las filas.

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

Ejecución del Paquete CURS_PKG

```

1 SET SERVEROUTPUT ON
2
3 EXECUTE curs_pkg.open
4 DECLARE
5   v_more BOOLEAN := curs_pkg.next(3);
6 BEGIN
7   IF NOT v_more THEN
8     curs_pkg.close;
9   END IF;
10 END;
11 /

```

Results

```

anonymous block completed
anonymous block completed
Id: 100
Id: 101
Id: 102

anonymous block completed
anonymous block completed
Id: 103
Id: 104
Id: 105

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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. 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 `b_more` en TRUE, porque hay más de tres filas en la tabla `EMPLOYEES`. 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 :100
Id :101
Id :102

```

Si vuelve a hacer clic en el icono Run Script (F5), se muestran las tres filas siguientes:

```

Id :103
Id :104
Id :105

```

Para cerrar el cursor, puede emitir el siguiente comando para cerrar el cursor en el paquete o salir de la sesión:

```
EXECUTE curs_pkgclose
```

Uso de Matrices Asociativas 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(p_emps OUT emp_table_type);
END emp_pkg;
```

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

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Matrices Asociativas en Paquetes

Las matrices asociativas se denominaban tablas basadas en índices.

El paquete emp_pkg contiene un procedimiento get_employees, que lee filas de la tabla EMPLOYEES y devuelve las filas mediante el parámetro OUT, que es una matriz asociativa (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 SQL Developer, puede llamar al procedimiento get_employees en un bloque PL/SQL anónimo mediante la variable v_employees, tal y como se muestra en el siguiente ejemplo y salida:

```
SET SERVEROUTPUT ON
DECLARE
    v_employees emp_pkg.emp_table_type;
BEGIN
    emp_pkget_employees(v_employees);
    DBMS_OUTPUT.PUT_LINE('Emp 5: '||v_employees(4).last_name);
END;
anonymous block completed
Emp 5: Ernst
```

Prueba

Sobrecarga de Subprogramas en PL/SQL:

1. Permite crear dos o más subprogramas con el mismo nombre
2. Necesita que los parámetros formales del subprograma varíen en número, orden o familia de tipos de dato
3. Permite crear formas flexibles para llamar a los subprogramas con datos diferentes
4. Proporciona una forma de ampliar la funcionalidad sin perder el código existente; es decir, agregando nuevos parámetros a los subprogramas existentes

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: 1, 2, 3, 4

La función de sobrecarga en PL/SQL permite desarrollar dos o más subprogramas empaquetados con el mismo nombre. La sobrecarga es útil cuando deseé que un subprograma acepte juegos similares de parámetros con diferentes tipos de dato. Por ejemplo, la función TO_CHAR se puede llamar de más de una forma, lo que permite 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 familia de tipos de dato*.

Plantéese 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, es posible que deseé buscar empleados por su identificador de empleado, así como proporcionar una forma de buscar empleados por apellido. La lógica es esencialmente la misma, pero los parámetros o criterios de búsqueda varían.
- Si desea ampliar la funcionalidad cuando no desea sustituir el código existente.

Resumen

En esta lección, debe haber aprendido 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
- Utilizar matrices asociativas (tablas de índice) y registros en paquetes



Copyright © 2010, Oracle y/o sus filiales. 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.

Mediante el encapsulador PL/SQL puede ocultar el código fuente 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



Copyright © 2010, Oracle y/o sus filiales. 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.

Uso de Paquetes Proporcionados por Oracle en el Desarrollo de Aplicaciones



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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
- Describir las principales funciones de UTL_MAIL



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos de la Lección

En esta lección, aprenderá a utilizar algunos de los paquetes proporcionados por Oracle, así como sus capacidades.

Agenda de la Lección

- Identificación de las ventajas de utilizar los paquetes proporcionados por Oracle y enumeración de algunos de estos paquetes
- Uso de los siguientes paquetes proporcionados por Oracle:
 - DBMS_OUTPUT
 - UTL_FILE
 - UTL_MAIL

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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.



Copyright © 2010, Oracle y/o sus filiales. 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 con el que más se familiarizará durante este curso. Ya debe estar familiarizado con este paquete si ha asistido al curso *Oracle Database: Conceptos Fundamentales de PL/SQL*.

Ejemplos de Algunos Paquetes Proporcionados por Oracle

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

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



Copyright © 2010, Oracle y/o sus filiales. 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. En esta lección se abordan los tres primeros paquetes de la diapositiva. Para obtener más información, consulte *Oracle Database PL/SQL Packages and Types Reference* (Referencia de Tipos y Paquetes PL/SQL de Oracle Database). A continuación se muestra una breve descripción del resto de paquetes mostrados en la diapositiva:

- DBMS_OUTPUT proporciona la depuración y el almacenamiento en buffer de los datos de texto.
- UTL_FILE permite la lectura y la escritura de archivos de texto del sistema operativo.
- UTL_MAIL permite redactar y enviar mensajes de correo electrónico.
- DBMS_ALERT soporta notificación asíncrona de eventos de base de datos. Los mensajes o las alertas se envían en un comando COMMIT.
- DBMS_LOCK se utiliza para solicitar, convertir y liberar bloqueos a través de los servicios de gestión de bloqueos de Oracle.
- DBMS_SESSION permite la utilización programática de la sentencia SQL ALTER SESSION y de otros comandos a nivel de sesión.
- DBMS_APPLICATION_INFO se puede utilizar con Oracle Trace y la utilidad de rastreo de SQL para registrar nombres de módulos o transacciones en ejecución en la base de datos para su posterior uso al realizar un seguimiento del rendimiento de varios módulos y al depurar.
- El paquete HTP escribe datos con etiquetas HTML en los buffers de la base de datos.
- DBMS_SCHEDULER permite la planificación y ejecución automática de bloques PL/SQL, procedimientos almacenados y procedimientos externos y ejecutables (que se tratan en el Apéndice G).

Agenda de la Lección

- Identificación de las ventajas de utilizar los paquetes proporcionados por Oracle y enumeración de algunos de estos paquetes
- Uso de los siguientes paquetes proporcionados por Oracle:
 - DBMS_OUTPUT
 - UTL_FILE
 - UTL_MAIL

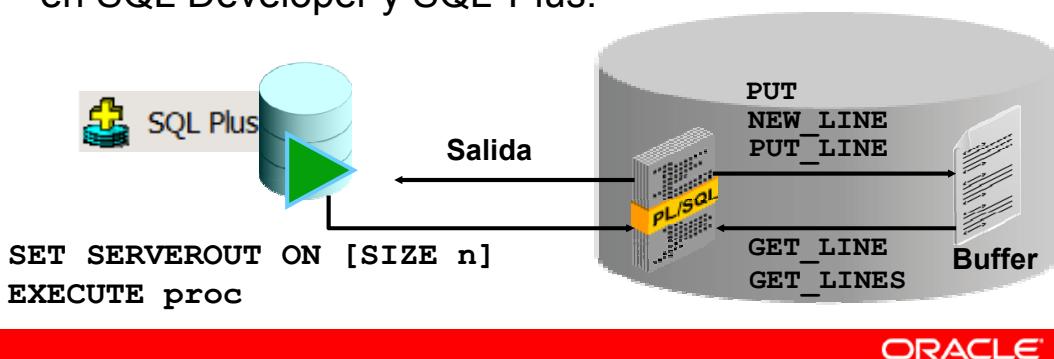
ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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 subprograma remitente o el disparador termina.
- Utilice SET SERVEROUTPUT ON para mostrar los mensajes en SQL Developer y SQL*Plus.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Funcionamiento 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 lo siguiente:

- PUT agrega texto desde el procedimiento a la línea actual del buffer de salida de línea.
- NEW_LINE coloca un marcador de final de línea en el buffer de salida.
- PUT_LINE combina la acción de PUT y NEW_LINE (para recortar los espacios iniciales).
- GET_LINE recupera la línea actual desde el buffer en una variable de procedimiento.
- GET_LINES recupera una matriz de líneas en una variable de matriz de procedimiento.
- ENABLE/DISABLE activa o desactiva 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 es el número de caracteres. El valor mínimo es 2.000 y el máximo es ilimitado. El valor por defecto es 20.000.
- Un parámetro entero entre 2.000 y 1.000.000 en el procedimiento ENABLE.

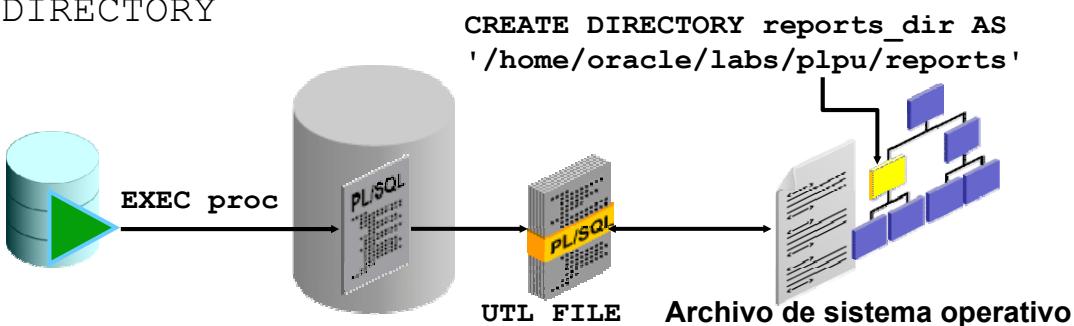
Puede hacer que la salida de los resultados sea en la ventana para la depuración. Puede rastrear una ruta de acceso de ejecución del código para una función o un procedimiento. Puede enviar mensajes entre subprogramas y disparadores.

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

Uso del Paquete UTL_FILE para Interactuar con Archivos del Sistema Operativo

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

- 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



ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 DIRECTORY my_dir TO public.
```
- Las rutas de acceso especificadas en el parámetro de inicialización de la base de datos `utl_file_dir`.

Se recomienda utilizar la función CREATE DIRECTORY en lugar de UTL_FILE_DIR para la verificación de acceso a directorios. Los objetos de directorio ofrecen más flexibilidad y control más específico al administrador de la aplicación UTL_FILE, se pueden mantener dinámicamente (es decir, sin cerrar la base de datos) y son consistentes con otras herramientas de Oracle. El privilegio CREATE DIRECTORY sólo se otorga a SYS y SYSTEM por defecto.

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.

Algunos de los Procedimientos y Funciones de UTL_FILE

Subprograma	Descripción
Función ISOPEN	Determina si un manejador de archivos hace referencia a un archivo abierto
Función FOPEN	Abre un archivo para entrada o salida
Función FCLOSE	Cierra todos los manejadores de archivos abiertos
Procedimiento FCOPY	Copia una parte contigua de un archivo en otro recién creado
Procedimiento FGETATTR	Lee y devuelve los atributos de un archivo de disco
Procedimiento GET_LINE	Lee texto de un archivo abierto
Procedimiento FREMOVE	Suprime un archivo de disco, si tiene privilegios suficientes
Procedimiento FRENAME	Cambia el nombre de un archivo existente
Procedimiento GET_LINE	Escribe una cadena en un archivo
Procedimiento PUT_LINE	Escribe una línea en un archivo y, por lo tanto, agrega un terminador de línea específico del sistema operativo

ORACLE

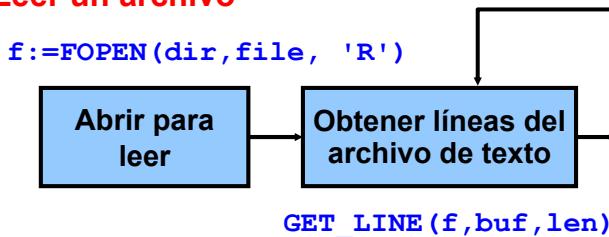
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Algunos de los Procedimientos y Funciones de UTL_FILE

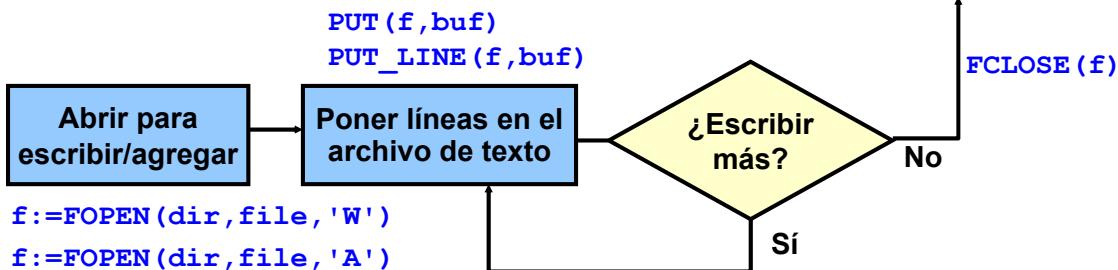
En la tabla de la diapositiva se muestran algunos de los subprogramas del paquete UTL_FILE. Para obtener una lista completa de los subprogramas del paquete, consulte *Oracle Database PL/SQL Packages and Types Reference (Referencia de Tipos y Paquetes PL/SQL de Oracle Database)*.

Procesamiento de Archivos con el Paquete UTL_FILE: Visión General

Leer un archivo



Escribir o agregar a un archivo



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Procesamiento de Archivos con el Paquete UTL_FILE

Puede utilizar los procedimientos y las funciones del paquete UTL_FILE para abrir archivos con la función FOPEN. A continuación, puede leer, escribir o agregar un elemento al archivo hasta que el procesamiento 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 archivos utilizado en operaciones de E/S posteriores.
- La función IS_OPEN devuelve un valor booleano cada vez que un manejador de archivos 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.
- 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.

Uso de las Excepciones Declaradas Disponibles en el Paquete UTL_FILE

Nombre de Excepción	Descripción
INVALID_PATH	Ubicación del archivo no válida
INVALID_MODE	El parámetro <code>open_mode</code> de <code>FOPEN</code> no es válido
INVALID_FILEHANDLE	Manejador de archivos no válido
INVALID_OPERATION	El archivo no se ha podido abrir o no se ha podido operar en él como se ha solicitado
READ_ERROR	Se ha producido un error en el sistema operativo durante la operación de lectura
WRITE_ERROR	Se ha producido un error en el sistema operativo durante la operación de escritura
INTERNAL_ERROR	Error PL/SQL no especificado

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Excepciones en el Paquete UTL_FILE

El paquete UTL_FILE declara quince excepciones que indican una condición de error en el procesamiento del archivo del sistema operativo. Puede que tenga que manejar una de estas excepciones al utilizar los subprogramas UTL_FILE.

En la diapositiva se muestra un subjuego de las excepciones. Para obtener más información sobre el resto de excepciones, consulte *Oracle Database PL/SQL Packages and Types Reference* (Referencia de Tipos y Paquetes PL/SQL de Oracle Database).

Nota: estas excepciones siempre deben llevar como prefijo el 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.

Funciones FOPEN e IS_OPEN: Ejemplo

- Esta función FOPEN abre un archivo para entrada o salida.

```
FUNCTION FOPEN (p_location  IN VARCHAR2,
               p_filename   IN VARCHAR2,
               p_open_mode  IN VARCHAR2)
RETURN UTL_FILE.FILE_TYPE;
```

- La función IS_OPEN determina si un manejador de archivos hace referencia a un archivo abierto.

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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Parámetros de las Funciones FOPEN e IS_OPEN: Ejemplo

Entre estos parámetros se incluyen:

- Parámetro p_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 p_filename: Especifica el nombre del archivo, incluida 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 archivos 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 archivos. La función IS_OPEN prueba un manejador de archivos para ver si identifica un archivo abierto. Devuelve un valor booleano TRUE si se ha abierto el archivo; de lo contrario, devuelve un valor FALSE que indica que el archivo no se ha abierto. En el ejemplo de la diapositiva se muestra cómo combinar la utilización de los dos subprogramas. Para obtener la sintaxis completa, consulte *Oracle Database PL/SQL Packages and Types Reference* (Referencia de Tipos y Paquetes PL/SQL de Oracle Database).

Parámetros de las Funciones FOPEN e IS_OPEN: Ejemplo (continuación)

Asegúrese de que el archivo externo y la base de datos están en la misma computadora.

```

CREATE OR REPLACE PROCEDURE read_file(p_dir VARCHAR2, p_filename
VARCHAR2) IS
    f_file UTL_FILE.FILE_TYPE;
    v_buffer VARCHAR2(200);
    v_lines PLS_INTEGER := 0;
BEGIN
    DBMS_OUTPUT.PUT_LINE(' Start ');
    IF NOT UTL_FILE.IS_OPEN(f_file) THEN
        DBMS_OUTPUT.PUT_LINE(' Open ');
        f_file := UTL_FILE.FOPEN (p_dir, p_filename, 'R');
        DBMS_OUTPUT.PUT_LINE(' Opened ');
    BEGIN
        LOOP
            UTL_FILE.GET_LINE(f_file, v_buffer);
            v_lines := v_lines + 1;
            DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_lines, '099') ||
                '||' || buffer);
        END LOOP;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE(' ** End of File **');
    END; -- ends Begin
    DBMS_OUTPUT.PUT_LINE(v_lines||' lines read from file');
    UTL_FILE.FCLOSE(f_file);
END IF;
END read_file;
/
SHOW ERRORS
EXECUTE read_file('REPORTS_DIR', 'instructor.txt')

```

La salida parcial del código anterior es la siguiente:

```

PROCEDURE read_file(dir Compiled.
No Errors.
line 27: SQLPLUS Command Skipped: set serveroutput on
anonymous block completed
Start
Open
Opened
001 SALARY REPORT: GENERATED ON
002                         08-MAR-01
003
004 DEPARTMENT: 10
005   EMPLOYEE: Whalen earns: 4400
...
120 DEPARTMENT: 110
121   EMPLOYEE: Higgins earns: 12000
122   EMPLOYEE: Gietz earns: 8300
123   EMPLOYEE: Grant earns: 7000
124 *** END OF REPORT ***
** End of File **
124 lines read from file

```

Uso de UTL_FILE: Ejemplo

```

CREATE OR REPLACE PROCEDURE sal_status(
    p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
    f_file UTL_FILE.FILE_TYPE;
    CURSOR cur_emp IS
        SELECT last_name, salary, department_id
        FROM employees ORDER BY department_id;
    v_newdeptno employees.department_id%TYPE;
    v_olddeptno employees.department_id%TYPE := 0;
BEGIN
    f_file:=UTL_FILE.FOPEN (p_dir, p_filename, 'W');
    UTL_FILE.PUT_LINE(f_file,
        'REPORT: GENERATED ON ' || SYSDATE);
    UTL_FILE.NEW_LINE (f_file);
    . . .

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de UTL_FILE: Ejemplo

En el ejemplo de la diapositiva, 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 `p_dir` para el nombre del directorio en el que escribir el archivo de texto
- El parámetro `p_filename` para especificar el nombre del archivo

Por ejemplo, para llamar al procedimiento, utilice lo siguiente después de asegurarse de que el archivo externo y la base de datos están en la misma computadora:

```
EXECUTE sal_status('REPORTS_DIR', 'salreport2.txt')
```

Nota: la ubicación de directorio utilizada (`REPORTS_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 ha terminado.

Uso de UTL_FILE: Ejemplo

```

.
.
.
FOR emp_rec IN cur_emp LOOP
    IF emp_rec.department_id <> v_olddeptno THEN
        UTL_FILE.PUT_LINE (f_file,
            'DEPARTMENT: ' || emp_rec.department_id);
        UTL_FILE.NEW_LINE (f_file);
    END IF;
    UTL_FILE.PUT_LINE (f_file,
        'EMPLOYEE: ' || emp_rec.last_name ||
        ' earns: ' || emp_rec.salary);
    v_olddeptno := emp_rec.department_id;
    UTL_FILE.NEW_LINE (f_file);
END LOOP;
UTL_FILE.PUT_LINE(f_file,'*** END OF REPORT ***');
UTL_FILE.FCLOSE (f_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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de UTL_FILE: Ejemplo (continuación)

A continuación se muestra un ejemplo del archivo de salida salreport2.txt:

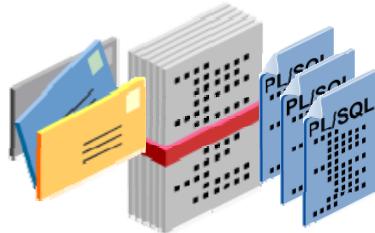
```

salreport2.txt *
REPORT: GENERATED ON 07-AUG-09
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
EMPLOYEE: Baida earns: 2900

```

¿Qué Es el Paquete UTL_MAIL?

- Una utilidad para gestionar correo electrónico
- Necesita la definición del parámetro de inicialización de la base de datos `SMTP_OUT_SERVER`
- 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 © 2010, Oracle y/o sus filiales. 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 SQL*Plus y ejecute los siguientes scripts:

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

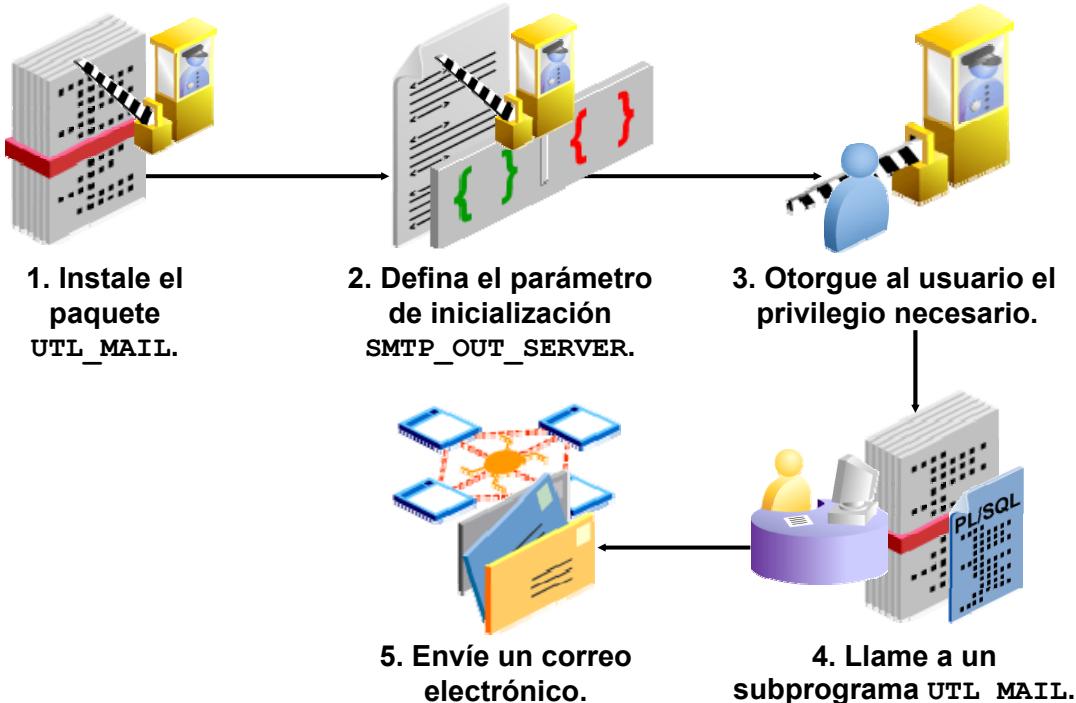
Debe definir el parámetro `SMTP_OUT_SERVER` en el archivo de inicialización de la base de datos `init.ora`:

```
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
```

Configuración y Uso de UTL_MAIL: Visión General



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Configuración y Uso de UTL_MAIL: Visión General

En Oracle Database 11g, el paquete UTL_MAIL es ahora un paquete de derechos del emisor y el usuario que llama necesitará tener otorgado el privilegio connect en la lista de control de acceso asignada al host de red remoto al que desea conectarse. El administrador de seguridad realiza esta tarea.

Nota

- Para obtener más información sobre cómo un usuario con las funciones SYSDBA otorga a un usuario los privilegios detallados necesarios para usar este paquete, consulte el tema “Gestión de Acceso Detallado a los Servicios de Red Externos” en *Oracle Database Security Guide 11g Release 2 (11.2)* [Guía de Seguridad de Oracle Database Versión 2 (11.2)] y el curso de formación dirigido por un instructor *Oracle Database 11g Advanced PL/SQL (PL/SQL Avanzado de Oracle Database 11g)*.
- Debido a las restricciones de firewall, los ejemplos de UTL_MAIL de esta lección no se pueden demostrar; por lo tanto, no se ha diseñado ningún ejercicio práctico para utilizar UTL_MAIL.

Resumen de los Subprogramas UTL_MAIL

Subprograma	Descripción
Procedimiento SEND	Empaquea un mensaje de correo electrónico, localiza la información de SMTP y entrega el mensaje al servidor SMTP para reenviar a los destinatarios
Procedimiento SEND_ATTACH_RAW	Representa el procedimiento SEND sobrecargado para anexos RAW
Procedimiento SEND_ATTACH_VARCHAR2	Representa el procedimiento SEND sobrecargado para anexos VARCHAR2



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Instalación y Uso de UTL_MAIL

- Como SYSDBA, con SQL Developer o SQL*Plus:
 - Instale el paquete UTL_MAIL

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

- Defina SMTP_OUT_SERVER

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

- 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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Instalación y Uso de UTL_MAIL

La diapositiva 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 el parámetro SMTP_OUT_SERVER, tendrá que reiniciar la instancia de la base de datos. Estas tareas las realiza un usuario con funciones SYSDBA.

En el último ejemplo de la diapositiva se 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 *Oracle Database PL/SQL Packages and Types Reference* (Referencia de Tipos y Paquetes PL/SQL de Oracle Database).

Sintaxis del Procedimiento SEND

Empaqueta un mensaje de correo electrónico con el formato adecuado, localiza la información de SMTP y entrega el mensaje al servidor SMTP para reenviar a los destinatarios.

```
UTL_MAIL.SEND (
    sender      IN      VARCHAR2 CHARACTER SET ANY_CS,
    recipients  IN      VARCHAR2 CHARACTER SET ANY_CS,
    cc          IN      VARCHAR2 CHARACTER SET ANY_CS
                      DEFAULT NULL,
    bcc         IN      VARCHAR2 CHARACTER SET ANY_CS
                      DEFAULT NULL,
    subject     IN      VARCHAR2 CHARACTER SET ANY_CS
                      DEFAULT NULL,
    message     IN      VARCHAR2 CHARACTER SET ANY_CS,
    mime_type   IN      VARCHAR2
                      DEFAULT 'text/plain; charset=us-ascii',
    priority    IN      PLS_INTEGER DEFAULT NULL);
```

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Procedimiento SEND

Este procedimiento empaqueta un mensaje de correo electrónico con el formato adecuado, localiza la información de SMTP y entrega el mensaje al servidor SMTP para reenviar a los destinatarios. Oculta la API SMTP y expone una utilidad de correo electrónico de una línea para facilitar el uso.

Parámetros del Procedimiento SEND

- **sender:** dirección de correo electrónico del remitente.
- **recipients:** direcciones de correo electrónico de los destinatarios, separadas por comas.
- **cc:** direcciones de correo electrónico de los destinatarios con copia, separadas por comas. El valor por defecto es NULL.
- **bcc:** direcciones de correo electrónico de los destinatarios con copia oculta, separadas por comas. El valor por defecto es NULL.
- **subject:** cadena que se va a incluir como cadena de asunto del mensaje electrónico. El valor por defecto es NULL.
- **message:** cuerpo del mensaje de texto.
- **mime_type:** tipo Mime del mensaje; el valor por defecto es 'text/plain; charset=us-ascii'.
- **priority:** prioridad del mensaje. El valor por defecto es NULL.

Procedimiento SEND_ATTACH_RAW

Este procedimiento es el procedimiento SEND sobrecargado para anexos RAW.

```
UTL_MAIL.SEND_ATTACH_RAW (
    sender          IN      VARCHAR2 CHARACTER SET ANY_CS,
    recipients     IN      VARCHAR2 CHARACTER SET ANY_CS,
    cc              IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    bcc             IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    subject         IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    message         IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    mime_type       IN      VARCHAR2 DEFAULT CHARACTER SET ANY_CS
                            DEFAULT 'text/plain; charset=us-ascii',
    priority        IN      PLS_INTEGER DEFAULT 3,
    attachment     IN      RAW,
    att_inline      IN      BOOLEAN DEFAULT TRUE,
    att_mime_type   IN      VARCHAR2 CHARACTER SET ANY_CS
                            DEFAULT 'text/plain; charset=us-ascii',
    att_filename    IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL);
```

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Parámetros del Procedimiento SEND_ATTACH_RAW

- **sender:** dirección de correo electrónico del remitente.
- **recipients:** direcciones de correo electrónico de los destinatarios, separadas por comas.
- **cc:** direcciones de correo electrónico de los destinatarios con copia, separadas por comas. El valor por defecto es NULL.
- **bcc:** direcciones de correo electrónico de los destinatarios con copia oculta, separadas por comas. El valor por defecto es NULL.
- **subject:** cadena que se va a incluir como cadena de asunto del mensaje electrónico. El valor por defecto es NULL.
- **message:** cuerpo del mensaje de texto.
- **mime_type:** tipo Mime del mensaje; el valor por defecto es 'text/plain; charset=us-ascii'
- **priority:** prioridad del mensaje. El valor por defecto es NULL.
- **attachment:** anexo RAW.
- **att_inline:** especifica si el anexo se puede ver en línea con el cuerpo del mensaje. El valor por defecto es TRUE.
- **att_mime_type:** tipo Mime del anexo; el valor por defecto es 'application/octet'
- **att_filename:** cadena que especifica un nombre de archivo que contiene el anexo. El valor por defecto es NULL.

Envío de Correo Electrónico con Anexos Binarios: Ejemplo

```

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 => 'oralogo.gif');
END;
/

```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Envío de Correo Electrónico con Anexos Binarios: Ejemplo

La diapositiva 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) 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. El código para get_image se muestra en la página siguiente.

Envío de Correo Electrónico con Anexos Binarios: Ejemplo (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 en SQL Developer o SQL*Plus:

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

Nota

- Necesita el privilegio del sistema CREATE ANY DIRECTORY para ejecutar esta sentencia.
- Debido a las restricciones de firewall en Oracle Education Center, los ejemplos de esta página y de la anterior no están disponibles para la demostración.

Procedimiento SEND_ATTACH_VARCHAR2

Este procedimiento es el procedimiento SEND sobrecargado para anexos VARCHAR2.

```
UTL_MAIL.SEND_ATTACH_VARCHAR2 (
    sender          IN  VARCHAR2 CHARACTER SET ANY_CS,
    recipients      IN  VARCHAR2 CHARACTER SET ANY_CS,
    cc              IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    bcc             IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    subject         IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    message         IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    mime_type       IN  VARCHAR2 CHARACTER SET ANY_CS
                           DEFAULT 'text/plain; charset=us-ascii',
    priority        IN  PLS_INTEGER DEFAULT 3,
    attachment      IN  VARCHAR2 CHARACTER SET ANY_CS,
    att_inline      IN  BOOLEAN DEFAULT TRUE,
    att_mime_type   IN  VARCHAR2 CHARACTER SET ANY_CS
                           DEFAULT 'text/plain; charset=us-ascii',
    att_filename    IN  VARCHAR2CHARACTER SET ANY_CS DEFAULT NULL);
```

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Parámetros del Procedimiento SEND_ATTACH_VARCHAR2

- **sender:** dirección de correo electrónico del remitente.
- **recipients:** direcciones de correo electrónico de los destinatarios, separadas por comas.
- **cc:** direcciones de correo electrónico de los destinatarios con copia, separadas por comas. El valor por defecto es NULL.
- **bcc:** direcciones de correo electrónico de los destinatarios con copia oculta, separadas por comas. El valor por defecto es NULL.
- **subject:** cadena que se va a incluir como cadena de asunto del mensaje electrónico. El valor por defecto es NULL.
- **Message:** cuerpo del mensaje de texto.
- **mime_type:** tipo Mime del mensaje; el valor por defecto es 'text/plain; charset=us-ascii'
- **priority:** prioridad del mensaje. El valor por defecto es NULL.
- **attachment:** anexo de texto.
- **att_inline:** especifica si el anexo está en línea. El valor por defecto es TRUE.
- **att_mime_type:** tipo Mime del anexo; el valor por defecto es 'text/plain; charset=us-ascii'
- **att_filename:** cadena que especifica un nombre de archivo que contiene el anexo. El valor por defecto es NULL.

Envío de Correo Electrónico con Anexos de Texto: Ejemplo

```

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;
/

```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Envío de Correo Electrónico con Anexos de Texto

En la diapositiva se 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 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) 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.

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

```
CREATE DIRECTORY temp AS '/temp';
```

Nota

- Se necesita el privilegio del sistema CREATE ANY DIRECTORY para ejecutar esta sentencia.
- Debido a las restricciones de firewall en Oracle Education Center, los ejemplos de esta página y de la anterior no están disponibles para la demostración.

Prueba

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 la funcionalidad a través de objetos de directorio para permitir el acceso a directorios de sistema operativo específicos.

1. Verdadero
2. Falso

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: 1

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

Resumen

En esta lección debe haber aprendido lo siguiente:

- Funcionamiento del paquete DBMS_OUTPUT
- Uso de UTL_FILE para dirigir la salida a los archivos del sistema operativo
- Acerca de las principales funciones de UTL_MAIL



Copyright © 2010, Oracle y/o sus filiales. 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 SQL*Plus.

En esta lección, debe haber aprendido a utilizar las potentes funciones que proporciona la base de datos para crear archivos de texto en el sistema operativo mediante UTL_FILE. También debe haber aprendido a enviar correo electrónico con o sin anexos binarios o de texto mediante el paquete UTL_MAIL.

Nota: para obtener más información sobre todos los paquetes y tipos PL/SQL, consulte *PL/SQL Packages and Types Reference* (Referencia de Tipos y Paquetes PL/SQL).

Práctica 5: Visión General

Esta práctica abarca cómo utilizar UTL_FILE para generar un informe de texto.



Copyright © 2010, Oracle y/o sus filiales. 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.

Uso de SQL Dinámico

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

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

- Describir el flujo de ejecución de sentencias SQL
- Crear y ejecutar sentencias SQL dinámicamente utilizando SQL dinámico nativo (NDS)
- Identificar situaciones en las que debe utilizar el paquete DBMS_SQL en lugar de NDS para crear y ejecutar sentencias SQL dinámicamente



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos de la Lección

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.

Agenda de la Lección

- Uso de SQL dinámico nativo (NDS)
- Uso del paquete DBMS_SQL

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Flujo de Ejecución de SQL

- Todas las sentencias SQL pasan por algunas o todas las fases siguientes:
 - Análisis
 - Enlace
 - Ejecución
 - Recuperación
- Puede que algunas fases no sean relevantes para todas las sentencias:
 - La fase de recuperación se aplica a consultas.
 - Para sentencias SQL embebidas como SELECT, DML, MERGE, COMMIT, SAVEPOINT 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 © 2010, Oracle y/o sus filiales. 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:

- **Análisis:** 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.
- **Enlace:** después de analizar, puede que el servidor de Oracle necesite valores de o para la variable de enlace 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 de enlace.
- **Ejecución:** en este punto, el servidor Oracle tiene toda la información y los recursos necesarios y la sentencia se ejecuta. Para sentencias que no sean de consulta, ésta es la última fase.
- **Recuperación:** 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.

Trabajar con 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, variable de cadena o expresión de cadena dentro de la aplicación
- Es una sentencia SQL con datos de columna variables o diferentes condiciones con o sin marcadores de posición (variables de enlace)
- Activa sentencias DDL, DCL o 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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

SQL Dinámico

Las sentencias SQL embebidas disponibles en PL/SQL se limitan a SELECT, INSERT, UPDATE, DELETE, MERGE, 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 de lenguaje de definición de datos (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 utilice SQL dinámico para operar en una tabla cuyo nombre no se conoce hasta el momento de ejecución o para ejecutar una sentencia de DDL (como CREATE TABLE), una sentencia de control de datos (como GRANT) o una sentencia de control de sesión (como ALTER SESSION).

Uso de SQL Dinámico

- Utilice SQL dinámico cuando todo el texto de la sentencia SQL dinámica sea desconocido hasta el momento de la ejecución; por lo tanto, su sintaxis se comprueba en *tiempo de ejecución* en lugar de en el *tiempo de compilación*.
- Utilice SQL dinámico cuando uno de los siguientes elementos sea desconocido en tiempo de precompilación:
 - Texto de la sentencia SQL como comandos, cláusulas, etc.
 - Número y tipos de dato de las variables del host
 - Referencias a objetos de base de datos como tablas, columnas, índices, secuencias, nombres de usuario y vistas
- Utilice SQL dinámico para hacer que los programas PL/SQL sean más generales y flexibles.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de SQL Dinámico

En PL/SQL, necesita que el SQL dinámico ejecute las siguientes sentencias SQL, donde todo el texto es desconocido en el tiempo de compilación como:

- Una sentencia SELECT que incluye un identificador desconocido en tiempo de compilación (como un nombre de tabla)
- Una sentencia WHERE en la que el nombre de columna es desconocido en tiempo de compilación

Nota

Para obtener más información sobre el SQL dinámico, consulte los siguientes recursos:

- *Pro*C/C++ Programmer's Guide*
 - *Lesson 13, Oracle Dynamic SQL* (Guía del Programador de Pro*C/C++, lección 13 sobre SQL dinámico en Oracle) abarca los cuatro métodos disponibles que puede utilizar para definir sentencias SQL dinámicas. Describe brevemente las funciones y limitaciones de cada método y, a continuación, ofrece instrucciones para seleccionar el método correcto. En las secciones posteriores de la misma guía se muestra cómo utilizar los métodos y se incluyen programas de ejemplo que puede estudiar.
 - *Lesson 15, Oracle Dynamic SQL: Method 4* (lección 15 sobre SQL dinámico en Oracle, método 4) contiene información muy detallada sobre el método 4 al definir sentencias SQL dinámicas.
- El libro *Oracle PL/SQL Programming* (Programación PL/SQL en Oracle) de Steven Feuerstein y Bill Pribyl. *Lesson 16, Dynamic SQL and Dynamic PL/SQL* (lección 16 sobre SQL dinámico y PL/SQL dinámico) contiene más información sobre SQL dinámico.

SQL Dinámico Nativo (NDS)

- Proporciona soporte nativo para SQL dinámico directamente en el lenguaje PL/SQL.
- Proporciona la capacidad de ejecutar sentencias SQL cuya estructura no se conoce hasta el tiempo de ejecución.
- Si la sentencia SQL dinámica es una sentencia SELECT que devuelve varias filas, NDS ofrece las siguientes opciones:
 - Utilice la sentencia EXECUTE IMMEDIATE con la cláusula BULK COLLECT INTO
 - Utilice las sentencias OPEN-FOR, FETCH y CLOSE
- En Oracle Database 11g, NDS soporta sentencias de más de 32 KB al aceptar un argumento CLOB.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

SQL Dinámico Nativo

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 de enlace en los parámetros dinámicos de las sentencias EXECUTE IMMEDIATE y OPEN. El SQL dinámico nativo incluye las siguientes funciones:

- Definir una sentencia SQL dinámica.
- Manejar las variables de enlace 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 NDS 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.

Copyright © 2010, Oracle y/o sus filiales. 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 haber una variable o campo correspondiente, compatible con el tipo 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 diapositiva no está completa porque existe soporte para operaciones de procesamiento en bloque (que es un tema que no abarca este curso).

Métodos Disponibles para Usar NDS

Nº de Método	Tipo de Sentencia SQL	Sentencias SQL NDS Utilizadas
Método 1	Que no es de consulta sin variables del host	EXECUTE IMMEDIATE sin las cláusulas USING e INTO
Método 2	Que no es de consulta con número conocido de variables del host de entrada	EXECUTE IMMEDIATE con una cláusula USING
Método 3	Consulta con número conocido de elementos de la lista select y variables del host de entrada	EXECUTE IMMEDIATE con las cláusulas USING e INTO
Método 4	Consulta con número desconocido de elementos de la lista select o variables del host de entrada	Uso del Paquete DBMS_SQL

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Métodos Disponibles para Usar NDS

Los cuatro métodos disponibles para NDS que se muestran en la diapositiva son cada vez más generales. Es decir, el método 2 engloba el método 1, el método 3 engloba los métodos 1 y 2, y el método 4 engloba los métodos 1, 2 y 3. Sin embargo, cada método resulta más útil para manejar un determinado tipo de sentencia SQL, de la siguiente forma:

Método 1:

Este método permite al programa aceptar o crear una sentencia SQL dinámica y, a continuación, ejecutarla inmediatamente con el comando EXECUTE IMMEDIATE. La sentencia SQL no debe ser una consulta (sentencia SELECT) y no debe contener ningún marcador de posición para variables del host de entrada. Por ejemplo, las siguientes cadenas de host están cualificadas:

- DELETE FROM EMPLOYEES WHERE DEPTNO = 20
- GRANT SELECT ON EMPLOYEES TO scott

Con el método 1, la sentencia SQL se analiza cada vez que se ejecuta.

Nota

- Entre los ejemplos que no son de consulta se incluyen las sentencias de lenguaje de definición de datos (DDL), UPDATE, INSERT o DELETE.
- El término *elemento de la lista select* incluye nombres de columna y expresiones como SAL * 1.10 y MAX (SAL).

Métodos Disponibles para Usar NDS (continuación)

Método 2:

Este método permite al programa aceptar o crear una sentencia SQL dinámica y, a continuación, procesarla con los comandos PREPARE y EXECUTE. La sentencia SQL no debe ser una consulta. El número de marcadores de posición para las variables del host de entrada y los tipos de dato de las variables del host de entrada se deben conocer en el tiempo de precompilación. Por ejemplo, las siguientes cadenas de host están en esta categoría:

- `INSERT INTO EMPLOYEES (FIRST_NAME, LAST_NAME, JOB_ID) VALUES (:emp_first_name, :emp_last_name, :job_id)`
- `DELETE FROM EMPLOYEES WHERE EMPLOYEE_ID = :emp_number`

Con el método 2, la sentencia SQL se analiza sólo una vez, pero se puede ejecutar varias veces con distintos valores para las variables del host. Las sentencias SQL de definición de datos como CREATE y GRANT se ejecutan cuando están preparadas.

Método 3:

Este método permite al programa aceptar o crear una consulta dinámica y, a continuación, procesarla con los comandos PREPARE con los comandos del cursor DECLARE, OPEN, FETCH y CLOSE. El número de elementos de la lista select, el número de marcadores de posición para las variables del host de entrada y los tipos de dato de las variables del host de entrada se deben conocer en el tiempo de precompilación. Por ejemplo, las siguientes cadenas de host están cualificadas:

- `SELECT DEPARTMENT_ID, MIN(SALARY), MAX(SALARY)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID`
- `SELECT LAST_NAME, EMPLOYEE_ID
FROM EMPLOYEES
WHERE DEPARTMENT_ID = :dept_number`

Método 4:

Este método permite al programa aceptar o crear una sentencia SQL dinámica y, a continuación, procesarla con descriptores. Un descriptor es un área de memoria utilizada por el programa y Oracle para contener una descripción completa de las variables en una sentencia SQL dinámica. El número de elementos de la lista select, el número de marcadores de posición para las variables del host de entrada y los tipos de dato de las variables del host de entrada pueden ser desconocidos hasta el tiempo de ejecución. Por ejemplo, las siguientes cadenas de host están en esta categoría:

- `INSERT INTO EMPLOYEES (<unknown>) VALUES (<unknown>)`
- `SELECT <unknown> FROM EMPLOYEES WHERE DEPARTMENT_ID = 20`

El método 4 es necesario para sentencias SQL dinámicas que contienen un número desconocido de elementos de la lista select o variables del host de entrada. Con este método, utilice el paquete DBMS_SQL, que se trata posteriormente en esta lección. Las situaciones que necesitan el uso del método 4 son escasas.

Nota

Para obtener más información sobre los cuatro métodos de SQL dinámico, consulte las siguientes lecciones en *Pro*C/C++ Programmer's Guide* (Guía del Programador de Pro*C/C++).

- *Lesson 13, Oracle Dynamic SQL (Lección 13 sobre SQL dinámico en Oracle)*
- *Lesson 15, Oracle Dynamic SQL: Method 4 (Lección 15 sobre SQL dinámico en Oracle, método 4)*

SQL Dinámico con una Sentencia DDL: Ejemplos

```
-- Create a table using dynamic SQL

CREATE OR REPLACE PROCEDURE create_table(
    p_table_name VARCHAR2, p_col_specs VARCHAR2) IS
BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE ' || p_table_name ||
                       ' (' || p_col_specs || ')';
END;
/
```

```
-- Call the procedure

BEGIN
    create_table('EMPLOYEE NAMES',
                 'id NUMBER(4) PRIMARY KEY, name VARCHAR2(40)');
END;
/
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 al procedimiento 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 diapositiva, 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 OR REPLACE PROCEDURE add_col(p_table_name VARCHAR2,
                                    p_col_spec VARCHAR2) IS
    v_stmt VARCHAR2(100) := 'ALTER TABLE ' || p_table_name ||
                           ' ADD ' || p_col_spec;
BEGIN
    EXECUTE IMMEDIATE v_stmt;
END;
/
```

Para agregar una nueva columna a una tabla, introduzca lo siguiente:

```
EXECUTE add_col('employee_names', 'salary number(8,2)')
```

SQL Dinámico con Sentencias DML

```
-- Delete rows from any table:
CREATE FUNCTION del_rows(p_table_name VARCHAR2)
RETURN NUMBER IS
BEGIN
    EXECUTE IMMEDIATE 'DELETE FROM '|| p_table_name;
    RETURN SQL%ROWCOUNT;
END;
/
BEGIN DBMS_OUTPUT.PUT_LINE(
    del_rows('EMPLOYEE_NAMES')|| ' rows deleted.');
END;
/
```

```
-- Insert a row into a table with two columns:
CREATE PROCEDURE add_row(p_table_name VARCHAR2,
    p_id NUMBER, p_name VARCHAR2) IS
BEGIN
    EXECUTE IMMEDIATE 'INSERT INTO '|| p_table_name ||
        ' VALUES (:1, :2)' USING p_id, p_name;
END;
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

SQL Dinámico con Sentencias DML

En el primer ejemplo de código de la diapositiva se define una sentencia SQL dinámica con el método 1, es decir, que no es de consulta sin variables del host. Los ejemplos de la diapositiva 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 de enlace `:1` y `:2` no son importantes, pero el orden de los nombres de los parámetros (`p_id` y `p_name`) en la cláusula `USING` está asociado a las variables de enlace por posición, según el orden de su respectiva aparición. Por lo tanto, el parámetro PL/SQL `p_id` se asigna al marcador de posición `:1` y el parámetro `p_name` se asigna al marcador de posición `:2`. Los nombres de las variables de marcador de posición o de enlace 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

```

CREATE FUNCTION get_emp( p_emp_id NUMBER )
  RETURN employees%ROWTYPE IS
    v_stmt VARCHAR2(200);
    v_emprec employees%ROWTYPE;
BEGIN
  v_stmt := 'SELECT * FROM employees ' ||
            'WHERE employee_id = :p_emp_id';
  EXECUTE IMMEDIATE v_stmt INTO v_emprec USING p_emp_id;
  RETURN v_emprec;
END;
/
DECLARE
  v_emprec employees%ROWTYPE := get_emp(100);
BEGIN
  DBMS_OUTPUT.PUT_LINE('Emp: '|| v_emprec.last_name);
END;
/

```

FUNCTION get_emp(p_emp_id Compiled.
anonymous block completed
Emp: King

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

SQL Dinámico con una Consulta de una Sola Fila

El ejemplo de código de la diapositiva es un ejemplo de definición de una sentencia SQL dinámica con el método 3 con una consulta de una sola fila, es decir, una consulta con un número conocido de elementos de la lista select y variables del host de entrada.

En el ejemplo de consulta de una sola fila se 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.

Nota

- Para obtener un ejemplo de SQL dinámico con una consulta de varias filas mediante REF CURSORS, consulte `demo_06_13_a` en la carpeta `/home/oracle/labs/plpu/demo`.
- Para obtener un ejemplo del uso de REF CURSORS, consulte `demo_06_13_b` en la carpeta `/home/oracle/labs/plpu/demo`.
- REF CURSORS se tratan en *Oracle Database 11g: PL/SQL Avanzado u Oracle Database 10g: PL/SQL Avanzado*.

Ejecución Dinámica de un Bloque PL/SQL Anónimo

```

CREATE FUNCTION annual_sal( p_emp_id NUMBER)
RETURN NUMBER IS
    v_plsql varchar2(200) := 
        'DECLARE |||
        ' rec_emp employees%ROWTYPE; |||
        'BEGIN |||
        '    rec_emp := get_emp(:empid); |||
        '    :res := rec_emp.salary * 12; |||
        'END;';
    v_result NUMBER;
BEGIN
    EXECUTE IMMEDIATE v_plsql
        USING IN p_emp_id, OUT v_result;
    RETURN v_result;
END;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(annual_sal(100))

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 de enlace para:

- La entrada del identificador de empleado utilizando el marcador de posición `:empid`
- El resultado de salida que calcula el salario anual de los empleados utilizando el marcador de posición `: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.

La salida del ejemplo de la diapositiva es la siguiente:

```

FUNCTION annual_sal(emp_id Compiled.
anonymous block completed
316800

```

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(p_name VARCHAR2,
    p_plsql_type VARCHAR2, p_options VARCHAR2 := NULL) IS
    v_stmt varchar2(200) := 'ALTER '|| p_plsql_type |||
                           ' '|| p_name ||' COMPILE';
BEGIN
    IF p_options IS NOT NULL THEN
        v_stmt := v_stmt || ' '|| p_options;
    END IF;
    EXECUTE IMMEDIATE v_stmt;
END;
/
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 diapositiva 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')
```

A continuación se muestra un ejemplo de compilación con debug activado para la función `get_emp`:

```
EXEC compile_plsql ('get_emp', 'function', 'debug')
```

Agenda de la Lección

- Uso de SQL dinámico nativo (NDS)
- Uso del paquete DBMS_SQL

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso del Paquete DBMS_SQL

- El paquete DBMS_SQL se utiliza para escribir SQL dinámico en procedimientos almacenados y para analizar sentencias DDL.
- Debe utilizar el paquete DBMS_SQL para ejecutar una sentencia SQL dinámica que tiene un número desconocido de variables de entrada o salida, también denominado método 4.
- En la mayoría de los casos, NDS es más fácil de utilizar y se ejecuta mejor que DBMS_SQL, excepto cuando se trabaja con el método 4.
- Por ejemplo, debe utilizar el paquete DBMS_SQL en las siguientes situaciones:
 - No conoce la lista SELECT en el tiempo de compilación
 - No sabe cuántas columnas devolverá una sentencia SELECT o qué tipos de dato contendrán

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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.

Método 4: el método 4 hace referencia a situaciones en las que, en una sentencia SQL dinámica, el número de columnas seleccionadas para una consulta o el número de variables de enlace definido no se conoce hasta el tiempo de ejecución. En este caso, debe utilizar el paquete DBMS_SQL.

Al generar SQL dinámico, puede utilizar el paquete DBMS_SQL proporcionado si se trata de situaciones del método 4 o puede utilizar SQL dinámico nativo. Antes de Oracle Database 11g, cada uno de estos métodos tenía limitaciones funcionales. En Oracle Database 11g, se ha agregado funcionalidad a ambos métodos para que sean más completos.

Las funciones para ejecutar SQL dinámico desde PL/SQL tenían algunas restricciones en Oracle Database 10g. Se necesitaba DBMS_SQL para supuestos del método 4, pero no podía manejar el rango completo de tipos de dato y el cliente no podía utilizar su representación de cursor para la base de datos. El SQL dinámico nativo era más adecuado para supuestos que no fueran del método 4, pero no soportaba sentencias de más de 32 KB. Oracle Database 11g elimina éstas y otras restricciones, para que el soporte de SQL dinámico desde PL/SQL sea funcionalmente completo.

Uso de los Subprogramas del Paquete DBMS_SQL

Ejemplos de los procedimientos y las funciones del paquete:

- OPEN_CURSOR
- PARSE
- BIND_VARIABLE
- EXECUTE
- FETCH_ROWS
- CLOSE_CURSOR



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de los Subprogramas del Paquete DBMS_SQL

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. Toda sentencia SQL debe analizarse llamando a los procedimientos PARSE. Al analizar la sentencia, se comprueba su sintaxis y se asocia al cursor en el programa. Puede analizar cualquier sentencia DML o DDL. Las sentencias DDL se ejecutan inmediatamente al ser analizadas.
- BIND_VARIABLE para enlazar un valor determinado a una variable de enlace identificada por su nombre en la sentencia que se está analizando. Esto no es necesario si la sentencia no tiene variables de enlace.
- 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 los Subprogramas del Paquete DBMS_SQL (continuación)

Parámetros del Procedimiento PARSE

El parámetro LANGUAGE_FLAG del procedimiento PARSE determina cómo maneja Oracle la sentencia SQL, es decir, el comportamiento de uso asociado a una versión de base de datos Oracle concreta. El uso de NATIVE (o 1) para este parámetro especifica el uso del comportamiento normal asociado a la base de datos a la que está conectado el programa.

Si el parámetro LANGUAGE_FLAG se define en V6 (o 0), se especifica el comportamiento de la versión 6. Si el parámetro LANGUAGE_FLAG se define en V7 (o 2), se especifica el comportamiento de la versión 7 de la base de datos Oracle.

Nota: para obtener más información, consulte *Oracle Database PL/SQL Packages and Types Reference* (Referencia de Tipos y Paquetes PL/SQL de Oracle Database).

Uso de DBMS_SQL con una Sentencia DML: Supresión de Filas

```

CREATE OR REPLACE FUNCTION delete_all_rows
  (p_table_name  VARCHAR2) RETURN NUMBER IS
    v_cur_id      INTEGER;
    v_rows_del    NUMBER;
BEGIN
  v_cur_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQLPARSE(v_cur_id,
    'DELETE FROM ' || p_table_name, DBMS_SQL.NATIVE);
  v_rows_del := DBMS_SQL.EXECUTE (v_cur_id);
  DBMS_SQL.CLOSE_CURSOR(v_cur_id);
  RETURN v_rows_del;
END;
/

```

```

CREATE TABLE temp_emp AS SELECT * FROM employees;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Rows Deleted: ' || 
delete_all_rows('temp_emp'));
END;
/

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de DBMS_SQL con una Sentencia DML

En la diapositiva, 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 de 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 0 para el número de filas procesadas, ya que las sentencias DDL no procesan filas.

```

CREATE TABLE succeeded.
anonymous block completed
Rows Deleted: 107

DROP TABLE temp_emp succeeded.

```

Uso de DBMS_SQL con una Sentencia DML con Parámetros

```

CREATE PROCEDURE insert_row (p_table_name VARCHAR2,
                            p_id VARCHAR2, p_name VARCHAR2, p_region NUMBER) IS
    v_cur_id      INTEGER;
    v_stmt        VARCHAR2(200);
    v_rows_added  NUMBER;
BEGIN
    v_stmt := 'INSERT INTO '|| p_table_name ||
              ' VALUES (:cid, :cname, :rid)';
    v_cur_id := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQLPARSE(v_cur_id, v_stmt, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cid', p_id);
    DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cname', p_name);
    DBMS_SQL.BIND_VARIABLE(v_cur_id, ':rid', p_region);
    v_rows_added := DBMS_SQL.EXECUTE(v_cur_id);
    DBMS_SQL.CLOSE_CURSOR(v_cur_id);
    DBMS_OUTPUT.PUT_LINE(v_rows_added||' row added');
END ;
/

```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de DBMS_SQL con una Sentencia DML con Parámetros

En el ejemplo de la diapositiva se realiza la operación DML de insertar una fila en la tabla especificada. El ejemplo demuestra el paso adicional necesario para asociar valores a variables de enlace que existen en la sentencia SQL. Por ejemplo, una llamada al procedimiento de la diapositiva es:

```
EXECUTE insert_row('countries', 'LB', 'Lebanon', 4)
```

Después de analizar la sentencia, debe llamar al procedimiento DBMS_SQL.BIND_VARIABLE para asignar valores para cada variable de enlace 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 de enlace en la consulta.
3. Para cada fila, realice los siguientes pasos:
 - a. Ejecute DBMS_SQL.FETCH_ROWS para recuperar una fila y devolver el número de filas recuperadas. Pare el procesamiento adicional cuando se devuelva un valor cero.
 - b. Ejecute DBMS_SQL.COLUMN_VALUE para recuperar el valor de cada columna seleccionada en cada 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.

Prueba

Todo el texto de la sentencia SQL dinámica puede ser desconocido hasta el momento de la ejecución; por lo tanto, su sintaxis se comprueba en *tiempo de ejecución* en lugar de en el *tiempo de compilación*.

- 1. Verdadero
- 2. Falso

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: 1

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Describir el flujo de ejecución de sentencias SQL
- Crear y ejecutar sentencias SQL dinámicamente utilizando SQL dinámico nativo (NDS)
- Identificar situaciones en las que debe utilizar el paquete DBMS_SQL en lugar de NDS para crear y ejecutar sentencias SQL dinámicamente



Copyright © 2010, Oracle y/o sus filiales. 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.

Visión General de la Práctica 6: Uso de SQL Dinámico Nativo

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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica 6: Visión General

En esta práctica, escribirá código para realizar las siguientes tareas:

- Crear un paquete que utilice SQL dinámico nativo para crear o borrar una tabla y para llenar, modificar y suprimir filas de una tabla.
- Crear 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`.

Consideraciones de Diseño para Código PL/SQL

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

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

- Crear constantes y excepciones estándar
- Escribir y llamar a subprogramas locales
- Controlar los privilegios en tiempo de ejecución de un subprograma
- Realizar transacciones autónomas
- Transferir parámetros por referencia mediante una indicación NOCOPY
- Utilizar la indicación PARALLEL ENABLE para la optimización
- Usar la caché de resultados de funciones PL/SQL entre sesiones
- Utilizar la cláusula DETERMINISTIC con funciones
- Utilizar la cláusula RETURNING y el enlace en bloque con DML



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos de la Lección

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 pueden 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.

Agenda de la Lección

- Estandarización de constantes y excepciones, con subprogramas locales, control de los privilegios en tiempo de ejecución de un subprograma y realización de transacciones autónomas
- Uso de las indicaciones NOCOPY y PARALLEL ENABLE, la caché de resultados de funciones PL/SQL entre sesiones y la cláusula DETERMINISTIC
- Uso de la cláusula RETURNING y el enlace en bloque con DML

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Estandarización de Constantes y Excepciones

Las constantes y las excepciones se suelen implantar mediante un paquete sin cuerpo (es decir, 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
 - Implantar los estándares de la compañía en aplicaciones completas
- Empezar con la estandarización de:
 - Nombres de excepciones
 - Definiciones de constantes



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Estandarización de Constantes y Excepciones

Cuando varios desarrolladores escriben sus propios manejadores de excepciones en una aplicación, puede 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:

- Implantar 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 implantar 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
    e_fk_err      EXCEPTION;
    e_seq_nbr_err EXCEPTION;
    PRAGMA EXCEPTION_INIT (e_fk_err, -2292);
    PRAGMA EXCEPTION_INIT (e_seq_nbr_err, -2277);
    ...
END error_pkg;
/
```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Estandarización de Excepciones

En el ejemplo de la diapositiva, 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 en la diapositiva 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, como en el ejemplo siguiente:

```
BEGIN
    DELETE FROM departments
    WHERE department_id = deptno;
    ...
EXCEPTION
    WHEN error_pkg.e_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`



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Estandarización del Manejo de Excepciones

El manejo de excepciones estandarizado se puede implantar 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:

- Convierta las variables en constantes para reducir el mantenimiento y la depuración
- Cree una especificación del paquete central y coloque 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;
```



Copyright © 2010, Oracle y/o sus filiales. 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 diapositiva 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 PROCEDURE o FUNCTION definido al final de la sección de declaraciones de un subprograma.

```
CREATE PROCEDURE employee_sal(p_id NUMBER) IS
    v_emp employees%ROWTYPE;
    FUNCTION tax(p_salary VARCHAR2) RETURN NUMBER IS
    BEGIN
        RETURN p_salary * 0.825;
    END tax;
BEGIN
    SELECT * INTO v_emp
    FROM EMPLOYEES WHERE employee_id = p_id;
    DBMS_OUTPUT.PUT_LINE('Tax: ' || tax(v_emp.salary));
END;
/
EXECUTE employee_sal(100)
```

```
PROCEDURE employee_sal(p_id Compiled.
anonymous block completed
Tax: 19800
```

Copyright © 2010, Oracle y/o sus filiales. 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, defínala como un subprograma local.

Puede definir un bloque PL/SQL con nombre en la sección de declaraciones de cualquier 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 diapositiva 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:

- 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:

- 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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Derechos del Responsable de la Definición frente a Derechos del Invocador

Modelo de Derechos del Responsable de la Definición

Por defecto, 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: Definición de AUTHID en CURRENT_USER

```
CREATE OR REPLACE PROCEDURE add_dept(
    p_id NUMBER, p_name VARCHAR2) AUTHID CURRENT_USER IS
BEGIN
    INSERT INTO departments
    VALUES (p_id, p_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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Especificación de Derechos del Invocador

Puede definir los derechos del invocador para distintas construcciones de subprogramas PL/SQL de la siguiente forma:

```
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...
```

El valor por defecto es AUTHID DEFINER, que especifica que el subprograma se ejecuta con los privilegios de su propietario. La mayoría de paquetes PL/SQL proporcionados como DBMS_LOB, DBMS_ROWID, etc. tienen derechos del invocador.

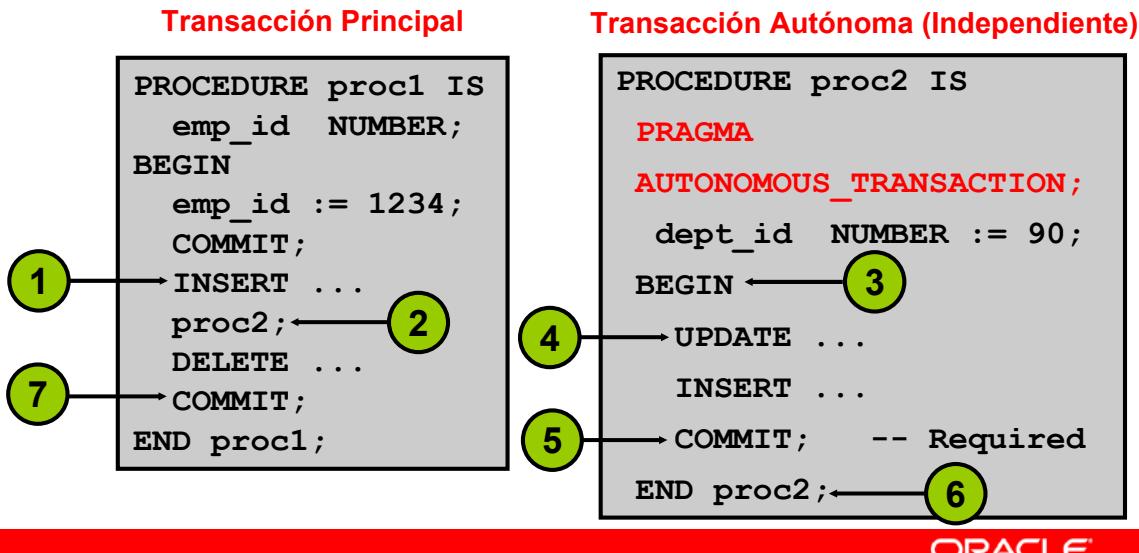
Resolución de Nombres

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, lenguaje de manipulación de datos (DML), SQL dinámico y el paquete 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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

ORACLE

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 diapositiva 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

- Son independientes de la transacción principal
- Suspender la transacción que realiza la llamada hasta que terminan las transacciones autónomas
- 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
- Las inician y finalizan subprogramas individuales y no bloques PL/SQL anónimos o anidados



Copyright © 2010, Oracle y/o sus filiales. 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

```

CREATE TABLE usage (card_id NUMBER, loc NUMBER)
/
CREATE TABLE txn (acc_id NUMBER, amount NUMBER)
/
CREATE OR REPLACE PROCEDURE log_usage (p_card_id NUMBER, p_loc NUMBER)
IS
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  INSERT INTO usage
  VALUES (p_card_id, p_loc);
  COMMIT;
END log_usage;
/
CREATE OR REPLACE PROCEDURE bank_trans(p_cardnbr NUMBER,p_loc NUMBER) IS
BEGIN
  INSERT INTO txn VALUES (9001, 1000);
  log_usage (p_cardnbr, p_loc);
END bank_trans;
/
EXECUTE bank_trans(50, 2000)

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Transacciones Autónomas

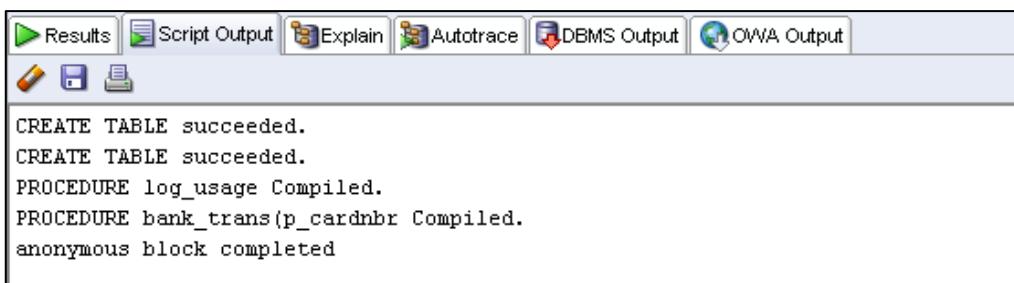
Para definir transacciones autónomas, utilice PRAGMA AUTONOMOUS_TRANSACTION. PRAGMA indica al compilador PL/SQL que marque una rutina como autónoma (independiente). En este contexto, el término “rutina” incluye bloques PL/SQL anónimos de nivel superior (no anidados); funciones y procedimientos locales, 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 diapositiva, 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 pueden 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.

Uso de Transacciones Autónomas (continuación)

El resultado de los ejemplos de la diapositiva anterior, las tablas TXN y USAGE son los siguientes:



```

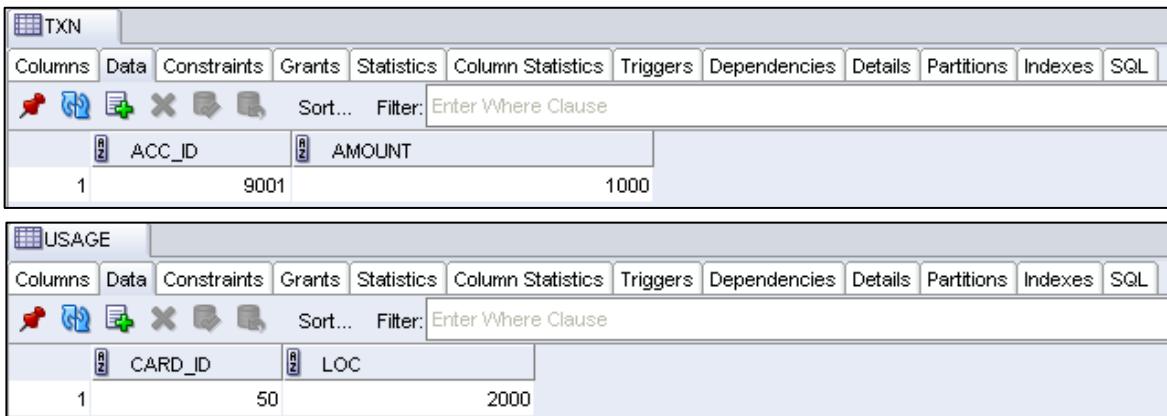
Results Script Output Explain Autotrace DBMS Output OWA Output
CREATE TABLE succeeded.
CREATE TABLE succeeded.
PROCEDURE log_usage Compiled.
PROCEDURE bank_trans(p_cardnbr Compiled.
anonymous block completed

```

Para ejecutar el código de la página anterior, introduzca el siguiente código:

```
EXECUTE bank_trans(50, 2000)
```

Utilice el separador Data en el nodo Tables del árbol Object Navigator para mostrar los valores de las tablas TXN y USAGE de la siguiente forma:



TXN		
Columns	Data	Constraints
	ACC_ID AMOUNT	
1	9001 1000	

USAGE		
Columns	Data	Constraints
	CARD_ID LOC	
1	50 2000	

Agenda de la Lección

- Estandarización de constantes y excepciones, con subprogramas locales, control de los privilegios en tiempo de ejecución de un subprograma y realización de transacciones autónomas
- Uso de las indicaciones NOCOPY y PARALLEL ENABLE, la caché de resultados de funciones PL/SQL entre sesiones y la cláusula DETERMINISTIC
- Uso de la cláusula RETURNING y el enlace en bloque con DML

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de la Indicación NOCOPY

- Permite al compilador PL/SQL 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      rec_emp_type IS TABLE OF employees%ROWTYPE;
    rec_emp  rec_emp_type;
    PROCEDURE populate(p_tab IN OUT NOCOPY emptabtype) IS
        BEGIN
            . . .
        END;
    BEGIN
        populate(rec_emp);
    END;
/

```



Copyright © 2010, Oracle y/o sus filiales. 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 copiar parámetros que representen grandes estructuras de datos (como recopilaciones, registros e instancias de tipos de objetos) 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 diapositiva 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.



Copyright © 2010, Oracle y/o sus filiales. 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.

¿Cuándo Ignora el Compilador PL/SQL la Indicación NOCOPY?

La indicación NOCOPY no tiene ningún efecto si:

- El parámetro real:
 - Es un elemento de matrices asociativas (tablas 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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

¿Cuándo Ignora el Compilador PL/SQL la Indicación NOCOPY?

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 matrices asociativas (tablas de índice). Esta restricción no se aplica a matrices asociativas 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.

Uso de la Indicación PARALLEL_ENABLE

- Se puede utilizar en funciones como indicación de optimización
- Indica que una función se puede utilizar en una consulta con paralelismo o en una sentencia DML con paralelismo

```
CREATE OR REPLACE FUNCTION f2 (p_p1 NUMBER)
  RETURN NUMBER PARALLEL_ENABLE IS
BEGIN
  RETURN p_p1 * 2;
END f2;
```

FUNCTION f2 Compiled.

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de la 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, antes de Oracle8i, la optimización del paralelismo buscaba si una función se anotaba como que tenía los cuatro RNDS, WNDS, 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, se examinaba implícitamente el código de estas funciones definidas con una sentencia CREATE FUNCTION 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 diapositiva.

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.

Uso de la Caché de Resultados de Funciones PL/SQL entre Sesiones

- Cada vez que se llama a una función PL/SQL de resultados almacenados en caché con distintos valores de parámetros, dichos parámetros y sus resultados se almacenan en caché.
- La caché de resultados de las funciones se almacena en un área global compartida (SGA), con lo que está disponible para cualquier sesión que ejecute la aplicación.
- Las llamadas posteriores a la misma función con los mismos parámetros utilizan el resultado de la caché.
- Se mejora el rendimiento y la escalabilidad.
- Esta función se utiliza con funciones que se llaman frecuentemente y que dependen de información que cambia con poca frecuencia.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Caché de Resultados de Funciones PL/SQL entre Sesiones

A partir de Oracle Database 11g, se puede utilizar el mecanismo de almacenamiento en caché de los resultados de funciones PL/SQL entre sesiones. Este mecanismo de almacenamiento en caché proporciona un medio compatible con el lenguaje y gestionado por el sistema para almacenar los resultados de las funciones PL/SQL en un área global compartida (SGA), que está disponible para todas las sesiones que ejecute la aplicación. El mecanismo de almacenamiento en caché es, a la vez, eficiente y fácil de usar, y libera del trabajo que supone diseñar y desarrollar cachés y políticas de gestión de cachés propias.

Cada vez que se llama a una función PL/SQL de resultados almacenados en caché con distintos valores de parámetros, dichos parámetros y sus resultados se almacenan en la caché. Posteriormente, cuando se llama a la misma función con los mismos valores de parámetros, el resultado se recupera de la caché, en lugar de volverlo a calcular. Si se actualiza un objeto de base de datos utilizado para calcular un resultado almacenado en caché, éste se convierte en no válido y se debe volver a calcular.

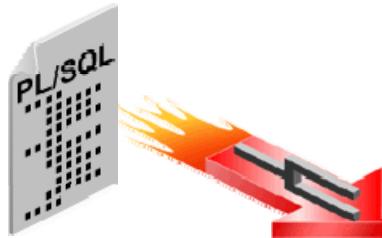
Utilice la función de almacenamiento en caché de resultados con funciones que se llamen frecuentemente y que dependan de información que cambia con poca frecuencia o nunca.

Nota: para obtener más información sobre la *caché de resultados de funciones PL/SQL entre sesiones*, consulte el curso *Oracle Database 11g: PL/SQL Avanzado*, el curso *Oracle Database 11g: Nuevas Funciones de SQL y PL/SQL* u *Oracle Database PL/SQL Language Reference 11g Release 2 (11.2)* [Referencia del Lenguaje PL/SQL de Oracle Database 11g Versión 2 (11.2)].

Activación del Almacenamiento en Caché de Resultados para una Función

Puede hacer que una función almacene resultados en caché de la forma siguiente:

- Incluya la cláusula `RESULT_CACHE` en lo siguiente:
 - Declaración de funciones
 - Definición de función
- Incluya una cláusula `RELIES_ON` opcional para especificar cualquier tabla o vista de la cual dependan los resultados de la función.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Activación del Almacenamiento en Caché de Resultados para una Función

Para activar el almacenamiento en caché de resultados para una función PL/SQL, utilice la cláusula `RESULT_CACHE`. Cuando se llama a una función de resultados almacenados en caché, el sistema comprueba la caché de resultados de la función. Si la caché contiene el resultado de una llamada anterior a la función con los mismos valores de parámetros, el sistema devuelve el resultado almacenado en caché y no vuelve a ejecutar el cuerpo de la función. Si la caché no contiene el resultado, el sistema ejecuta el cuerpo de la función y agrega el resultado (para estos valores de parámetros) a la caché antes de devolver el control al emisor de la llamada.

La caché puede acumular muchos resultados: un resultado para cada combinación única de valores de parámetros con los que se ha llamado a cada función de resultados almacenados en caché. Si el sistema necesita más memoria, hace que uno o más resultados almacenados en caché queden obsoletos (los suprime).

Nota: si la ejecución de la función provoca una excepción no tratada, el resultado de la excepción no se almacena en la caché.

Declaración y Definición de una Función de Resultados Almacenados en Caché: Ejemplo

```

CREATE OR REPLACE FUNCTION emp_hire_date (p_emp_id
    NUMBER) RETURN VARCHAR
RESULT_CACHE RELIES_ON (employees) IS
    v_date_hired DATE;
BEGIN
    SELECT hire_date INTO v_date_hired
    FROM HR.Employees
    WHERE Employee_ID = p_emp_ID;
    RETURN to_char(v_date_hired);
END ;

```

FUNCTION emp_hire_date Compiled.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Declaración y Definición de una Función de Resultados Almacenados en Caché: Ejemplo

Si una función depende de una configuración que puede variar de una sesión a otra, como NLS_DATE_FORMAT y TIME_ZONE, haga que la función almacene resultados en caché sólo si puede modificarla para manejar varias configuraciones.

En el ejemplo de la diapositiva, la función emp_hire_date utiliza la función to_char para convertir un elemento DATE en un elemento VARCHAR. emp_hire_date no especifica ninguna máscara de formato, por lo que ésta se define por defecto en la que especifica NLS_DATE_FORMAT. Si las sesiones que llaman a emp_hire_date tienen una configuración NLS_DATE_FORMAT distinta, los resultados almacenados en caché pueden tener formatos distintos. Si un resultado almacenado en caché calculado por una sesión queda obsoleto y otra sesión lo vuelve a calcular, el formato puede variar incluso para el mismo valor de parámetro. Si una sesión obtiene un resultado almacenado en caché cuyo formato difiere de su propio formato, el resultado probablemente sea incorrecto.

Algunas de las soluciones posibles a este problema son las siguientes:

- Cambie el tipo de retorno de emp_hire_date a DATE y haga que cada sesión llame a la función to_char.
- Si es aceptable un formato común para todas las sesiones, especifique una máscara de formato, eliminando la dependencia de NLS_DATE_FORMAT, por ejemplo,
to_char(date_hired, 'mm/dd/yy') ;.

Declaración y Definición de una Función de Resultados Almacenados en Caché: Ejemplo (continuación)

- Agregue un parámetro de máscara de formato a HireDate de la siguiente forma:

```
CREATE OR REPLACE FUNCTION emp_hire_date (p_emp_id NUMBER, fmt
                                         VARCHAR) RETURN VARCHAR
  RESULT_CACHE RELIES_ON (employees) IS
    v_date_hired DATE;
BEGIN
  SELECT hire_date INTO v_date_hired
  FROM employees
  WHERE employee_id = p_emp_id;
  RETURN to_char(v_date_hired, fmt);
END;
```

Uso de la Cláusula DETERMINISTIC con Funciones

- Especifique DETERMINISTIC para indicar que la función devuelve el mismo valor de resultado si se llama con los mismos valores para sus argumentos.
- Esto ayuda al optimizador a evitar llamadas a funciones redundantes.
- Si se ha llamado anteriormente a una función con los mismos argumentos, el optimizador puede elegir utilizar el resultado anterior.
- No especifique DETERMINISTIC para una función cuyo resultado depende del estado de las variables de sesión u objetos de esquema.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de la Cláusula DETERMINISTIC con Funciones

Puede utilizar la cláusula de función DETERMINISTIC para indicar que la función devuelve el mismo valor de resultado si se llama con los mismos valores para sus argumentos.

Debe especificar esta palabra clave si pretende llamar a la función en la expresión de un índice basado en funciones o desde la consulta de una vista materializada marcada como REFRESH FAST o ENABLE QUERY REWRITE. Cuando Oracle Database encuentra una función determinista en uno de estos contextos, intenta utilizar los resultados calculados anteriormente cuando es posible, en lugar de volver a ejecutar la función. Si posteriormente cambia la semántica de la función, debe volver a crear manualmente todos los índices basados en funciones y vistas materializadas dependientes.

No especifique esta cláusula para definir una función que utiliza variables de paquetes o que accede a la base de datos de cualquier forma que pueda afectar al resultado devuelto de la función. Los resultados de ello no se capturarán si Oracle Database decide no volver a ejecutar la función.

Nota

- No especifique DETERMINISTIC para una función cuyo resultado depende del estado de las variables de sesión u objetos de esquema porque los resultados pueden variar según la llamada. En su lugar, considere hacer que la función almacene los resultados en caché.
- Para obtener más información acerca de la cláusula DETERMINISTIC, consulte *Oracle Database SQL Language Reference* (Referencia del Lenguaje PL/SQL de Oracle Database).

Agenda de la Lección

- Estandarización de constantes y excepciones, con subprogramas locales, control de los privilegios en tiempo de ejecución de un subprograma y realización de transacciones autónomas
- Uso de las indicaciones NOCOPY y PARALLEL ENABLE, la caché de resultados de funciones PL/SQL entre sesiones y la cláusula DETERMINISTIC
- Uso de la cláusula RETURNING y el enlace en bloque con DML

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de 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 OR REPLACE PROCEDURE update_salary(p_emp_id
NUMBER) IS
    v_name      employees.last_name%TYPE;
    v_new_sal   employees.salary%TYPE;
BEGIN
    UPDATE employees
        SET salary = salary * 1.1
        WHERE employee_id = p_emp_id
        RETURNING last_name, salary INTO v_name, v_new_sal;
        DBMS_OUTPUT.PUT_LINE(v_name || ' new salary is ' ||
        v_new_sal);
END update_salary;
/

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de la 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 ficha después de un 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 cursorios y menos memoria del servidor.

En el ejemplo de la diapositiva 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. Para probar el ejemplo de código, emita los siguientes comandos que comprueban el salario de employee_id 108 antes de actualizarlo. A continuación, se llama al procedimiento transfiriendo employee_id 108 como parámetro.

```

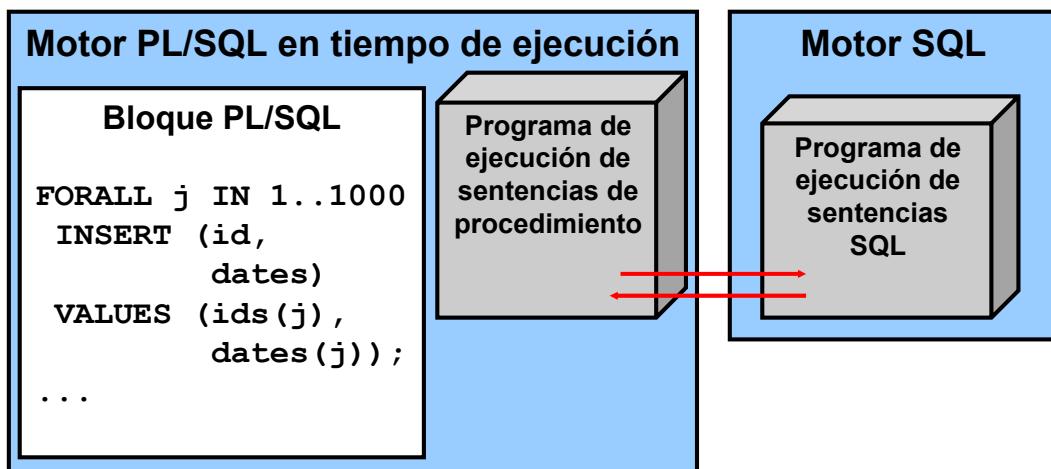
1|SET SERVEROUTPUT ON
2|
3|SELECT last_name, salary
4|FROM employees
5|WHERE employee_id = 108;
6|
7|EXECUTE update_salary(108)

```

LAST_NAME	SALARY
Greenberg	12000
1 rows selected	
anonymous block completed	
Greenberg new salary is 13200	

Uso de Enlaces 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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Enlaces en Bloque

El servidor de Oracle utiliza dos motores para ejecutar subprogramas y bloques PL/SQL:

- El motor PL/SQL en tiempo de ejecución, que ejecuta sentencias de procedimiento, pero transfiere las sentencias SQL al motor SQL.
- El motor SQL, que analiza y ejecuta la sentencia SQL y, en algunos casos, devuelve datos al motor PL/SQL.

Durante la ejecución, cada una de las sentencias SQL provoca un intercambio de contexto entre los dos motores, lo que produce una penalización de rendimiento para cantidades excesivas de procesamiento SQL. Esto es típico de aplicaciones que tienen una sentencia SQL en un bucle que utiliza valores de recopilaciones indexadas. 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 motor SQL. Es decir, un proceso de enlace en bloque transfiere la recopilación completa de valores entre los dos motores 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.

Enlaces en Bloque: Sintaxis y Palabras Clave

- La palabra clave `FORALL` indica al *motor PL/SQL* que debe realizar un enlace en bloque de las recopilaciones de entrada antes de enviarlas al motor SQL.

```
FORALL index IN lower_bound ... upper_bound
  [SAVE EXCEPTIONS]
  sql_statement;
```

- La palabra clave `BULK COLLECT` indica al *motor SQL* que debe realizar un enlace en bloque de las recopilaciones de salida antes de devolverlas al motor PL/SQL.

```
... BULK COLLECT INTO
  collection_name[,collection_name] ...
```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Enlaces en Bloque: Sintaxis y Palabras Clave

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`

La palabra clave `FORALL` indica al motor PL/SQL que debe realizar un enlace en bloque de las recopilaciones de entrada antes de enviarlas al motor SQL. Aunque la sentencia `FORALL` contiene un esquema de iteración, no es un bucle FOR.

La palabra clave `BULK COLLECT` indica al motor SQL que debe realizar un enlace en bloque de las recopilaciones de salida antes de devolverlas al motor 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.

Enlaces en Bloque: Sintaxis y Palabras Clave (continuación)

Manejo de Excepciones de FORALL con el Atributo %BULK_EXCEPTIONS

Para gestionar excepciones y para que se termine el enlace en bloque a pesar de los errores, agregue las palabras claves SAVE EXCEPTIONS a la sentencia FORALL después de los límites y antes de la sentencia DML.

Todas las excepciones emitidas durante la ejecución se guardan en el atributo cursor %BULK_EXCEPTIONS, que almacena una recopilación de registros. Cada registro tiene dos campos: %BULK_EXCEPTIONS(i).ERROR_INDEX contiene la "iteración" de la sentencia FORALL durante la que se emitió la excepción y %BULK_EXCEPTIONS(i).ERROR_CODE contiene el código de error de Oracle correspondiente.

Los valores almacenados en %BULK_EXCEPTIONS hacen referencia a la sentencia FORALL ejecutada más recientemente. Sus scripts secundarios varían de 1 a %BULK_EXCEPTIONS.COUNT.

Nota: para obtener más información sobre los enlaces en bloque y el manejo de excepciones de enlaces en bloque, consulte *Oracle Database PL/SQL User's Guide and Reference (Guía del Usuario y Referencia de PL/SQL de Oracle Database)*

Enlace en Bloque FORALL: Ejemplo

```

CREATE PROCEDURE raise_salary(p_percent NUMBER) IS
    TYPE numlist_type IS TABLE OF NUMBER
        INDEX BY BINARY_INTEGER;
    v_id  numlist_type; -- collection
BEGIN
    v_id(1) := 100; v_id(2) := 102; v_id(3) := 104; v_id(4) := 110;
    -- bulk-bind the PL/SQL table
    FORALL i IN v_id.FIRST .. v_id.LAST
        UPDATE employees
            SET salary = (1 + p_percent/100) * salary
            WHERE employee_id = v_id(i);
END;
/

```

```
EXECUTE raise_salary(10)
```

anonymous block completed

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Enlace en Bloque FORALL: Ejemplo

Nota: para poder ejecutar el ejemplo de la diapositiva, debe desactivar el disparador update_job_history de la siguiente forma:

```
ALTER TRIGGER update_job_history DISABLE;
```

En el ejemplo de la diapositiva, el bloque PL/SQL aumenta el salario de los empleados con los identificadores 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 motor SQL para cada registro de empleado actualizado. Si hay muchos registros de empleado que actualizar, el gran número de intercambios de contexto entre el motor PL/SQL y el motor 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.

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^{a} 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 num_list_type IS TABLE OF NUMBER
        INDEX BY BINARY_INTEGER;
    v_nums num_list_type;
BEGIN
    v_nums(1) := 1;
    v_nums(2) := 3;
    v_nums(3) := 5;
    v_nums(4) := 7;
    v_nums(5) := 11;
    FORALL i IN v_nums.FIRST .. v_nums.LAST
        INSERT INTO v_num_table (n) VALUES (v_nums(i));
    FOR i IN v_nums.FIRST .. v_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:

```

CREATE TABLE succeeded.
anonymous block completed
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

DROP TABLE num_table succeeded.

```

Uso de BULK COLLECT INTO con Consultas

La sentencia SELECT soporta la sintaxis BULK COLLECT INTO.

```
CREATE PROCEDURE get_departments(p_loc NUMBER) IS
    TYPE dept_tab_type IS
        TABLE OF departments%ROWTYPE;
    v_depts dept_tab_type;
BEGIN
    SELECT * BULK COLLECT INTO v_depts
    FROM departments
    WHERE location_id = p_loc;
    FOR i IN 1 .. v_depts.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE(v_depts(i).department_id
            ||' '|| v_depts(i).department_name);
    END LOOP;
END ;
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de BULK COLLECT INTO con Consultas

A partir de Oracle Database 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 diapositiva. 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 soportar la sintaxis `BULK COLLECT INTO`.

```

CREATE OR REPLACE PROCEDURE get_departments(p_loc NUMBER) IS
  CURSOR cur_dept IS
    SELECT * FROM departments
    WHERE location_id = p_loc;
  TYPE dept_tab_type IS TABLE OF cur_dept%ROWTYPE;
  v_depts dept_tab_type;
BEGIN
  OPEN cur_dept;
  FETCH cur_dept BULK COLLECT INTO v_depts;
  CLOSE cur_dept;
  FOR i IN 1 .. v_depts.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_depts(i).department_id
      ||' '|| v_depts(i).department_name);
  END LOOP;
END;

```

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de BULK COLLECT INTO con Cursos

En Oracle Database 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 diapositiva.

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 diapositiva se puede modificar de la siguiente forma:

```

CREATE OR REPLACE PROCEDURE get_departments(p_loc NUMBER,
  p_nrows NUMBER) IS
  CURSOR dept_csr IS SELECT *
    FROM departments
    WHERE location_id = p_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

```

CREATE OR REPLACE PROCEDURE raise_salary(p_rate NUMBER)
IS
  TYPE emplist_type IS TABLE OF NUMBER;
  TYPE numlist_type IS TABLE OF employees.salary%TYPE
    INDEX BY BINARY_INTEGER;
  v_emp_ids emplist_type :=
    emplist_type(100,101,102,104);
  v_new_sals numlist_type;
BEGIN
  FORALL i IN v_emp_ids.FIRST .. v_emp_ids.LAST
    UPDATE employees
      SET commission_pct = p_rate * salary
    WHERE employee_id = v_emp_ids(i)
    RETURNING salary BULK COLLECT INTO v_new_sals;
  FOR i IN 1 .. v_new_sals.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_new_sals(i));
  END LOOP;
END ;

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de BULK COLLECT INTO con una Cláusula RETURNING

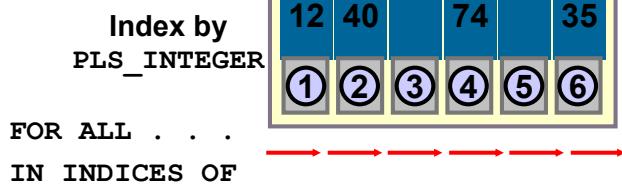
Se pueden utilizar enlaces en bloque para mejorar el rendimiento de los bucles FOR que hacen referencia a recopilaciones y devuelven DML. Si tiene, o planea tener, código PL/SQL que haga esto, puede utilizar la palabra clave FORALL junto con las palabras clave RETURNING y BULK COLLECT INTO para mejorar el rendimiento.

En el ejemplo que aparece en la diapositiva, 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 motor PL/SQL.

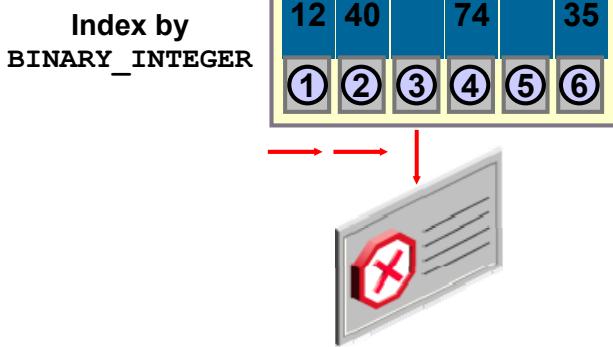
En el ejemplo de la diapositiva se muestra un bucle FOR incompleto que se utiliza para iterar con los datos de nuevo salario recibidos de la operación UPDATE y, a continuación, procesar los resultados.

Uso de Enlaces en Bloque en Recopilaciones Dispersas

Versiones actuales:



Antes de 10g:



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Soporte de FORALL para Recopilaciones Dispersas

- En versiones anteriores, PL/SQL no permitía el uso de recopilaciones dispersas con la sentencia FORALL.
- Si no se especificaba la palabra clave SAVE EXCEPTIONS, la sentencia terminaba cuando se encontraba el primer elemento suprimido. Incluso cuando se utilizaba SAVE EXCEPTION, el motor PL/SQL intentaba iterar en todos los elementos (existentes y no existentes). Esto degradaba considerablemente el rendimiento de la operación DML si el porcentaje relativo de los elementos suprimidos era elevado.
- Cuando utiliza la palabra clave INDICES (disponible con Oracle Database 10g y versiones posteriores), puede realizar correctamente un bucle de una recopilación dispersa con la sentencia FORALL. Esta sintaxis enlaza recopilaciones dispersas de forma más eficaz y también soporta un enfoque más general cuando se puede especificar que una matriz de índice itere las recopilaciones.
- El uso de una recopilación dispersa y matrices de índice en operaciones en bloque mejora el rendimiento.

Uso de Enlaces en Bloque en Recopilaciones Dispersas

```
-- The INDICES OF syntax allows the bound arrays
-- themselves to be sparse.

FORALL index_name IN INDICES OF sparse_array_name
    BETWEEN LOWER_BOUND AND UPPER_BOUND -- optional
    SAVE EXCEPTIONS -- optional, but recommended
    INSERT INTO table_name VALUES
        sparse_array(index_name);
    . . .
```

```
-- The VALUES OF syntax lets you indicate a subset
-- of the binding arrays.

FORALL index_name IN VALUES OF index_array_name
    SAVE EXCEPTIONS -- optional, but recommended
    INSERT INTO table_name VALUES
        binding_array_name(index_name);
    . . .
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Soporte de FORALL para Recopilaciones Dispersas (continuación)

- Puede utilizar la sintaxis INDICES OF y VALUES OF con la sentencia FORALL.
- El enlace en bloque para una sintaxis de matriz dispersa se puede utilizar en todas las sintaxis DML.
- En la sintaxis, la matriz de índice debe ser densa, las matrices de enlace pueden ser densas o dispersas y los elementos indicados deben existir.

Uso de Enlaces en Bloque en Recopilaciones Dispersas

La aplicación típica de esta función es en una entrada de pedido y un sistema de procesamiento de pedidos donde:

- Los usuarios introducen pedidos a través de la web
- Los pedidos se colocan en una tabla temporal antes de la validación
- Los datos se validan más tarde según reglas de negocio complejas (normalmente implantadas mediante programación con PL/SQL)
- Los pedidos no válidos se separan de los válidos
- Los pedidos válidos se insertan en una tabla permanente para su procesamiento



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Enlaces en Bloque en Recopilaciones Dispersas

Esta función se puede utilizar en cualquier aplicación con una tabla PL/SQL densa de registros o una tabla de datos escalares que se rellenan mediante una recopilación en bloque. Se utiliza como la matriz de enlaces. Una matriz densa (puntero), cuyos elementos indican los índices de la matriz de enlaces, se convierte en dispersa según la lógica de la aplicación. A continuación, se utiliza esta matriz de puntero en la sentencia `FORALL` para realizar DML en bloque con las matrices de enlaces. Cualquier excepción encontrada se puede guardar y posteriormente procesar en la sección de manejo de excepciones, quizás mediante otra sentencia `FORALL`.

Uso de Enlaces en Bloque con Matrices de Índice

```
CREATE OR REPLACE PROCEDURE ins_emp2 AS
  TYPE emptab_type IS TABLE OF employees%ROWTYPE;
  v_emp emptab_type;
  TYPE values_of_tab_type IS TABLE OF PLS_INTEGER
    INDEX BY PLS_INTEGER;
  v_num   values_of_tab_type;
  . . .
BEGIN
  . . .
  FORALL k IN VALUES OF v_num
    INSERT INTO new_employees VALUES v_emp(k);
END;
```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Enlaces en Bloque con Matrices de Índice

Puede utilizar una recopilación de índices de PLS_INTEGER o BINARY_INTEGER (o uno de sus subtipos) cuyos valores sean los índices de las recopilaciones implicadas en la operación DML de enlace en bloque con FORALL. A continuación, estas recopilaciones de índices se pueden utilizar en una sentencia FORALL para procesar DML en bloque con la cláusula VALUES OF.

En el ejemplo mostrado, V_NUM es una recopilación cuyo tipo es PLS_INTEGER. En el ejemplo, está creando un procedimiento INS_EMP2, que identifica sólo un empleado para cada incidencia de la primera letra del apellido. A continuación, este procedimiento inserta en la tabla NEW_EMPLOYEES creada anteriormente con la sintaxis FORALL..IN VALUES OF.

Prueba

La indicación `NOCOPY` permite al compilador PL/SQL transferir parámetros `OUT` e `IN OUT` por referencia en lugar de por valor. Esto mejora el rendimiento mediante la reducción de la sobrecarga al transferir parámetros.

1. Verdadero
2. Falso



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: 1

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 copiar parámetros que representen grandes estructuras de datos (como recopilaciones, registros e instancias de tipos de objetos) 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.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Crear constantes y excepciones estándar
- Escribir y llamar a subprogramas locales
- Controlar los privilegios en tiempo de ejecución de un subprograma
- Realizar transacciones autónomas
- Transferir parámetros por referencia mediante una indicación NOCOPY
- Utilizar la indicación PARALLEL ENABLE para la optimización
- Usar la caché de resultados de funciones PL/SQL entre sesiones
- Utilizar la cláusula DETERMINISTIC con funciones
- Utilizar la cláusula RETURNING y el enlace en bloque con DML

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 en las sentencias SQL, y la indicación PARALLEL_ENABLE para la optimización de 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



Copyright © 2010, Oracle y/o sus filiales. 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.

Creación de Disparadores

8

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

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

- Describir los disparadores de base de datos y sus usos
- Describir los diferentes tipos de disparadores
- Crear disparadores de base de datos
- Describir las reglas de arranque de disparadores de base de datos
- Eliminar disparadores de base de datos
- Mostrar la información del disparador



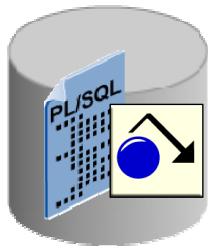
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos de la Lección

En esta lección, aprenderá a crear y utilizar disparadores de base de datos.

¿Qué Son los Disparadores?

- Un disparador es un bloque PL/SQL que se almacena en la base de datos y se arranca (ejecuta) como respuesta a un evento concreto.
- La base de datos Oracle ejecuta automáticamente un disparador cuando se producen condiciones especificadas.



ORACLE

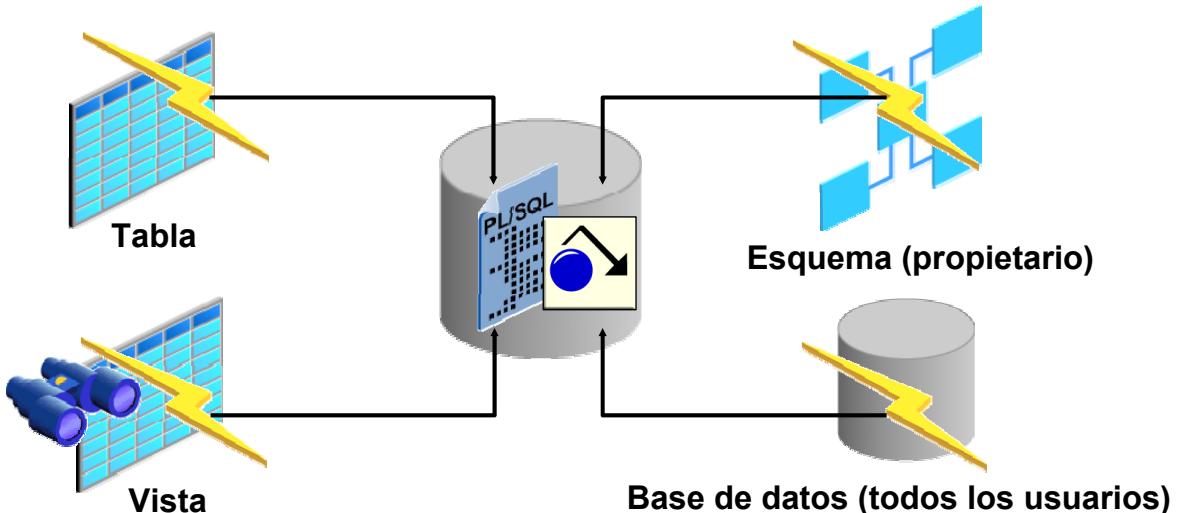
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Trabajar con Disparadores: Visión General

Los disparadores son similares a los procedimientos almacenados. Un disparador almacenado en la base de datos contiene PL/SQL en forma de bloque anónimo, sentencia de llamada o bloque de disparador compuesto. Sin embargo, los procedimientos y los disparadores difieren en la forma en que se llaman. Un usuario, una aplicación o un disparador ejecutan implícitamente un procedimiento. La base de datos Oracle arranca implícitamente un disparador cuando se produce un evento de disparo, independientemente del usuario que esté conectado o de la aplicación que se utilice.

Definición de Disparadores

Un disparador se puede definir en la tabla, la vista, el esquema (propietario de esquema) o la base de datos (todos los usuarios).



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Tipos de Evento de Disparador

Puede escribir disparadores que arranquen siempre que se produzca una de las operaciones siguientes en la base de datos:

- Una sentencia de manipulación de base de datos (DML) (DELETE, INSERT o UPDATE).
- Una sentencia de definición de la base de datos (DDL) (CREATE, ALTER o DROP).
- Una operación de base de datos como SERVERERROR, LOGON, LOGOFF, STARTUP o SHUTDOWN.



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

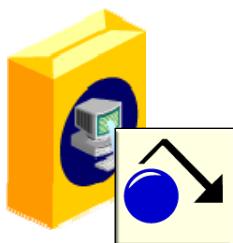
Sentencia o Evento Disparador

Una sentencia o evento disparador es la sentencia SQL, evento de base de dato o evento de usuario que hace que arranque un disparador. Un evento disparador puede ser uno o más de los siguientes:

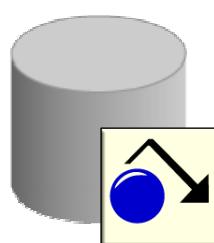
- Una sentencia INSERT, UPDATE o DELETE en una tabla concreta (o vista, en algunos casos)
- Una sentencia CREATE, ALTER o DROP en cualquier objeto de esquema
- Un inicio de la base de datos o cierre de la instancia
- Un mensaje de error específico o cualquier mensaje de error
- Una conexión o desconexión del usuario

Disparadores de Aplicación y de Base de Datos

- Un disparador de base de datos (tratado en este curso):
 - Arranca siempre que se produce un evento del sistema DML o DLL en un esquema o base de datos
- Un disparador de aplicación:
 - Arranca siempre que se produce un evento en una aplicación concreta



Disparador de Aplicación



Disparador de Base de Datos

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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).

Los disparadores de base de datos pueden ser disparadores del sistema en una base de datos o en un esquema (se trata en la siguiente lección). 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. Oracle Forms puede definir, almacenar y ejecutar disparadores de un tipo distinto. Sin embargo, no confunda los disparadores de Oracle Forms con los disparadores tratados en esta lección.

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



Copyright © 2010, Oracle y/o sus filiales. 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 implantar el servidor de Oracle, o como alternativa a las que proporciona el servidor.

- **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 implantan las reglas complejas de integridad.
- **Integridad referencial:** el servidor de Oracle aplica reglas estándar de integridad referencial. Los disparadores implantan 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.

Tipos de Disparadores Disponibles

- Disparadores DML simples
 - BEFORE
 - AFTER
 - INSTEAD OF
- Disparadores compuestos
- Disparadores no DML
 - Disparadores de eventos DDL
 - Disparadores de eventos de base de datos

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Nota

En esta lección, trataremos los disparadores BEFORE, AFTER e INSTEAD OF. Los otros tipos de disparadores se tratan en la lección titulada “Creación de Disparadores Compuestos, de DDL y de Eventos de Base de Datos”.

Tipos de Evento y Cuerpo del Disparador

- Un tipo de evento de disparador determina qué sentencia DML hace que se ejecute el disparador. Los eventos posibles son:
 - INSERT
 - UPDATE [OF column]
 - DELETE
- Un cuerpo del disparador determina qué acción se realiza y es un bloque PL/SQL o CALL para un procedimiento.



Copyright © 2010, Oracle y/o sus filiales. 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 Disparadores de DML mediante la Sentencia CREATE TRIGGER

```

CREATE [OR REPLACE] TRIGGER trigger_name
timing -- when to fire the trigger
event1 [OR event2 OR event3]
ON object_name
[REFERENCING OLD AS old | NEW AS new]
FOR EACH ROW -- default is statement level trigger
WHEN (condition) ]
DECLARE]
BEGIN
... trigger_body -- executable statements
[EXCEPTION . . .]
END [trigger_name];

```

timing = BEFORE | AFTER | INSTEAD OF

event = INSERT | DELETE | UPDATE | UPDATE OF *column_list*

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 cuándo se arranca el disparador en relación con el evento disparador. Los valores son BEFORE, AFTER e INSTEAD OF.
- *event* identifica la operación DML que provoca que 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 (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, implantada 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;

Especificación del Arranque del Disparador (Temporización)

Puede especificar la temporización del disparador como si se debe ejecutar la acción del disparador antes o después de la sentencia disparadora:

- BEFORE: ejecuta el cuerpo del disparador antes del evento DML disparador en una tabla.
- AFTER: ejecuta el cuerpo del disparador después del evento DML disparador en una tabla.
- INSTEAD OF: ejecuta el cuerpo del disparador en lugar de la sentencia disparadora. Se utiliza para vistas que no se pueden modificar de otra forma.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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.

Si es práctico, sustituya el juego de disparadores individuales con distintos puntos de temporización por un único disparador compuesto que codifique explícitamente las acciones en el orden deseado. Si se definen dos o más disparadores con el mismo punto de temporización y el orden en el que arrancan es importante, puede controlar el orden de arranque con las cláusulas **FOLLOWS** y **PRECEDES**.

Disparadores de Nivel de Sentencia frente a Disparadores de Nivel de Fila

Disparadores de Nivel de Sentencia	Disparadores de Nivel de Fila
Son el tipo por defecto al crear un disparador	Utilizan la cláusula FOR EACH ROW al crear un disparador
Arrancan una vez para el evento disparador	Arrancan una vez para cada fila afectada por el evento disparador
Arrancan una vez incluso aunque no haya filas afectadas	No arrancan si el evento disparador no afecta a ninguna fila

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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

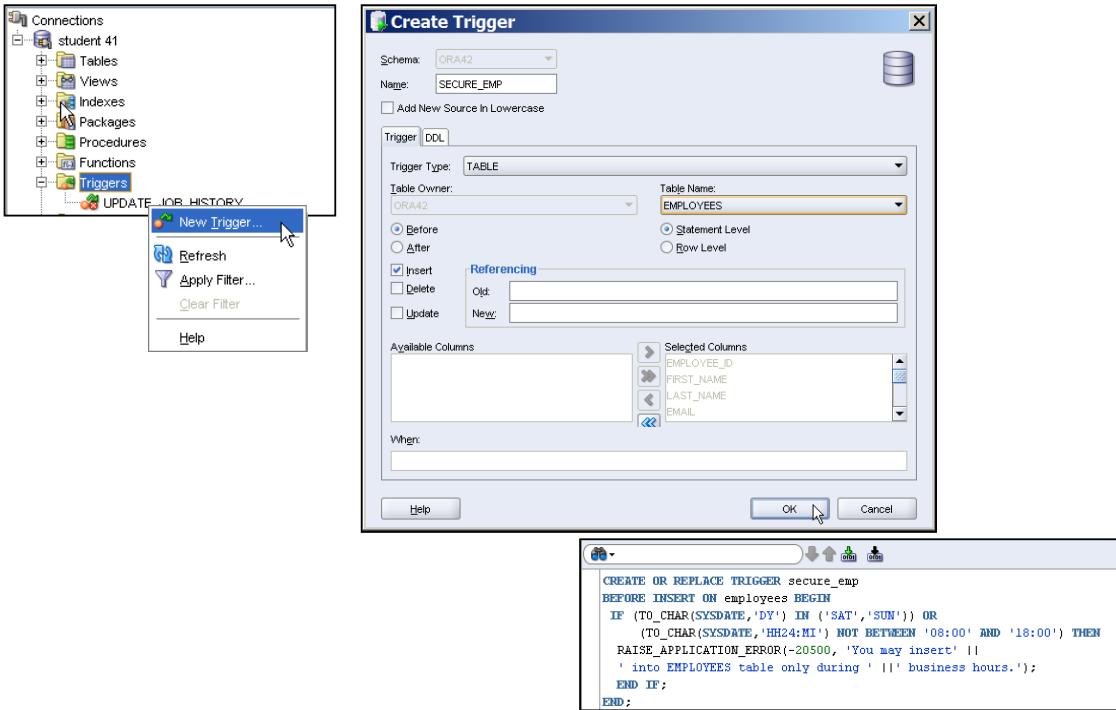
Un disparador de sentencia se arranca una vez en nombre del evento disparador, incluso aunque no haya filas afectadas en absoluto. Los disparadores de sentencia son útiles si la acción del disparador no depende de los datos de las filas afectadas o de los datos proporcionados por el evento disparador 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.

Creación de Disparadores de DML mediante SQL Developer

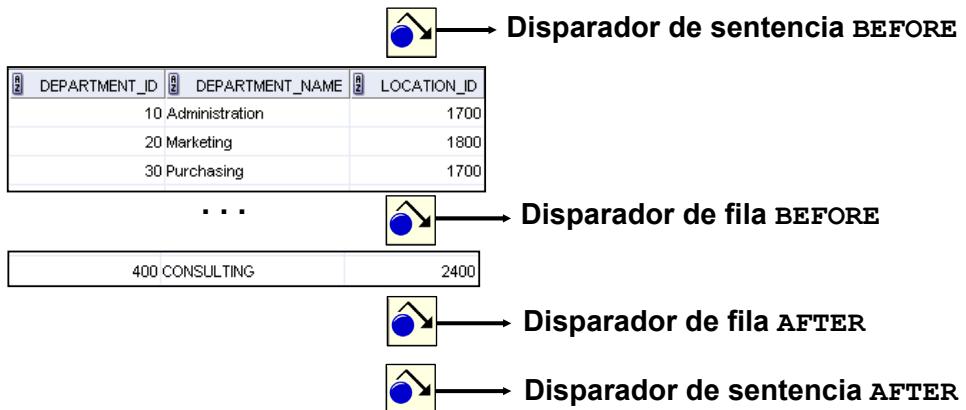


Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Secuencia de Arranque de Disparadores: Manipulación de una Sola Fila

Utilice la siguiente secuencia de arranque para un disparador en una tabla cuando se manipule una única fila:

```
INSERT INTO departments
  (department_id, department_name, location_id)
VALUES (400, 'CONSULTING', 2400);
```



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Secuencia de Arranque de Disparadores: Manipulación de Una Sola Fila

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.

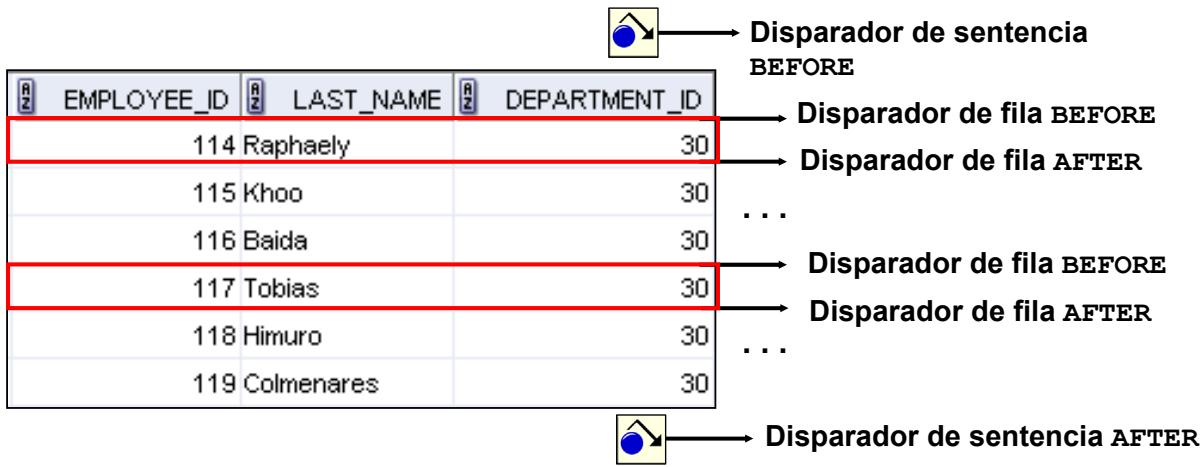
Ejemplo

La sentencia SQL de la diapositiva 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: Manipulación de Varias Filas

Utilice 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;
```



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

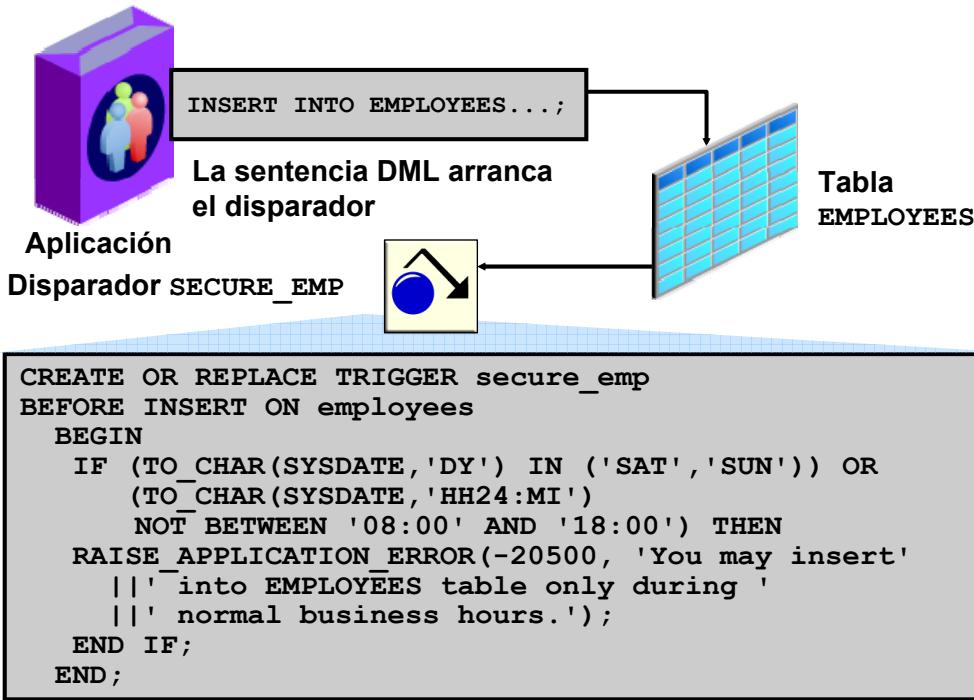
Secuencia de Arranque de Disparadores: Manipulación de Varias Filas

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 diapositiva 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).

Ejemplo de Creación de un Disparador de Sentencia DML: SECURE_EMP



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de un Disparador de Sentencia DML

En el ejemplo de la diapositiva, el disparador de base de datos `SECURE_EMP` es un disparador de sentencia `BEFORE` que impide que se realice 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 del Disparador 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);
```

The screenshot shows the Oracle SQL Developer interface. At the top, there is a toolbar with several buttons: Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. Below the toolbar is a menu bar with options like File, Edit, View, Tools, Database, Help, and a user icon. The main area contains the following text:

```
Error starting at line 1 in command:  
INSERT INTO employees (employee_id, last_name, first_name, email, hire_date,  
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE, 'IT_PROG', 4500, 60)  
Error report:  
SQL Error: ORA-20500: You may insert into EMPLOYEES table only during business hours.  
ORA-06512: at "ORA42.SECURE_EMP", line 4  
ORA-04088: error during execution of trigger 'ORA42.SECURE_EMP'
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Prueba de SECURE_EMP

Para probar el disparador, inserte una fila en la tabla EMPLOYEES durante horas no laborables. Cuando la fecha y la hora están fuera de las horas laborales especificadas en el disparador, recibe el mensaje de error que se muestra en la diapositiva.

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 normal business hours.');
    ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(
      -20500,'You may insert into EMPLOYEES table'|||
      'only during normal business hours.');
    ELSIF UPDATING ('SALARY') THEN
      RAISE_APPLICATION_ERROR(-20503, 'You may '|||
      'update SALARY only normal during business hours.');
    ELSE RAISE_APPLICATION_ERROR(-20504,'You may'|||
      ' update EMPLOYEES table only during'|||
      ' normal business hours.');
    END IF;
  END IF;
END ;

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Detección de la Operación DML que Arrancó un Disparador

Si más de un tipo de operación DML puede arrancar un disparador (por ejemplo, ON INSERT OR DELETE OR UPDATE OF Emp_tab), el cuerpo del disparador puede utilizar los predicados condicionales INSERTING, DELETING y UPDATING para comprobar qué tipo de sentencia arrancó el disparador.

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 8 a.m. a 6 p.m., 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 ;
```

```
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell';
```

```
Error starting at line 1 in command:
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell'
Error report:
SQL Error: ORA-20202: Employee cannot earn more than $15,000.
ORA-06512: at "ORA62.RESTRICT_SALARY", line 4
ORA-04088: error during execution of trigger 'ORA62.RESTRICT_SALARY'
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 primer ejemplo de la diapositiva, se crea un disparador para permitir que sólo los empleados cuyos identificadores de trabajo son AD_PRES o AD_VP ganen un salario de más de 15.000. Si intenta actualizar el salario del empleado Russell cuyo identificador de empleado es SA_MAN, el disparador produce la excepción mostrada en la diapositiva.

Nota: antes de ejecutar el primer ejemplo de código de la diapositiva, asegúrese de desactivar los disparadores secure_emp y secure_employees.

Uso de los Cualificadores OLD y NEW

- Cuando se arranca un disparador de nivel de fila, el motor PL/SQL en tiempo de ejecución crea y rellena dos estructuras de datos:
 - OLD: almacena los valores originales del registro procesado por el disparador
 - NEW: contiene los valores nuevos
- NEW y OLD tienen la misma estructura que un registro declarado mediante %ROWTYPE en la tabla a la que está asociado el disparador.

Operaciones 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

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de los Cualificadores OLD y NEW

En un disparador ROW, puede hacer referencia al valor de una columna antes y después del cambio de los datos agregando como prefijo los cualificadores OLD y NEW.

Nota

- 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.
- Los disparadores de fila pueden disminuir el rendimiento si realiza muchas actualizaciones en tablas más grandes.

Uso de los Cualificadores OLD y NEW: Ejemplo

```

CREATE TABLE audit_emp (
    user_name      VARCHAR2(30),
    time_stamp     date,
    id             NUMBER(6),
    old_last_name VARCHAR2(25),
    new_last_name VARCHAR2(25),
    old_title      VARCHAR2(10),
    new_title      VARCHAR2(10),
    old_salary     NUMBER(8,2),
    new_salary     NUMBER(8,2)
)

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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de los Cualificadores OLD y NEW: Ejemplo

En el ejemplo de la diapositiva, el disparador AUDIT_EMP_VALUES se crea en la tabla EMPLOYEES. El disparador agrega 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.

Uso de los Cualificadores OLD y NEW: Ejemplo

```

INSERT INTO employees (employee_id, last_name, job_id,
salary, email, hire_date)
VALUES (999, 'Temp emp', 'SA_REP', 6000, 'TEMPEMP',
TRUNC(SYSDATE))
/
UPDATE employees
SET salary = 7000, last_name = 'Smith'
WHERE employee_id = 999
/
SELECT *
FROM audit_emp;

```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	USER_NAME	TIME_STAMP	ID	OLD_LAST_NAME	NEW_LAST_NAME	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
1	ORA61	04-JUN-09	(null)	(null)	Temp emp	(null)	SA_REP	(null)	6000
2	ORA61	04-JUN-09	999	Temp emp	Smith	SA_REP	SA_REP	6000	7000

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de los Cualificadores OLD y NEW: Ejemplo del Uso de 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.

A continuación, se muestra el resultado de insertar el registro de empleado en la tabla EMPLOYEES:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
999 (null)	Smith	TEMPEMP	(null)	04-JUN-09	SA_REP	7000	(null)	(null)	(null)	(null)
300 Rob	Smith	RSMITH	(null)	04-JUN-09	IT_PROG	4500	(null)	(null)	(null)	60
206 William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300	(null)	205	110	

...

A continuación, se muestra el resultado de actualizar el salario del empleado “Smith”:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
999 (null)	Smith	TEMPEMP	(null)	04-JUN-09	SA_REP	7000	(null)	(null)	(null)	(null)

...

Uso de la Cláusula WHEN para Arrancar un Disparador de Fila Basado en una Condición

```

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 ;
/

```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Restricción de un Disparador de Fila: Ejemplo

Opcionalmente, puede incluir una restricción de disparador en la definición de un disparador de fila especificando una expresión SQL booleana en una cláusula WHEN. Si incluye una cláusula WHEN en el disparador, la expresión de la cláusula WHEN se evalúa para cada fila a la que afecta el disparador. Si la expresión se evalúa en TRUE para una fila, el cuerpo del disparador se ejecuta en nombre de dicha fila. Sin embargo, si la expresión se evalúa en FALSE o NOT TRUE para una fila (desconocida, como con valores nulos), el cuerpo del disparador no se ejecuta en nombre de dicha fila. La evaluación de la cláusula WHEN no tiene ningún efecto en la ejecución de la sentencia SQL disparadora (es decir, no se realiza el rollback de la sentencia disparadora si la expresión de una cláusula WHEN se evalúa en FALSE).

Nota: una cláusula WHEN no se puede incluir en la definición de un disparador de sentencia.

En el ejemplo de la diapositiva, se crea 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 pueden 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. Ejecute todos los disparadores BEFORE STATEMENT.
2. Realice un bucle *para cada fila* afectada por la sentencia SQL:
 - a. Ejecute todos los disparadores BEFORE ROW *para dicha fila*.
 - b. Ejecute la sentencia DML y realice la comprobación de restricción de integridad*para dicha fila*.
 - c. Ejecute todos los disparadores AFTER ROW *para dicha fila*.
3. Ejecute todos los disparadores AFTER STATEMENT.



Copyright © 2010, Oracle y/o sus filiales. 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).

Nota: la comprobación de integridad se puede diferir hasta que se realice la operación COMMIT.

Implantación de una Restricción de Integridad con un Disparador After

```
-- Integrity constraint violation error -2991 raised.
UPDATE employees SET department_id = 999
WHERE employee_id = 170;
```

```
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;
/
```

```
-- Successful after trigger is fired
UPDATE employees SET department_id = 999
WHERE employee_id = 170;
```

1 rows updated

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Implantación de una Restricción de Integridad con un Disparador After

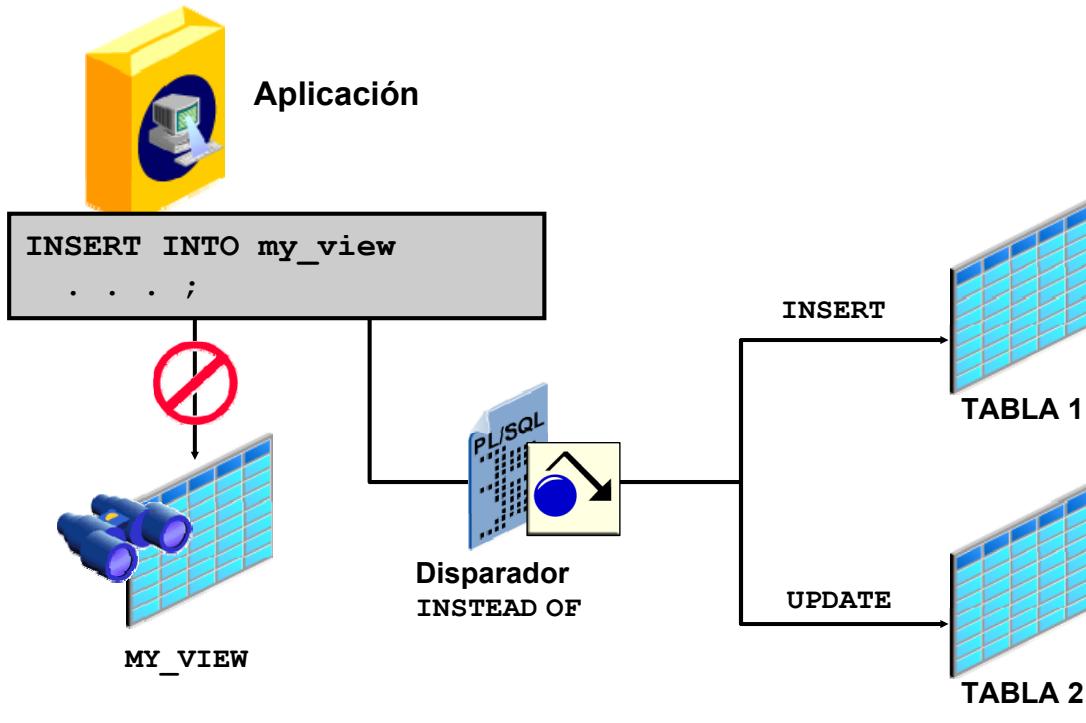
En el ejemplo de la diapositiva se explica una situación en la que se puede ocupar de la restricción de integridad mediante un disparador AFTER. La tabla EMPLOYEES tiene una restricción de clave ajena en la columna DEPARTMENT_ID de la tabla DEPARTMENTS.

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 -2291 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 arranca cuando la sentencia UPDATE modifica el DEPARTMENT_ID del empleado 170 al 999. Cuando se comprueba la restricción de clave ajena, es correcta porque el disparador ha insertado 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: aunque el ejemplo mostrado en la diapositiva se ha logrado de alguna forma debido a los datos limitados del esquema HR, la cuestión es que si difiere la comprobación de restricción hasta la confirmación, tiene la capacidad de crear un disparador para detectar dicho fallo de restricción y repararlo antes de la acción de confirmación.

Disparadores INSTEAD OF



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

ORACLE

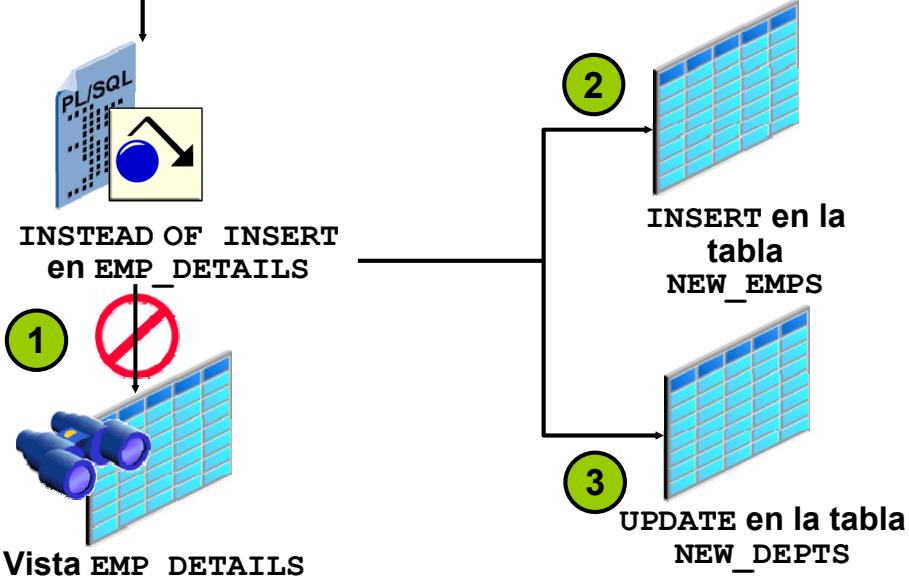
Disparadores INSTEAD OF

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. Estos disparadores se utilizan para realizar operaciones INSERT, UPDATE y DELETE directamente en las tablas subyacentes. Puede escribir sentencias INSERT, UPDATE y DELETE en relación a 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 inherentemente 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: Ejemplo

```
INSERT INTO emp_details
VALUES (9001,'ABBOTT',3000, 10, 'Administration');
```



ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 de la diapositiva 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 arranque el disparador. A continuación, el disparador `INSTEAD OF` emite las operaciones `INSERT` y `UPDATE` adecuadas en 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. 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: antes de ejecutar el ejemplo de esta diapositiva, debe crear las estructuras necesarias mostradas en las dos páginas siguientes.

Creación de un Disparador 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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de un Disparador INSTEAD OF (continuación)

El ejemplo de la diapositiva crea dos nuevas tablas, NEW_EMPS y NEW_DEPTS, que se basan 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. En el ejemplo se 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;
/

```

DEPARTMENT_ID	DEPARTMENT_NAME	DEPT_SAL
10	Administration	7400

1 rows selected

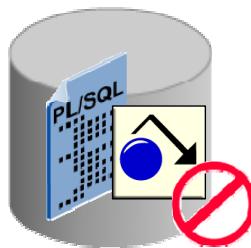
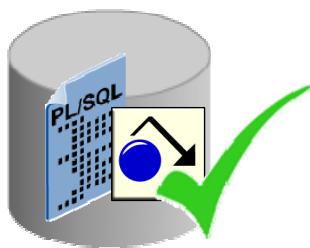
EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
200	Whalen	4400	10
9001	ABBOTT	3000	10

2 rows selected

Estado de un Disparador

Un disparado tiene uno de dos modos distintos:

- Activado: el disparador ejecuta su acción de disparador si se emite una sentencia disparadora y la restricción de disparador (si la hay) se evalúa en true (valor por defecto).
- Desactivado: el disparador no ejecuta su acción de disparador, incluso si se emite una sentencia disparadora y la restricción de disparador (si la hay) se evaluara en true.



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de un Disparador Desactivado

- Antes de Oracle Database 11g, si creaba un disparado cuyo cuerpo tenía un error de compilación PL/SQL, fallaba el DML en la tabla.
- En Oracle Database 11g, puede crear un disparador desactivado y, a continuación, activarlo sólo cuando sepa que se compilará correctamente.

```
CREATE OR REPLACE TRIGGER mytrg
  BEFORE INSERT ON mytable FOR EACH ROW
  DISABLE
BEGIN
  :New.ID := my_seq.Nextval;
  . . .
END ;
/
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de un Disparador Desactivado

Antes de Oracle Database 11g, si creaba un disparado cuyo cuerpo tenía un error de compilación PL/SQL, fallaba el DML en la tabla. Aparecía el siguiente mensaje de error:

ORA-04098: trigger 'TRG' is invalid and failed re-validation

En Oracle Database 11g, puede crear un disparador desactivado y, a continuación, activarlo sólo cuando sepa que se compilará correctamente.

También puede desactivar temporalmente un disparador en las situaciones siguientes:

- Un objeto al que hace referencia no está disponible.
- Tiene que realizar una gran carga de datos y desea que continúe rápidamente sin arrancar disparadores.
- Está volviendo a cargar datos.

Nota: en el ejemplo de código de la diapositiva, se supone que tiene una secuencia existente denominada `my_seq`.

Gestión de Disparadores mediante las Sentencias SQL ALTER y DROP

```
-- Disable or reenable a database trigger:
```

```
ALTER TRIGGER trigger_name DISABLE | ENABLE;
```

```
-- Disable or reenable all triggers for a table:
```

```
ALTER TABLE table_name DISABLE | ENABLE ALL TRIGGERS;
```

```
-- Recompile a trigger for a table:
```

```
ALTER TRIGGER trigger_name COMPILE;
```

```
-- Remove a trigger from the database:
```

```
DROP TRIGGER trigger_name;
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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

Utilice el comando ALTER TRIGGER para desactivar un disparador. También puede desactivar todos los disparadores de una tabla mediante el comando ALTER TABLE. Puede desactivar disparadores para mejorar el rendimiento o evitar comprobaciones de integridad de datos al cargar grandes cantidades de datos con utilidades como, por ejemplo, SQL*Loader. También 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 fuera de línea o un tablespace fuera de línea.

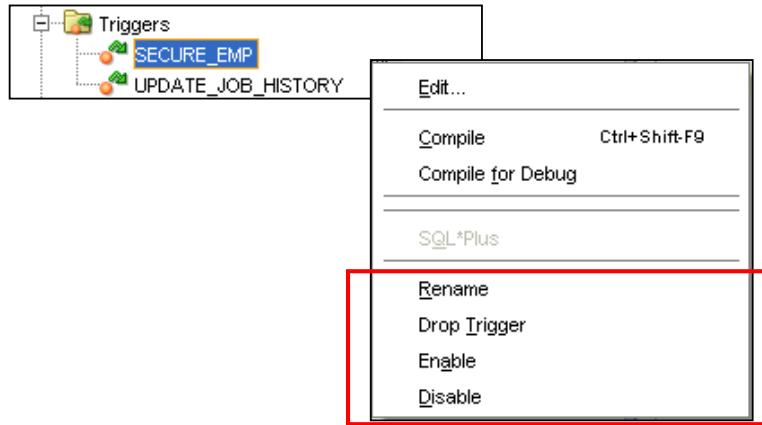
Recompilación de un Disparador

Utilice el comando ALTER TRIGGER para recompilar explícitamente un disparador que no es válido.

Eliminación de Disparadores

Cuando ya no se necesita un disparador, utilice una sentencia SQL en SQL Developer o SQL*Plus para eliminarlo. Al eliminar una tabla, también se eliminan todos los disparadores de dicha tabla.

Gestión de Disparadores mediante SQL Developer



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Gestión de Disparadores mediante SQL Developer

Puede utilizar el nodo Triggers del árbol de navegación Connections para gestionar disparadores. Haga clic con el botón derecho en un nombre de disparador y, a continuación, seleccione una de las siguientes opciones:

- Edit
- Compile
- Compile for Debug
- Rename
- Drop Trigger
- Enable
- Disable

Prueba de Disparadores

- Pruebe cada una de las operaciones de datos disparadoras, así como las no disparadoras.
- Pruebe cada caso de la cláusula WHEN.
- Haga que el disparador se arranque directamente a partir de una operación de datos básica, así como indirectamente a partir de un procedimiento.
- Pruebe el efecto del disparador en otros disparadores.
- Pruebe el efecto de otros disparadores en el disparador.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Prueba de Disparadores

El proceso de prueba del código puede tardar bastante tiempo. Realice lo siguiente 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.

Visualización de Información de Disparador

Puede ver la siguiente información del disparador:

Vista del Diccionario de Datos	Descripción
USER_OBJECTS	Muestra la información del objeto
USER/ALL/DBA_TRIGGER	Muestra la información del disparador
USER_ERRORS	Muestra los errores de sintaxis PL/SQL para un disparador

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización de Información de Disparador

En la diapositiva se muestran 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_TRIGGER 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

DESCRIBE user_triggers

Name	Null	Type
TRIGGER_NAME		VARCHAR2(30)
TRIGGER_TYPE		VARCHAR2(16)
TRIGGERING_EVENT		VARCHAR2(227)
TABLE_OWNER		VARCHAR2(30)
BASE_OBJECT_TYPE		VARCHAR2(16)
TABLE_NAME		VARCHAR2(30)
COLUMN_NAME		VARCHAR2(4000)
REFERENCING_NAMES		VARCHAR2(128)
WHEN_CLAUSE		VARCHAR2(4000)
STATUS		VARCHAR2(8)
DESCRIPTION		VARCHAR2(4000)
ACTION_TYPE		VARCHAR2(11)
TRIGGER_BODY		LONGC
CROSSEDITON		VARCHAR2(7)
BEFORE_STATEMENT		VARCHAR2(3)
BEFORE_ROW		VARCHAR2(3)
AFTER_ROW		VARCHAR2(3)
AFTER_STATEMENT		VARCHAR2(3)
INSTEAD_OF_ROW		VARCHAR2(3)
FIRE_ONCE		VARCHAR2(3)
APPLY_SERVER_ONLY		VARCHAR2(3)

21 rows selected

```
SELECT trigger_type, trigger_body
FROM user_triggers
WHERE trigger_name = 'SECURE_EMP' ;
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de USER_TRIGGERS

Si el archivo de origen no está disponible, puede utilizar SQL Worksheet en SQL Developer o SQL*Plus para regenerarlo desde USER_TRIGGERS. También puede examinar las vistas ALL_TRIGGERS y DBA_TRIGGERS, que contienen la columna adicional OWNER para el propietario del objeto. El resultado del segundo ejemplo de la diapositiva es el siguiente:

TRIGGER_TYPE	TRIGGER_BODY
BEFORE STATEMENT BEGIN	<pre>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 normal business hours.'); ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(-20500, 'You may insert into EMPLOYEES table only during normal business hours.'); ELSIF UPDATING('SALARY') THEN RAISE_APPLICATION_ERROR(-20503, 'You may update SALARY only during normal business hours.'); ELSE RAISE_APPLICATION_ERROR(-20504, 'You may update EMPLOYEES table only during normal business hours.'); END IF; END IF; END;</pre>

Prueba

Un evento disparador puede ser uno o más de los siguientes:

1. Una sentencia INSERT, UPDATE o DELETE en una tabla concreta (o vista, en algunos casos)
2. Una sentencia CREATE, ALTER o DROP en cualquier objeto de esquema
3. Un inicio de la base de datos o cierre de la instancia
4. Un mensaje de error específico o cualquier mensaje de error
5. Una conexión o desconexión del usuario

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: 1, 2, 3, 4, 5

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 © 2010, Oracle y/o sus filiales. 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.

Visión General de la Práctica 8: Creación de Disparadores de Sentencia y de Fila

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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica 8: Visión General

En esta práctica, creará disparadores de sentencia y de fila. También creará procedimientos que se llamarán desde los disparadores.

9

Creación de Disparadores Compuestos, de DDL y de Eventos de Base de Datos

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

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

- Describir disparadores compuestos
- Describir tablas mutantes
- Crear disparadores en sentencias DDL
- Crear disparadores en eventos de sistema
- Mostrar información sobre disparadores



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos de la Lección

En esta lección, aprenderá a crear y utilizar disparadores de base de datos.

¿Qué son los Disparadores Compuestos?

Un disparador único en una tabla que permite especificar acciones para cada uno de los cuatro puntos de temporización siguientes:

- Antes de la sentencia de arranque
- Antes de cada fila a la que afecta la sentencia de arranque
- Despues de cada fila a la que afecta la sentencia de arranque
- Despues de la sentencia de arranque

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

¿Qué son los Disparadores Compuestos?

A partir de Oracle Database 11g, puede utilizar disparadores compuestos. Un disparador compuesto es un disparado único en una tabla que permite especificar acciones para cada uno de los cuatro puntos de temporización disparadores:

- Antes de la sentencia de arranque
- Antes de cada fila a la que afecta la sentencia de arranque
- Despues de cada fila a la que afecta la sentencia de arranque
- Despues de la sentencia de arranque

Nota: para obtener más información sobre los disparadores, consulte *Oracle Database PL/SQL Language Reference 11g Release 2 (11.2) [Referencia del Lenguaje PL/SQL de Oracle Database 11g Versión 2 (11.2)]*.

Trabajar con Disparadores Compuestos

- El cuerpo del disparador compuesto soporta un estado de PL/SQL común al que puede acceder el código para cada punto de temporización.
- El estado común del disparador compuesto:
 - Se establece cuando se inicia la sentencia disparadora
 - Se destruye cuando termina la sentencia disparadora
- Un disparador compuesto tiene una sección de declaraciones y una sección para cada uno de sus puntos de temporización.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Trabajar con Disparadores Compuestos

El cuerpo del disparador compuesto soporta un estado de PL/SQL común al que puede acceder el código para cada punto de temporización. El estado común se destruye automáticamente cuando termina la sentencia disparadora, incluso cuando la sentencia disparadora produce un error. Las aplicaciones pueden evitar el error en la tabla mutante permitiendo que se acumulen filas destinadas a una segunda tabla (como una tabla de historial o una tabla de auditoría) y, a continuación, insertándolas en bloque.

Antes de Oracle Database 11g Versión 1 (11.1), tenía que modelar el estado común con un paquete auxiliar. Este enfoque era laborioso de programar y estaba sujeto a pérdida de memoria cuando la sentencia disparadora provocaba un error y no se arrancaba el disparador de sentencia After. Los disparadores compuestos facilitan el uso de PL/SQL y mejoran el rendimiento y la escalabilidad en tiempo de ejecución.

Ventajas del Uso de un Disparador Compuesto

Puede utilizar disparadores compuestos para:

- Programar un enfoque donde desee que las acciones que implante para los distintos puntos de temporización comparten datos comunes.
- Acumular filas destinadas a una segunda tabla para que pueda insertarlas en bloque periódicamente.
- Evitar el error en la tabla mutante (ORA-04091) permitiendo que se acumulen filas destinadas a una segunda tabla y, a continuación, insertándolas en bloque.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Secciones de Punto de Temporización de un Disparador Compuesto de Tabla

Un disparador compuesto definido en una tabla tiene una o más de las secciones de punto de temporización siguientes. Las secciones de punto de temporización deben aparecer en el orden mostrado en la tabla.

Punto de Temporización	Sección de Disparadores Compuestos
Antes de que se ejecute la sentencia disparadora	Sentencia BEFORE
Después de que se ejecute la sentencia disparadora	Sentencia AFTER
Antes de cada fila a la que afecta la sentencia disparadora	BEFORE EACH ROW
Después de cada fila a la que afecta la sentencia disparadora	AFTER EACH ROW



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Nota

Las secciones de punto de temporización deben aparecer en el orden mostrado en la diapositiva. Si falta una sección de punto de temporización, no sucede nada en su punto de temporización.

Estructura de los Disparadores Compuestos para Tablas

```
CREATE OR REPLACE TRIGGER schema.trigger
FOR dml_event_clause ON schema.table
COMPOUND TRIGGER
```

-- Initial section
-- Declarations
-- Subprograms

1

-- Optional section
BEFORE STATEMENT IS ...;

-- Optional section
AFTER STATEMENT IS ...;

-- Optional section
BEFORE EACH ROW IS ...;

-- Optional section
AFTER EACH ROW IS ...;

2

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Estructura de los Disparadores Compuestos para Tablas

Un disparador compuesto tiene dos secciones principales:

- Una sección inicial en la que se declaran las variables y los subprogramas. El código de esta sección se ejecuta antes que cualquier código de la sección opcional.
- Una sección opcional que define el código para cada posible punto de disparador. En función de si está definiendo un disparador compuesto para una tabla o para una vista, estos puntos disparadores son distintos y se muestran en la imagen que aparece anteriormente y en la página siguiente. El código para los puntos disparadores debe seguir el orden mostrado anteriormente.

Nota: para obtener más información sobre los disparadores compuestos, consulte *Oracle Database PL/SQL Language Reference 11g Release 2 (11.2)* [Referencia del Lenguaje PL/SQL de Oracle Database 11g Versión 2 (11.2)].

Estructura de los Disparadores Compuestos para Vistas

```
CREATE OR REPLACE TRIGGER
schema.trigger

FOR dml_event_clause ON schema.view
COMPOUND TRIGGER

-- Initial section
-- Declarations
-- Subprograms

-- Optional section (exclusive)
INSTEAD OF EACH ROW IS
...;
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Estructura de los Disparadores Compuestos para Vistas

Con las vistas, la única sección permitida es una cláusula INSTEAD OF EACH ROW.

Restricciones de Disparadores Compuestos

- Un disparador compuesto debe ser un disparador DML y estar definido en una tabla o una vista.
- El cuerpo de un disparador compuesto debe ser un bloque de disparador compuesto, escrito en PL/SQL.
- Un cuerpo de disparador compuesto no puede tener un bloque de inicialización; por lo tanto, no puede tener ninguna sección de excepciones.
- Una excepción que se produzca en una sección se debe manejar en esa sección. No se puede transferir el control a otra sección.
- :OLD y :NEW no pueden aparecer en la sección de declaraciones, BEFORE STATEMENT o AFTER STATEMENT.
- Sólo la sección BEFORE EACH ROW puede cambiar el valor de :NEW.
- El orden de arranque de los disparadores compuestos no está garantizado a menos que utilice la cláusula `FOLLOW`.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Restricciones de Disparadores Compuestos

Las siguientes son algunas de las restricciones al trabajar con disparadores compuestos:

- El cuerpo de un disparador compuesto debe ser un bloque de disparador compuesto, escrito en PL/SQL.
- Un disparador compuesto debe ser un disparador DML.
- Un disparador compuesto debe estar definido en una tabla o una vista.
- Un cuerpo de disparador compuesto no puede tener un bloque de inicialización; por lo tanto, no puede tener ninguna sección de excepciones. Esto no es un problema, porque la sección BEFORE STATEMENT siempre se ejecuta exactamente una vez antes de que se ejecute cualquier otra sección de punto de temporización.
- Una excepción que se produzca en una sección se debe manejar en esa sección. No se puede transferir el control a otra sección.
- :OLD, :NEW y :PARENT no pueden aparecer en la sección de declaraciones, la sección BEFORE STATEMENT o la sección AFTER STATEMENT.
- El orden de arranque de los disparadores compuestos no está garantizado a menos que utilice la cláusula `FOLLOW`.

Restricciones de Disparadores en Tablas Mutantes

- Una tabla mutante es:
 - Una tabla modificada con una sentencia UPDATE, DELETE o INSERT, o bien
 - Una tabla que se puede actualizar mediante los efectos de una restricción DELETE CASCADE
- La sección que emitió la sentencia disparadora no puede consultar ni modificar una tabla mutante.
- Esta restricción impide que un disparador vea un juego de datos inconsistente.
- Esta restricción se aplica a todos los disparadores que utilizan la cláusula FOR EACH ROW.
- Las vistas modificadas en los disparadores INSTEAD OF no se consideran mutantes.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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

Una tabla mutante es una tabla que está siendo modificada actualmente por una sentencia UPDATE, DELETE o INSERT o 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.

Se produce un error en la tabla mutante (ORA-4091) cuando un disparador de nivel de fila intenta cambiar o examinar una tabla que ya está realizando un cambio mediante una sentencia DML.

La tabla disparada es una tabla mutante, así como cualquier tabla a la que se haga referencia con la restricción FOREIGN KEY. Esta restricción impide 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
  v_minsalary employees.salary%TYPE;
  v_maxsalary employees.salary%TYPE;
BEGIN
  SELECT MIN(salary), MAX(salary)
    INTO v_minsalary, v_maxsalary
   FROM employees
  WHERE job_id = :NEW.job_id;
  IF :NEW.salary < v_minsalary OR :NEW.salary > v_maxsalary THEN
    RAISE_APPLICATION_ERROR(-20505,'Out of range');
  END IF;
END;
/
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Tabla Mutante: Ejemplo

El disparador CHECK_SALARY del ejemplo de la diapositiva 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';
```

```
TRIGGER check_salary Compiled.

Error starting at line 1 in command:
UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles'
Error report:
SQL Error: ORA-04091: table ORA42.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA42.CHECK_SALARY", line 5
ORA-04088: error during execution of trigger 'ORA42.CHECK_SALARY'
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
*Cause:    A trigger (or a user defined plsql function that is referenced in
          this statement) attempted to look at (or modify) a table that was
          in the middle of being modified by the statement which fired it.
*Action:   Rewrite the trigger (or function) so it does not read that table.
```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Tabla Mutante: Ejemplo (continuación)

En el ejemplo de la diapositiva, 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:

- Utilizar un disparador compuesto como se describe anteriormente en esta lección.
- 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 implantación de las reglas en la aplicación o nivel medio y evite el uso de los disparadores de base de datos para ejecutar reglas de negocio demasiado complejas. Una sentencia de inserción en el ejemplo de código de la diapositiva no generará un ejemplo de tabla mutante.

Uso de un Disparador Compuesto para Resolver el Error en la Tabla Mutante

```

CREATE OR REPLACE TRIGGER check_salary
  FOR INSERT OR UPDATE OF salary, job_id
  ON employees
  WHEN (NEW.job_id <> 'AD_PRES')
  COMPOUND TRIGGER

    TYPE salaries_t          IS TABLE OF employees.salary%TYPE;
    min_salaries             salaries_t;
    max_salaries              salaries_t;

    TYPE department_ids_t     IS TABLE OF employees.department_id%TYPE;
    department_ids            department_ids_t;

    TYPE department_salaries_t IS TABLE OF employees.salary%TYPE
      INDEX BY VARCHAR2(80);
    department_min_salaries   department_salaries_t;
    department_max_salaries   department_salaries_t;

  -- example continues on next slide

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de un Disparador Compuesto para Resolver el Error en la Tabla Mutante

El disparador compuesto CHECK_SALARY resuelve el error en la tabla mutante del ejemplo anterior. Esto se consigue almacenando los valores en recopilaciones de PL/SQL y, a continuación, realizando una inserción/actualización en bloque en la sección “before statement” del disparador compuesto. *Se trata de código de sólo 11g*. En el ejemplo de la diapositiva, se utilizan recopilaciones de PL/SQL. Los tipos de elementos utilizados se basan en las columnas SALARY y DEPARTMENT_ID de la tabla EMPLOYEES. Para crear recopilaciones, defina un tipo de recopilación y, a continuación, declare variables de ese tipo. Las recopilaciones se instancian al introducir un bloque o subprograma y dejar de existir cuando sale. *min_salaries* se utiliza para contener el salario mínimo de cada departamento y *max_salaries* se utiliza para contener el salario máximo de cada departamento.

department_ids se utiliza para contener los identificadores de departamento. Si el empleado que gana el salario mínimo o máximo no tiene un departamento asignado, utilice la función NVL para almacenar -1 para el identificador de departamento en lugar de NULL. A continuación, recopile el salario mínimo, el salario máximo y el identificador de departamento mediante una inserción en bloque en *min_salaries*, *max_salaries* y *department_ids* respectivamente agrupados por identificador de departamento. La sentencia de selección devuelve 13 filas. Los valores de *department_ids* se utilizan como índice para las tablas *department_min_salaries* y *department_max_salaries*. Por lo tanto, el índice de esas dos tablas (VARCHAR2) representa los valores de *department_ids* reales.

Uso de un Disparador Compuesto para Resolver el Error en la Tabla Mutante

```

. . .
BEFORE STATEMENT IS
BEGIN
    SELECT MIN(salary), MAX(salary), NVL(department_id, -1)
    BULK COLLECT INTO min_Salaries, max_salaries, department_ids
    FROM employees
    GROUP BY department_id;
    FOR j IN 1..department_ids.COUNT() LOOP
        department_min_salaries(department_ids(j)) := min_salaries(j);
        department_max_salaries(department_ids(j)) := max_salaries(j);
    END LOOP;
END BEFORE STATEMENT;

AFTER EACH ROW IS
BEGIN
    IF :NEW.salary < department_min_salaries(:NEW.department_id)
    OR :NEW.salary > department_max_salaries(:NEW.department_id) THEN
        RAISE_APPLICATION_ERROR(-20505,'New Salary is out of acceptable
                                         range');
    END IF;
END AFTER EACH ROW;
END check_salary;

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de un Disparador Compuesto para Resolver el Error en la Tabla Mutante (continuación)

Después de agregar cada fila, si el nuevo salario es menor que el salario mínimo para ese departamento o mayor que el salario máximo del departamento, se muestra un mensaje de error.

Para probar el disparador compuesto recién creado, emita la siguiente sentencia:

```

UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles';

```

1 rows updated

Para asegurarse de que el salario del empleado Stiles se ha actualizado, emita la siguiente consulta mediante la tecla F9 en SQL Developer:

```

SELECT employee_id, first_name, last_name, job_id, department_id,
       salary
FROM employees
WHERE last_name = 'Stiles';

```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID	DEPARTMENT_ID	SALARY
1	138	Stephen	Stiles	ST_CLERK	50	3400

Creación de Disparadores en Sentencias DDL

```
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE | AFTER -- Timing
[ddl_event1 [OR ddl_event2 OR ...]]
ON {DATABASE | SCHEMA}
trigger_body
```

Eventos DDL de Ejemplo	Arranca cuando
CREATE	Se crea cualquier objeto de base de datos con el comando CREATE.
ALTER	Se modifica cualquier objeto de base de datos con el comando ALTER.
DROP	Se borra cualquier objeto de base de datos con el comando DROP.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de Disparadores en Sentencias DDL

Puede especificar uno o más tipos de sentencias DDL que pueden provocar que se arranque el disparador. Puede crear disparadores para estos eventos en DATABASE o SCHEMA a menos que se indique lo contrario. Especifique también BEFORE y AFTER para la temporización del disparador. La base de datos Oracle arranca el disparador en la transacción del usuario existente.

No puede especificar como evento disparador cualquier operación DDL realizada mediante un procedimiento PL/SQL.

El cuerpo del disparador en la sintaxis de la diapositiva representa un bloque PL/SQL completo.

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 de Eventos 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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de Disparadores de Base de Datos

Antes de codificar el cuerpo del disparador, elija los componentes del disparador.

Los disparadores de los eventos del sistema se pueden definir a nivel de esquema o de base de datos. Por ejemplo, un disparador de cierre de base de datos se define a nivel de base de datos. Los disparadores de las sentencias de lenguaje de definición de datos (DDL) o una conexión o desconexión de usuario, se pueden definir a nivel de base de datos o de esquema. Los disparadores de las sentencias de lenguaje de manipulación de datos (DML) se definen en una vista o tabla determinada.

Un disparador definido a nivel de base de datos se arranca para todos los usuarios mientras que el que se define a nivel de tabla o de esquema sólo se arranca cuando el evento disparador incluye dicho esquema o tabla.

A continuación, se indican los eventos disparadores que pueden hacer que un disparador se arranque:

- Una sentencia de definición de datos en un objeto de la base de datos o del esquema
- La conexión o desconexión de un usuario determinado (o de cualquier usuario)
- El cierre o inicio de una base de datos
- Cualquier error que se produzca

Creación de Disparadores en Eventos de Sistema

```
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE | AFTER -- timing
[database_event1 [OR database_event2 OR ...]]
ON {DATABASE | SCHEMA}
trigger_body
```

Evento de Base de Datos	El Disparador se Arranca cuando
AFTER SERVERERROR	Se produce un error de Oracle
AFTER LOGON	Un usuario se conecta a la base de datos
BEFORE LOGOFF	Un usuario se desconecta de la base de datos
AFTER STARTUP	Se abre la base de datos
BEFORE SHUTDOWN	Se cierra la base de datos de forma normal



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de Disparadores en Eventos de Sistema

Puede crear disparadores para los eventos mostrados en la tabla de la diapositiva en DATABASE o SCHEMA, excepto SHUTDOWN y STARTUP, que sólo se aplican a DATABASE.

Disparadores LOGON y LOGOFF: Ejemplo

```
-- Create the log_trig_table shown in the notes page
-- first

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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Disparadores LOGON y LOGOFF: Ejemplo

Puede crear estos disparadores para supervisar la frecuencia de conexión y desconexión, o bien puede que desee escribir un informe que supervise 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.

La definición de log_trig_table utilizado en los ejemplos de la diapositiva es la siguiente:

```
CREATE TABLE log_trig_table(
    user_id  VARCHAR2(30),
    log_date DATE,
    action   VARCHAR2(40))
/
```

Sentencias CALL en Disparadores

```
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 PROCEDURE log_execution IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('log_execution: Employee Inserted');
END;
/
CREATE OR REPLACE TRIGGER log_employee
BEFORE INSERT ON EMPLOYEES
CALL log_execution -- no semicolon needed
/
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Sentencias CALL en Disparadores

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 implantar 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 OR REPLACE 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.

Ventajas de los Disparadores de Eventos 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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Ventajas de los Disparadores de Eventos 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

Privilegios del Sistema Necesarios para Gestionar Disparadores

Los siguientes privilegios del sistema son necesarios para gestionar los disparadores:

- Privilegios que le permiten crear, modificar y borrar disparadores en cualquier esquema:
 - GRANT CREATE TRIGGER TO ora61
 - GRANT ALTER ANY TRIGGER TO ora61
 - GRANT DROP ANY TRIGGER TO ora61
- Privilegio que le permite crear un disparador en la base de datos:
 - GRANT ADMINISTER DATABASE TRIGGER TO ora61
- Privilegio EXECUTE (si el disparador hace referencia a objetos que no están en el esquema)



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Privilegios del Sistema Necesarios para Gestionar 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.

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.

Nota: 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.

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 © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Instrucciones para el Diseño de Disparadores

- Utilice disparadores para garantizar que las acciones relacionadas se realizan para una operación concreta y 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, implante reglas de integridad mediante restricciones declarativas en lugar de 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 JavaBeans (EJB) en el nivel medio.
 - 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.
- 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.

Prueba

1. Un disparador se define con una sentencia CREATE TRIGGER.
2. El código fuente de un disparador se incluye en el diccionario de datos USER_TRIGGERS.
3. Un disparador se llama de forma explícita.
4. DML llama a un disparador de forma implícita.
5. No se permiten las sentencias COMMIT, SAVEPOINT y ROLLBACK al trabajar con un disparador.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: 1, 2, 4, 5

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Describir disparadores compuestos
- Describir tablas mutantes
- Crear disparadores en sentencias DDL
- Crear disparadores en eventos de sistema
- Mostrar información sobre disparadores



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica 9: 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



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica 9: Visión General

En esta práctica, implantará 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 sección práctica de la lección anterior. El efecto en cascada originará una excepción de tabla mutante en la tabla JOBS. A continuación, cree un paquete PL/SQL y disparadores adicionales para resolver el problema de la tabla mutante.

10

Uso del Compilador PL/SQL

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

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

- Utilizar los parámetros de inicialización del compilador PL/SQL
- Utilizar las advertencias de tiempo de compilación de PL/SQL

Tenga en cuenta que el contenido de este capítulo es específico de 11g. Algunos de los ejemplos mostrados pueden funcionar o no en un entorno de 10g.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Agenda de la Lección

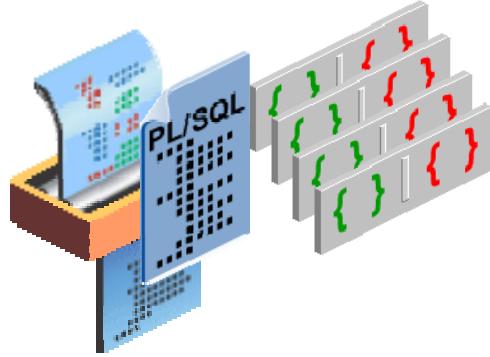
- Uso de los parámetros de inicialización `PLSQL_CODE_TYPE` y `PLSQL_OPTIMIZE_LEVEL` para la compilación PL/SQL
- Uso de las advertencias de tiempo de compilación de PL/SQL:
 - Uso del parámetro de inicialización `PLSQL_WARNING`
 - Uso de los subprogramas del paquete `DBMS_WARNING`

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Parámetros de Inicialización para la Compilación PL/SQL

- PLSQL_CODE_TYPE
- PLSQL_OPTIMIZE_LEVEL
- PLSQL_CCFLAGS
- PLSQL_WARNINGS



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Parámetros de Inicialización para la Compilación PL/SQL

En las versiones anteriores a Oracle Database 10g, el compilador PL/SQL traducía el código a código de máquina sin aplicar muchos cambios por cuestiones de rendimiento. Desde Oracle Database 11g en adelante, PL/SQL utiliza un compilador de optimización que puede reorganizar el código para obtener un mejor rendimiento. No tiene que hacer nada para aprovechar las ventajas de este nuevo optimizador; está activado por defecto.

Nota

- El parámetro de inicialización PLSQL_CCFLAGS se trata en la lección titulada “Gestión del Código PL/SQL”.
- El parámetro de inicialización PLSQL_WARNINGS se trata más adelante en esta lección.

Uso de los Parámetros de Inicialización para la Compilación PL/SQL

- **PLSQL_CODE_TYPE**: especifica el modo de compilación para unidades de biblioteca PL/SQL

```
PLSQL_CODE_TYPE = { INTERPRETED | NATIVE }
```

- **PLSQL_OPTIMIZE_LEVEL**: especifica el nivel de optimización que se va a utilizar para compilar unidades de biblioteca PL/SQL

```
PLSQL_OPTIMIZE_LEVEL = { 0 | 1 | 2 | 3 }
```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de los Parámetros de Inicialización para la Compilación PL/SQL

Parámetro **PLSQL_CODE_TYPE**

Este parámetro especifica el modo de compilación para unidades de biblioteca PL/SQL. Si selecciona INTERPRETED, las unidades de biblioteca PL/SQL se compilarán con formato de código de byte PL/SQL. El motor de intérprete de PL/SQL ejecuta dichos módulos. Si selecciona NATIVE, las unidades de biblioteca PL/SQL (con la posible excepción de bloques PL/SQL anónimos de nivel superior) se compilarán en código nativo (de máquina). Dichos módulos se ejecutarán de forma nativa sin incurrir en ninguna sobrecarga de intérprete. Cuando se cambia el valor de este parámetro, no tiene ningún efecto en las unidades de biblioteca PL/SQL que ya se han compilado. El valor de este parámetro se almacena de forma persistente con cada unidad de biblioteca. Si una unidad de biblioteca PL/SQL se compila de forma nativa, todas las recompilaciones automáticas posteriores de esa unidad de biblioteca utilizarán la compilación nativa. En Oracle Database 11g, la compilación nativa es más fácil y está más integrada, con pocos parámetros de inicialización que definir.

En casos excepcionales, si la sobrecarga del optimizador que realiza la compilación de aplicaciones muy grandes tarda mucho, puede reducir el nivel de optimización definiendo el parámetro de inicialización **PLSQL_OPTIMIZE_LEVEL** en 1 en lugar de su valor por defecto 2. En casos más excepcionales, puede ver un cambio en el comportamiento de la excepción: una excepción que no se emite en absoluto o una que se emite antes de lo esperado. La definición de **PLSQL_OPTIMIZE_LEVEL** en 0 impide que el código se reorganice.

Uso de los Parámetros de Inicialización para la Compilación PL/SQL (continuación)

Parámetro `PLSQL_OPTIMIZE_LEVEL`

Este parámetro especifica el nivel de optimización que se utilizará para compilar unidades de biblioteca PL/SQL. Cuanto mayor sea el valor de este parámetro, mayor esfuerzo realizará el compilador para optimizar las unidades de biblioteca PL/SQL. Los valores disponibles son (0, 1 y 2 estaban disponibles a partir de Oracle 10g Versión 2):

0: mantiene el orden de evaluación y, por lo tanto, el patrón de efectos secundarios, excepciones e inicializaciones de paquetes de Oracle9i y versiones anteriores. También elimina la nueva identidad semántica de `BINARY_INTEGER` y `PLS_INTEGER`, además de restaurar las reglas anteriores para la evaluación de expresiones de enteros. Aunque el código se ejecutará algo más rápido que en Oracle9i, el uso del nivel 0 revocará la mayor parte de las mejoras de rendimiento de PL/SQL a partir de Oracle Database 10g.

1: aplica un amplio rango de optimizaciones a programas PL/SQL, que incluyen la eliminación de cálculos y excepciones innecesarias, pero generalmente no mueve código fuente fuera de su orden de origen original.

2: aplica un amplio rango de técnicas de optimización modernas más allá de las del nivel 1, que incluyen cambios que pueden mover código fuente relativamente lejos de su ubicación original.

3: este valor es nuevo en Oracle Database 11g. Aplica un amplio rango de técnicas de optimización más allá de las del nivel 2, que incluyen automáticamente técnicas no solicitadas específicamente. Esto permite la función en línea de procedimientos, que es un proceso de optimización que sustituye llamadas a procedimientos por una copia del cuerpo del procedimiento que se va a llamar. El procedimiento copiado casi siempre se ejecuta más rápidamente que la llamada original. Para permitir la función en línea de subprogramas, acepte el valor por defecto del parámetro de inicialización `PLSQL_OPTIMIZE_LEVEL` (que es 2) o defínalos en 3. Con

`PLSQL_OPTIMIZE_LEVEL` = 2, debe especificar cada subprograma que se va a poner en línea. Con `PLSQL_OPTIMIZE_LEVEL` = 3, el compilador PL/SQL busca oportunidades para poner en línea subprogramas más allá de las especificadas.

Nota: para obtener más información sobre la función en línea, consulte *Oracle Database PL/SQL Language Reference 11g Release 2 (11.2)* [Referencia del Lenguaje PL/SQL de Oracle Database 11g Versión 2 (11.2)] y el curso dirigido por un instructor *Oracle Database 11g: PL/SQL Avanzado*.

Normalmente, al definir este parámetro en 2, se compensa en términos de mejor rendimiento de la ejecución. Sin embargo, si el compilador se ejecuta lentamente en un módulo de origen concreto o si la optimización no tiene sentido por algún motivo (por ejemplo, durante un rápido desarrollo de respuesta), la definición de este parámetro en 1 tiene como resultado una compilación casi tan buena como un menor uso de recursos de tiempo de compilación. El valor de este parámetro se almacena de forma persistente con la unidad de biblioteca.

Nota

El parámetro `PLSQL_CODE_TYPE` en Oracle Database 10g sustituía los siguientes parámetros obsoletos:

- `PLSQL_NATIVE_C_COMPILER`
- `PLSQL_NATIVE_MAKE_FILE_NAME`
- `PLSQL_NATIVE_C_COMPILER`
- `PLSQL_NATIVE_MAKE.Utility`
- `PLSQL_NATIVE_LINKER`

El parámetro `PLSQL_DEBUG` ya no se utiliza en Oracle Database 11g. El parámetro `PLSQL_DEBUG` ya no controla la generación de información de depuración mediante el compilador PL/SQL; siempre se genera información de depuración y no se necesita ningún parámetro especial.

Configuración del Compilador

Opción del Compilador	Descripción
<code>PLSQL_CODE_TYPE</code>	Especifica el modo de compilación para unidades de biblioteca PL/SQL.
<code>PLSQL_OPTIMIZE_LEVEL</code>	Especifica el nivel de optimización que se va a utilizar para compilar unidades de biblioteca PL/SQL.
<code>PLSQL_WARNINGS</code>	Activa o desactiva el informe de mensajes de advertencia del compilador PL/SQL.
<code>PLSQL_CCFLAGS</code>	Controla la compilación condicional de cada unidad de biblioteca PL/SQL independientemente.

En general, para un rendimiento más rápido, utilice el siguiente valor:

```
PLSQL_CODE_TYPE = NATIVE
PLSQL_OPTIMIZE_LEVEL = 2
```

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Configuración del Compilador

El nuevo compilador aumenta el rendimiento del código PL/SQL y permite que se ejecute aproximadamente dos veces más rápido que Oracle8i Database y entre 1,5 y 1,75 veces más rápido que Oracle9i Database Versión 2.

Para obtener un rendimiento mayor, el valor del compilador debe ser:

```
PLSQL_CODE_TYPE = NATIVE
PLSQL_OPTIMIZE_LEVEL = 2
```

Visualización de los Parámetros de Inicialización de PL/SQL

Utilice las vistas del diccionario de datos

USER | ALL | DBA _PLSQL _OBJECT _SETTINGS para visualizar la configuración de un objeto PL/SQL:

```
DESCRIBE USER_PLSQL_OBJECT_SETTINGS
```

Name	Null	Type
NAME	NOT NULL	VARCHAR2(30)
TYPE		VARCHAR2(12)
PLSQL_OPTIMIZE_LEVEL		NUMBER
PLSQL_CODE_TYPE		VARCHAR2(4000)
PLSQL_DEBUG		VARCHAR2(4000)
PLSQL_WARNINGS		VARCHAR2(4000)
NLS_LENGTH_SEMANTICS		VARCHAR2(4000)
PLSQL_CCFLAGS		VARCHAR2(4000)
PLSCOPE_SETTINGS		VARCHAR2(4000)
9 rows selected		

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización de los Parámetros de Inicialización de PL/SQL

Las columnas de la vista del diccionario de datos USER_PLSQL_OBJECTS_SETTINGS son:

Owner: propietario del objeto. Esta columna no se muestra en la vista USER_PLSQL_OBJECTS_SETTINGS.

Name: nombre del objeto.

Type: las opciones disponibles son las siguientes: PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER, TYPE o TYPE BODY.

PLSQL_OPTIMIZE_LEVEL: nivel de optimización utilizado para compilar el objeto.

PLSQL_CODE_TYPE: modo de compilación del objeto.

PLSQL_DEBUG: especifica si el objeto se ha compilado para la depuración.

PLSQL_WARNINGS: configuración de advertencias del compilador utilizada para compilar el objeto.

NLS_LENGTH_SEMANTICS: semántica de longitud de NLS utilizada para compilar el objeto.

PLSQL_CCFLAGS: indicador de compilación condicional utilizado para compilar el objeto.

PLSCOPE_SETTINGS: controla la recopilación de tiempo de compilación, las referencias cruzadas y el almacenamiento de datos de identificador de código fuente PL/SQL (nuevo en Oracle Database 11g).

Visualización y Definición de los Parámetros de Inicialización de PL/SQL

```
SELECT name, type, plsql_code_type AS CODE_TYPE,
       plsql_optimize_level AS OPT_LVL
  FROM user_plsql_object_settings;
```

NAME	TYPE	CODE_TYPE	OPT_LVL
1 ADD_EMPLOYEE	PROCEDURE	INTERPRETED	2
2 ADD_JOB_HIST	PROCEDURE	INTERPRETED	2
3 ADD_JOB_HISTORY	PROCEDURE	INTERPRETED	2
4 BANK_TRANS	PROCEDURE	INTERPRETED	2
5 CHECK_SALARY	PROCEDURE	INTERPRETED	2

...

- Defina el valor del parámetro de inicialización del compilador con las sentencias ALTER SYSTEM o ALTER SESSION.
- Se accede a los valores de los parámetros cuando se ejecuta la sentencia CREATE OR REPLACE.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Nota

- Para obtener más información sobre las sentencias ALTER SYSTEM o ALTER SESSION, consulte *Oracle Database SQL Reference (Referencia SQL de Oracle Database)*.
- La familia de vistas del diccionario de datos DBA_STORED_SETTINGS ya no se utiliza en Oracle Database 10g y se sustituye por la familia de vistas del diccionario de datos DBA_PLSQL_OBJECT_SETTINGS.

Cambio de los Parámetros de Inicialización de PL/SQL: Ejemplo

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL = 1;
ALTER SESSION SET PLSQL_CODE_TYPE = 'NATIVE';
```

```
ALTER SESSION SET succeeded.
ALTER SESSION SET succeeded.
```

```
-- code displayed in the notes page
CREATE OR REPLACE PROCEDURE add_job_history
. . .
```

```
@code_10_10_s.sql
```

NAME	TYPE	CODE_TYPE	OPT_LVL
1 ADD_COL	PROCEDURE	INTERPRETED	2
2 ADD_EMPLOYEE	PROCEDURE	INTERPRETED	2
3 ADD_JOB_HIST	PROCEDURE	INTERPRETED	2
4 ADD_JOB_HISTORY	PROCEDURE	NATIVE	1

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Cambio de los Parámetros de Inicialización de PL/SQL: Ejemplo

Para cambiar un objeto PL/SQL compilado del tipo de código interpretado al tipo de código nativo, debe definir primero el parámetro PLSQL_CODE_TYPE en NATIVE (opcionalmente defina los otros parámetros) y, a continuación, recompile el programa. Para forzar la compilación nativa en todo el código PL/SQL, debe recomilar cada uno. Los scripts (en el directorio rdmbs/admin) se proporcionan para que logre la conversión en compilación nativa completa (dbmsupgnv.sql) o la compilación interpretada completa (dbmsupgln.sql). El procedimiento add_job_history se crea de la siguiente forma:

```
CREATE OR REPLACE PROCEDURE add_job_history
( p_emp_id          job_history.employee_id%type
, p_start_date      job_history.start_date%type
, p_end_date        job_history.end_date%type
, p_job_id          job_history.job_id%type
, p_department_id   job_history.department_id%type )
IS
BEGIN
    INSERT INTO job_history (employee_id, start_date,
                           end_date, job_id, department_id)
    VALUES(p_emp_id, p_start_date, p_end_date,
           p_job_id, p_department_id);
END add_job_history;
/
```

Agenda de la Lección

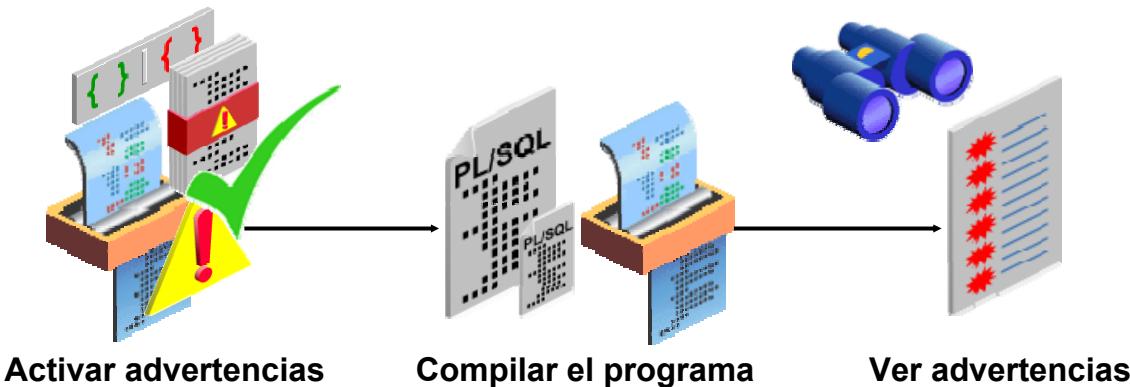
- Uso de los parámetros de inicialización `PLSQL_CODE_TYPE` y `PLSQL_OPTIMIZE_LEVEL` para la compilación PL/SQL
- Uso de las advertencias de tiempo de compilación de PL/SQL:
 - Uso del parámetro de inicialización `PLSQL_WARNINGS`
 - Uso de los subprogramas del paquete `DBMS_WARNING`

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visión General de las Advertencias de Tiempo de Compilación PL/SQL para Subprogramas

A partir de Oracle 10g, el compilador PL/SQL se ha mejorado para producir advertencias para subprogramas.



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visión General de las Advertencias de Tiempo de Compilación PL/SQL para Subprogramas

Para que los programas sean más sólidos y evitar problemas en tiempo de ejecución, puede activar la comprobación de determinadas condiciones de advertencia. Estas condiciones no son lo suficientemente graves para producir un error e impedir que compile un subprograma. Pueden apuntar a algo en el subprograma que produce un resultado indefinido o pueden crear un problema de rendimiento.

En versiones anteriores a Oracle Database 10g, la compilación de un programa PL/SQL tenía dos posibles resultados:

- Correcto, que produce una unidad compilada válida
- Fallo, con errores de compilación que indican que el programa tenía errores de sintaxis o semánticos

Sin embargo, incluso cuando la compilación de un programa era correcta, el programa podía haber violado las prácticas recomendadas o se podía haber codificado para ser más eficaz. Oracle Database 10g introdujo una nueva función fácil de utilizar que permite que el compilador PL/SQL comunique mensajes de advertencia en estas situaciones. Las advertencias del compilador permiten a los desarrolladores evitar problemas de codificación comunes, mejorando así la productividad.

Visión General de las Advertencias de Tiempo de Compilación PL/SQL para Subprogramas (continuación)

PL/SQL soporta la transferencia de los parámetros `IN` `OUT` y `OUT` por valor o por referencia mediante la indicación del compilador `NOCOPY`. La transferencia de parámetros por valor es inherentemente menos eficaz porque implica realizar varias copias de los datos. Con Oracle Database 11g, el compilador detecta y recomienda automáticamente el uso de la indicación `NOCOPY`, donde los tipos de parámetros son grandes tipos de objeto, de registro y de recopilación.

Con la función de advertencia del compilador PL/SQL, la compilación de un programa PL/SQL puede tener posibles resultados adicionales:

- Correcto con advertencias de compilación
- Fallo con errores de compilación y advertencias de compilación

Tenga en cuenta que el compilador puede emitir mensajes de advertencia incluso en una compilación correcta. Se debe corregir un error de compilación para poder utilizar el procedimiento almacenado siempre que haya una advertencia para fines informativos.

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`

Ventajas de las Advertencias del Compilador

- Hacer que los programas sean más robustos y evitar problemas en tiempo de ejecución
- Identificar posibles problemas de rendimiento
- Identificar factores que producen resultados no definidos



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Ventajas de las Advertencias del Compilador

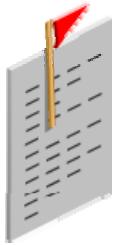
El uso de advertencias del compilador puede ayudar a lo siguiente:

- Hacer que los programas sean más robustos y evitar problemas en tiempo de ejecución
- Identificar posibles problemas de rendimiento
- Identificar 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.
- Los mensajes de advertencia se pueden emitir durante la compilación de subprogramas PL/SQL; los bloques anónimos no producen ninguna advertencia.
- Todos los mensajes de advertencia PL/SQL utilizan el prefijo PLW.

Categorías de Mensajes de Advertencia de Tiempo de Compilación PL/SQL



SEVERE



PERFORMANCE



INFORMATIONAL



ALL

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Categorías de Mensajes de Advertencia de Tiempo de Compilación PL/SQL

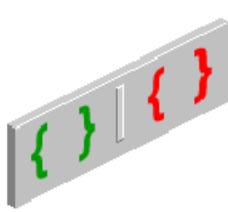
Los mensajes de advertencia PL/SQL se dividen en categorías, por lo que puede suprimir o mostrar grupos de advertencias similares durante la compilación. Las categorías son:

- SEVERE: mensajes para condiciones que pueden provocar un comportamiento inesperado o resultados incorrectos, como problemas de alias con parámetros
- PERFORMANCE: mensajes para condiciones que pueden provocar problemas de rendimiento, como la transferencia de un valor VARCHAR2 a una columna NUMBER en una sentencia INSERT
- INFORMATIONAL: mensajes para condiciones que no tienen un efecto en el rendimiento o corrección, pero que puede desear cambiar para que el código se pueda mantener mejor, como código inaccesible que nunca se puede ejecutar

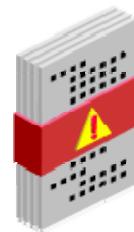
Definición de Niveles de Mensajes de Advertencia

Puede definir niveles de advertencia con uno de los métodos siguientes:

- De forma declarativa:
 - Mediante el parámetro de inicialización `PLSQL_WARNINGS`
- Mediante programación:
 - Con el paquete `DBMS_WARNING`



**Parámetro de
inicialización
`PLSQL_WARNINGS`**



**Paquete
`DBMS_WARNING`**

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Definición de Niveles de Mensajes de Advertencia

Puede definir niveles de mensajes de advertencia del compilador con uno de los métodos siguientes:

Uso del Parámetro de Inicialización `PLSQL_WARNINGS`

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. La configuración para el parámetro `PLSQL_WARNINGS` se almacena junto con cada subprograma compilado. Puede utilizar el parámetro de inicialización `PLSQL_WARNINGS` para hacer lo siguiente:

- Activar o desactivar la creación de informes de todas las advertencias, las advertencias de una categoría seleccionada o los mensajes de advertencia concretos.
- Tratar todas las advertencias, una categoría seleccionada de advertencia o mensajes de advertencia concretos como errores.
- Cualquier combinación válida de los anteriores.

La palabra clave `All` es una forma abreviada de hacer referencia a todos los mensajes de advertencia: `SEVERE`, `PERFORMANCE` y `INFORMATIONAL`.

Uso del 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 la configuración actual del sistema o sesión. Este paquete se trata más adelante en esta lección.

Definición de Niveles de Advertencia del Compilador: mediante PLSQL_WARNINGS

```
ALTER [SESSION|SYSTEM]
PLSQL_WARNINGS = 'value_clause1'[, 'value_clause2']...
```

```
value_clause = Qualifier Value : Modifier Value
```

```
Qualifier Value = { ENABLE | DISABLE | ERROR }

Modifier Value =
{ ALL | SEVERE | INFORMATIONAL | PERFORMANCE |
{ integer | (integer [, integer] ...) } }
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Modificación de la Configuración de Advertencia del Compilador

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, DISABLE y ERROR. 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.

Posibles valores para ENABLE, DISABLE y ERROR:

- ALL
- SEVERE
- INFORMATIONAL
- PERFORMANCE
- numeric_value

Los valores para numeric_value están dentro de lo siguiente:

- Rango 5000-5999 para grave
- Rango 6000-6249 para informativo
- Rango 7000-7249 para rendimiento

Definición de Niveles de Advertencia del Compilador: mediante PLSQL_WARNINGS, Ejemplos

```
ALTER SESSION
SET plsql_warnings = 'enable:severe',
'enable:performance',
'disable:informational';
```

ALTER SESSION succeeded.

```
ALTER SESSION
SET plsql_warnings = 'enable:severe';
```

ALTER SESSION succeeded.

```
ALTER SESSION SET PLSQL_WARNINGS='ENABLE:SEVERE',
'DISABLE:PERFORMANCE' , 'ERROR:05003';
```

ALTER SESSION SET succeeded.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Definición de Niveles de Advertencia del Compilador: mediante PLSQL_WARNINGS, Ejemplos

Puede utilizar el comando ALTER SESSION o ALTER SYSTEM para cambiar el parámetro de inicialización PLSQL_WARNINGS. El gráfico de la diapositiva muestra los distintos ejemplos de activar y desactivar advertencias del compilador.

Ejemplo 1

En este ejemplo, activa advertencias SEVERE y PERFORMANCE y desactiva advertencias INFORMATIONAL.

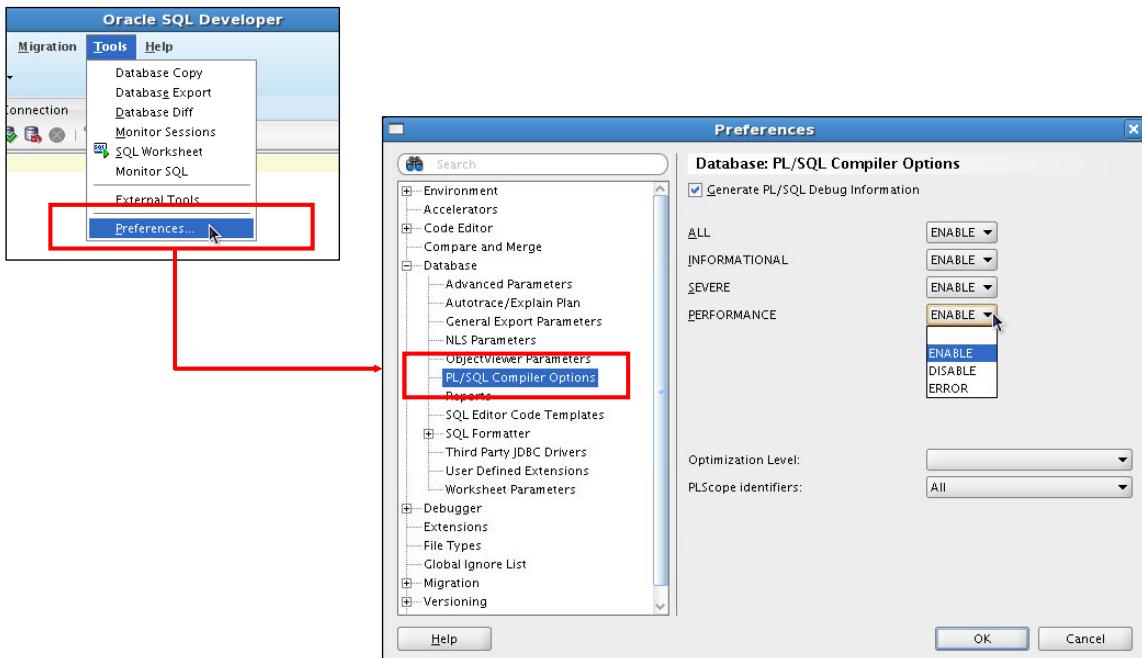
Ejemplo 2

En el segundo ejemplo, activa sólo las advertencias SEVERE.

Ejemplo 3

También puede tratar mensajes concretos como errores en lugar de advertencias. En este ejemplo, si sabe que el mensaje de advertencia PLW-05003 representa un problema grave en el código, la inclusión de 'ERROR:05003' en el valor PLSQL_WARNINGS hace que la condición dispare un mensaje de error (PLS_05003) en lugar de un mensaje de advertencia. Un mensaje de error hace que la compilación falle. En este ejemplo, también desactiva las advertencias PERFORMANCE.

Definición de Niveles de Advertencia del Compilador: mediante PLSQL_WARNINGS en SQL Developer



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

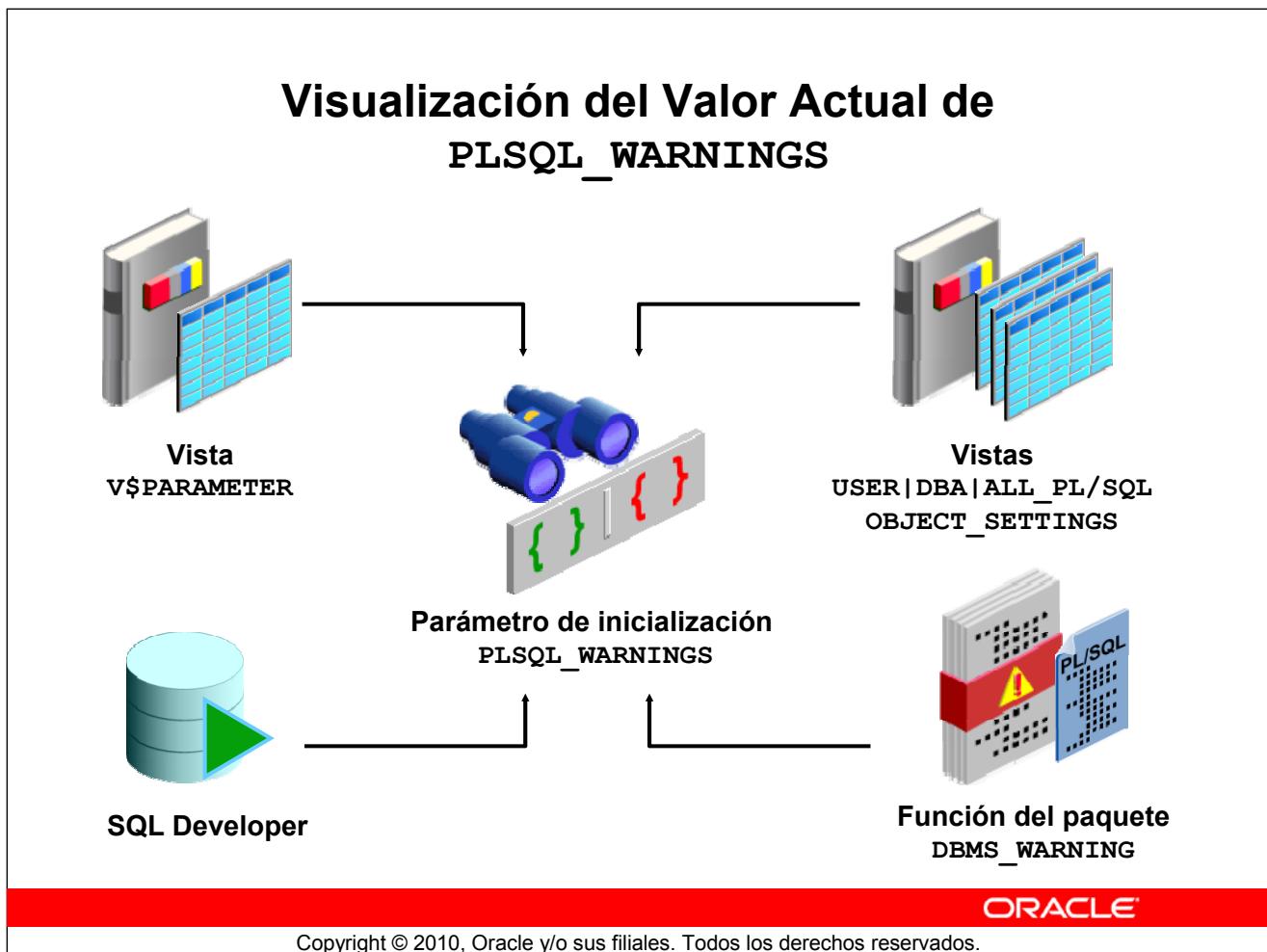
Definición de Niveles de Advertencia del Compilador: mediante PLSQL_WARNINGS en SQL Developer

El panel PL/SQL Compiler especifica opciones para la compilación de subprogramas PL/SQL. Si se selecciona la casilla de control **Generate PL/SQL Debug Information**, la información de depuración de PL/SQL se incluye en el código compilado; si no se selecciona esta opción, esta información de depuración no se incluye. La capacidad para parar en líneas de código individuales y el acceso del depurador a variables sólo se permiten en código compilado con información de depuración generada.

Definición y Visualización de las Categorías de Mensajes de Advertencia de Tiempo de Compilación PL/SQL en SQL Developer

Puede controlar la visualización de mensajes informativos, graves y relacionados con el rendimiento. El tipo **ALL** sustituye cualquier especificación individual para los otros tipos de mensajes. Para cada tipo de mensaje, puede especificar lo siguiente:

- **Ninguna entrada (en blanco):** utilice cualquier valor especificado para **ALL**; y, si no se especifica ningún valor, utilice el valor por defecto de Oracle.
- **Enable:** active la visualización de todos los mensajes de esta categoría.
- **Disable:** desactive la visualización de todos los mensajes de esta categoría.
- **Error:** active la visualización de sólo los mensajes de esta categoría.



Visualización del Valor Actual del Parámetro PLSQL_WARNINGS

Puede examinar el valor actual del parámetro PLSQL_WARNINGS emitiendo una sentencia SELECT en la vista V\$PARAMETER. Por ejemplo:

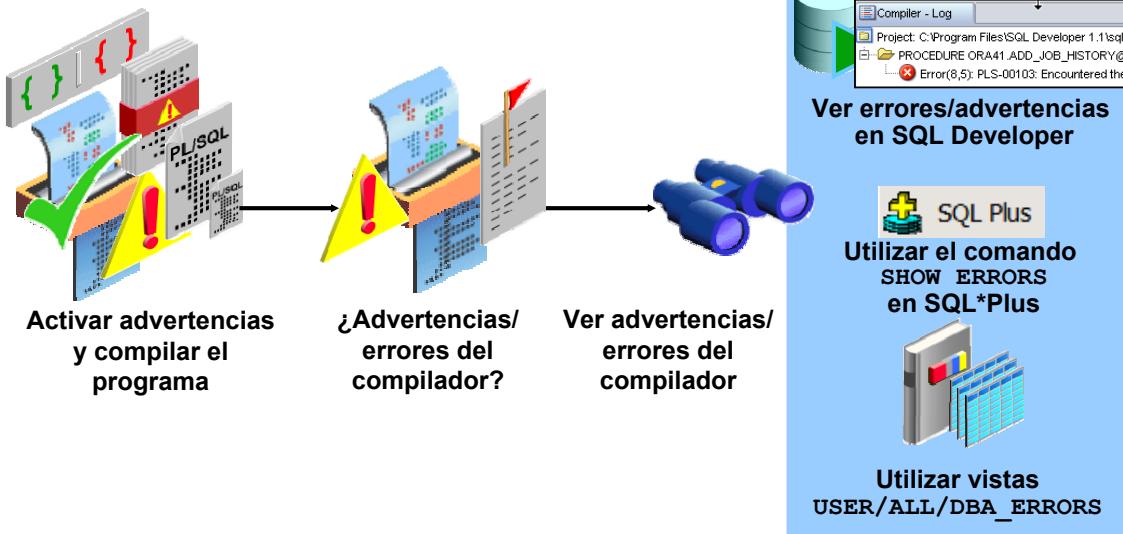
```
ALTER SESSION SET plsql_warnings = 'enable:severe',
    'enable:performance', 'enable:informational';
Session altered.
SELECT value FROM v$parameter WHERE name='plsql_warnings';
VALUE
-----
ENABLE:ALL
```

Asimismo, puede utilizar el paquete DBMS_WARNING.GET_WARNING_SETTING_STRING y el procedimiento para recuperar la configuración actual del parámetro PLSQL_WARNINGS:

```
DECLARE s VARCHAR2(1000);
BEGIN
    s := dbms_warning.get_warning_setting_string();
    dbms_output.put_line (s);
END;
/
```

```
anonymous block completed
ENABLE:ALL
```

Visualización de Advertencias del Compilador: mediante SQL Developer, SQL*Plus o las Vistas del Diccionario de Datos



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización de Advertencias del Compilador

Puede utilizar SQL*Plus para ver cualquier advertencia emitida como resultado de la compilación de un bloque PL/SQL. SQL*Plus indica que se ha producido una advertencia de compilación. Se muestra el mensaje “**SP2-08xx: <object> created with compilation warnings.**” para objetos compilados con los modificadores PERFORMANCE, INFORMATIONAL o SEVERE. No hay diferenciación entre los tres. Debe activar las advertencias del compilador antes de compilar el programa. Puede visualizar los mensajes de advertencia del compilador con uno de los métodos siguientes:

Mediante el comando SQL*Plus SHOW ERRORS

Este comando muestra cualquier error del compilador, incluidas las nuevas advertencias del compilador y los mensajes informativos. Este comando se llama inmediatamente después de utilizar un comando CREATE [PROCEDURE | FUNCTION | PACKAGE]. El comando SHOW ERRORS muestra advertencias y errores del compilador. Las nuevas advertencias del compilador y los mensajes informativos se “intercalan” con errores del compilador cuando se llama a SHOW ERRORS.

Mediante las Vistas del Diccionario de Datos

Puede seleccionar entre las vistas del diccionario de datos USER_ | ALL_ | DBA_ERRORS para visualizar advertencias del compilador PL/SQL. La columna ATTRIBUTES de estas vistas tiene un nuevo atributo denominado WARNING y el mensaje de advertencia aparece en la columna TEXT.

Mensajes de Advertencia de SQL*Plus: Ejemplo

```
CREATE OR REPLACE PROCEDURE bad_proc(p_out ...) IS
BEGIN
    . . . ;
END ;
/
```

SP2-0804: Procedure created with compilation warnings.

```
SHOW ERRORS;
Errors for PROCEDURE BAD_PROC:

LINE/COL      ERROR
-----
6/24          PLW-07203: parameter 'p_out' may benefit
                  from use of the NOCOPY compiler hint
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Mensajes de Advertencia de SQL*Plus: Ejemplo

Utilice el comando `SHOW ERRORS` en SQL*Plus para visualizar los errores de compilación de un procedimiento almacenado. Al especificar esta opción sin argumentos, SQL*Plus muestra los errores de compilación para el procedimiento almacenado creado o modificado más recientemente. Si SQL*Plus muestra un mensaje de advertencias de compilación después de que haya creado o modificado un procedimiento almacenado, puede utilizar los comandos `SHOW ERRORS` para obtener más información.

Con la introducción del soporte de advertencias PL/SQL, el rango de mensajes de comentarios se amplía para incluir un tercer mensaje, de la forma siguiente:

SP2-08xx: <object> created with compilation warnings.

Esto le permite diferenciar entre la incidencia de una advertencia de compilación y un error de compilación. Se debe corregir un error si desea utilizar el procedimiento almacenado, mientras que una advertencia sólo tiene fines informativos.

El prefijo SP2 se incluye con el mensaje de advertencia, porque esto le proporciona la capacidad para buscar el número de mensaje correspondiente en *SQL*Plus User's Guide and Reference* (Guía del Usuario y Referencia de SQL*Plus) para determinar la causa y la acción para el mensaje concreto.

Nota: el comando `SHOW` de SQL*Plus no está soportado en la versión 1.5.4 de SQL Developer que se utiliza en esta clase. Puede ver los errores y advertencias del compilador con las vistas del diccionario de datos `USER_ | ALL_ | DBA_ERRORS`.

Instrucciones para el Uso de `PLSQL_WARNINGS`

- La configuración para el parámetro `PLSQL_WARNINGS` se almacena junto con cada subprograma compilado.
- Si recompila el subprograma con una de las sentencias siguientes, se utilizará la configuración actual para dicha sesión:
 - `CREATE OR REPLACE`
 - `ALTER ... COMPILE`
- Si recompila el subprograma con la sentencia `ALTER ... COMPILE` con la cláusula `REUSE SETTINGS`, se utiliza la configuración original almacenada con el programa.



Copyright © 2010, Oracle y/o sus filiales. 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.

La configuración para el parámetro `PLSQL_WARNINGS` se almacena junto con cada subprograma compilado. Si recompila el subprograma con una sentencia `CREATE OR REPLACE`, se utilizará la configuración actual 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.

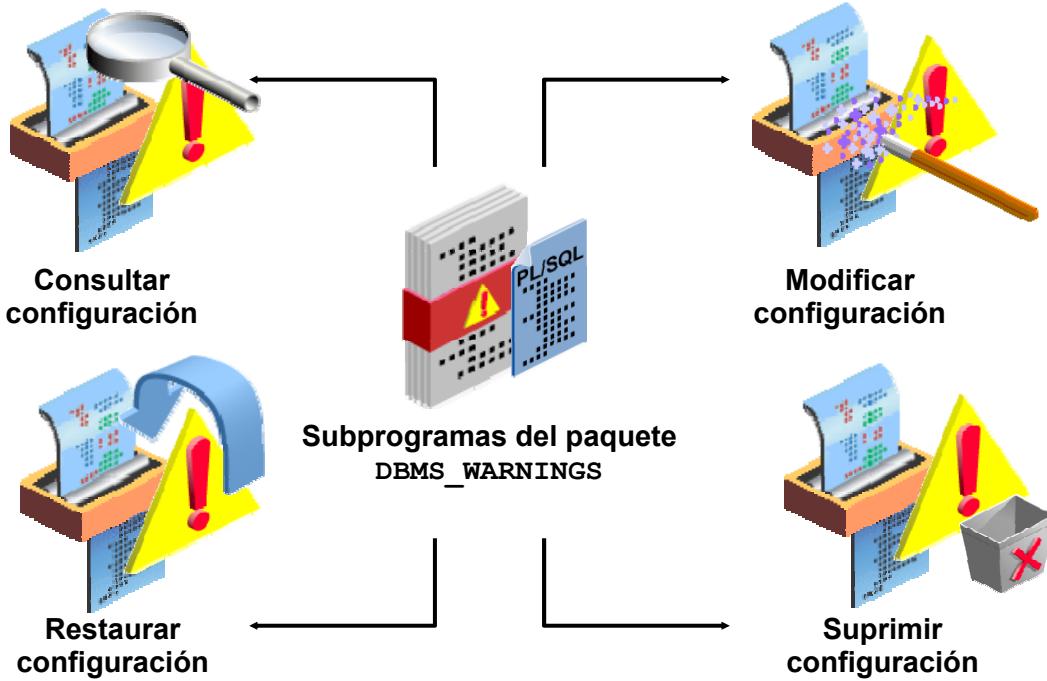
Agenda de la Lección

- Uso de los parámetros de inicialización `PLSQL_CODE_TYPE` y `PLSQL_OPTIMIZE_LEVEL` para la compilación PL/SQL
- Uso de las advertencias de tiempo de compilación de PL/SQL:
 - Uso del parámetro de inicialización `PLSQL_WARNINGS`
 - Uso de los subprogramas del paquete `DBMS_WARNING`

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Definición de Niveles de Advertencia del Compilador: mediante el Paquete DBMS_WARNING



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Definición de Niveles de Advertencia del Compilador: mediante el Paquete DBMS_WARNING

Utilice el paquete DBMS_WARNING para proporcionar una forma de manipular mediante programación el comportamiento de la configuración de advertencia PL/SQL del sistema o sesión actual. 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 la configuración actual del sistema o sesión.

El paquete DBMS_WARNING 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.

Definición de Niveles de Advertencia del Compilador: mediante el Paquete DBMS_WARNING (continuación)

Visión General de las Advertencias de Tiempo de Compilación PL/SQL para Subprogramas: Ejemplo

Suponga 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 configuración de nivel de advertencia adecuada 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_WARNING, puede detectar la configuración de advertencia actual, cambiar la configuración para que se ajusten a los requisitos del negocio y restaurar la configuración original una vez terminado el procesamiento.

Cuando utiliza el comando ALTER SESSION o ALTER SYSTEM para definir el parámetro PLSQL_WARNINGS, el nuevo valor especificado sustituye completamente el valor anterior. Hay un nuevo paquete, DBMS_WARNING, disponible desde Oracle Database 10g en adelante, que tiene interfaces para consultar y cambiar incrementalmente la configuración del parámetro PLSQL_WARNINGS y hacer que sea más específico para sus requisitos.

El paquete DBMS_WARNING se puede utilizar para cambiar el parámetro PLSQL_WARNINGS incrementalmente, para que pueda definir las advertencias que desea definir, sin tener que solucionar cómo mantener los valores de cualquier advertencia que no le interese directamente. Por ejemplo, el DBA sólo puede activar advertencias graves para toda la base de datos en el archivo de parámetros de inicialización, pero un desarrollador que prueba nuevo código puede que desee ver mensajes de rendimiento e informativos concretos. A continuación, el desarrollador puede utilizar el paquete DBMS_WARNING para agregar incrementalmente las advertencias concretas que desea ver. Esto permite al desarrollador ver los mensajes que desea ver sin sustituir la configuración del DBA.

Uso de los Subprogramas del Paquete DBMS_WARNING

Supuesto	Subprogramas que Utilizar
Definir advertencias	ADD_WARNING_SETTING_CAT (procedimiento) ADD_WARNING_SETTING_NUM (procedimiento)
Consultar advertencias	GET_WARNING_SETTING_CAT (función) GET_WARNING_SETTING_NUM (función) GET_WARNING_SETTING_STRING (función)
Sustituir advertencias	SET_WARNING_SETTING_STRING (procedimiento)
Obtener nombres de categorías de advertencias	GET_CATEGORY (función)

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de los Subprogramas DBMS_WARNING

A continuación se muestra una lista de los subprogramas DBMS_WARNING:

- ADD_WARNING_SETTING_CAT: modifica la configuración de advertencia actual de sesión o de sistema de warning_category previamente proporcionados
- ADD_WARNING_SETTING_NUM: modifica la configuración de advertencia actual de sesión o de sistema de warning_number previamente proporcionados
- GET_CATEGORY: devuelve el nombre de categoría para el número de mensaje especificado
- GET_WARNING_SETTING_CAT: devuelve la categoría de advertencia concreta en la sesión
- GET_WARNING_SETTING_NUM: devuelve el número de advertencia concreto en la sesión
- GET_WARNING_SETTING_STRING: devuelve toda la cadena de advertencia para la sesión actual
- SET_WARNING_SETTING_STRING: sustituye la configuración anterior por el nuevo valor

Nota: para obtener más información sobre los subprogramas anteriores, consulte *Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (11.2) [Referencia de Tipos y Paquetes PL/SQL de Oracle Database 11g Versión 2 (11.2)]* u *Oracle Database PL/SQL Packages and Types Reference 10g (Referencia de Tipos y Paquetes PL/SQL de Oracle Database 10g)* (según proceda para la versión que utiliza).

Procedimientos DBMS_WARNING: Sintaxis, Parámetros y Valores Permitidos

```
EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_CAT (
    warning_category      IN      VARCHAR2,
    warning_value         IN      VARCHAR2,
    scope                 IN      VARCAHR2);
```

```
EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_NUM (
    warning_number        IN      NUMBER,
    warning_value         IN      VARCHAR2,
    scope                 IN      VARCAHR2);
```

```
EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING (
    warning_value         IN      VARCHAR2,
    scope                 IN      VARCHAR2);
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Procedimientos DBMS_WARNING: Sintaxis, Parámetros y Valores Permitidos

`warning_category` es el nombre de la categoría. Los valores permitidos son:

- ALL
- INFORMATIONAL
- SEVERE
- PERFORMANCE

`warning_value` es el valor para la categoría. Los valores permitidos son:

- ENABLE
- DISABLE
- ERROR

`warning_number` es el número de mensaje de advertencia. Los valores permitidos son todos números de advertencia válidos.

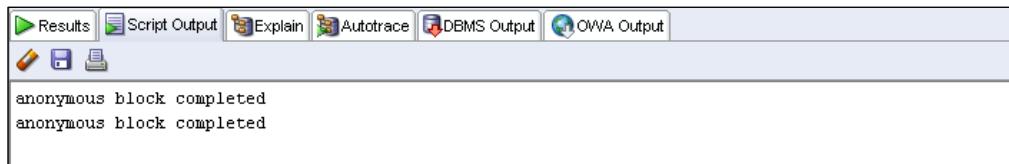
`scope` especifica si los cambios se realizan en el contexto de la sesión o del sistema. Los valores permitidos son SESSION o SYSTEM. Para utilizar SYSTEM, se necesita el privilegio ALTER SYSTEM.

Procedimientos DBMS_WARNING: Ejemplo

```
-- Establish the following warning setting string in the
-- current session:
-- ENABLE:INFORMATIONAL,
-- DISABLE:PERFORMANCE,
-- ENABLE:SEVERE

EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING(
  'ENABLE:ALL' , 'SESSION');

EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_CAT(
  'PERFORMANCE' , 'DISABLE' , 'SESSION');
```



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Procedimientos DBMS_WARNING: Ejemplo

Mediante el procedimiento SET_WARNING_SETTING_STRING, puede definir un valor de advertencia. Si tiene varias configuraciones 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 más configuraciones a la cadena inicial.

En el ejemplo de la diapositiva se establece la siguiente cadena de valor de advertencia en la sesión actual:

ENABLE:INFORMATIONAL,DISABLE:PERFORMANCE,ENABLE:SEVERE

Funciones DBMS_WARNING: Sintaxis, Parámetros y Valores Permitidos

```
DBMS_WARNING.GET_WARNING_SETTING_CAT (
    warning_category IN VARCHAR2) RETURN warning_value;
```

```
DBMS_WARNING.GET_WARNING_SETTING_NUM (
    warning_number IN NUMBER) RETURN warning_value;
```

```
DBMS_WARNING.GET_WARNING_SETTING_STRING
RETURN pls_integer;
```

```
DBMS_WARNING.GET_CATEGORY (
    warning_number IN pls_integer) RETURN VARCHAR2;
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Funciones DBMS_WARNING: Sintaxis, Parámetros y Valores Permitidos

`warning_category` es el nombre de la categoría. Los valores permitidos son:

- ALL
- INFORMATIONAL
- SEVERE
- PERFORMANCE

`warning_number` es el número de mensaje de advertencia. Los valores permitidos son todos números de advertencia válidos.

`scope` especifica si los cambios se realizan en el contexto de la sesión o del sistema. Los valores permitidos son SESSION o SYSTEM. Para utilizar SYSTEM, se necesita el privilegio ALTER SYSTEM.

Nota: utilice la función GET_WARNING_SETTING_STRING cuando no tenga el privilegio SELECT en las tablas fijas v\$parameter o v\$parameter2 o si desea analizar la cadena de advertencia y, a continuación, modificar y definir el nuevo valor mediante SET_WARNING_SETTING_STRING.

Funciones DBMS_WARNING: Ejemplo

```
-- Determine the current session warning settings
SET SERVEROUTPUT ON
EXECUTE DBMS_OUTPUT.PUT_LINE( -
DBMS_WARNING.GET_WARNING_SETTING_STRING);
```

```
anonymous block completed
ENABLE:INFORMATIONAL,DISABLE:PERFORMANCE,ENABLE:SEVERE
```

```
-- Determine the category for warning message number
-- PLW-07203
SET SERVEROUTPUT ON
EXECUTE DBMS_OUTPUT.PUT_LINE( -
DBMS_WARNING.GET_CATEGORY(7203));
```

```
anonymous block completed
PERFORMANCE
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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

```

CREATE OR REPLACE PROCEDURE compile_code(p_pkg_name VARCHAR2) IS
  v_warn_value  VARCHAR2(200);
  v_compile_stmt VARCHAR2(200) := 
    'ALTER PACKAGE ''' || p_pkg_name ||' COMPILE';

BEGIN
  v_warn_value := DBMS_WARNING.GET_WARNING_SETTING_STRING;
  DBMS_OUTPUT.PUT_LINE('Current warning settings: ' ||
    v_warn_value);
  DBMS_WARNING.ADD_WARNING_SETTING_CAT(
    'PERFORMANCE', 'DISABLE', 'SESSION');
  DBMS_OUTPUT.PUT_LINE('Modified warning settings: ' ||
    DBMS_WARNING.GET_WARNING_SETTING_STRING);
  EXECUTE IMMEDIATE v_compile_stmt;
  DBMS_WARNING.SET_WARNING_SETTING_STRING(v_warn_value,
    'SESSION');
  DBMS_OUTPUT.PUT_LINE('Restored warning settings: ' ||
    DBMS_WARNING.GET_WARNING_SETTING_STRING);
END;
/

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de DBMS_WARNING: Ejemplo

Nota: antes de ejecutar el código proporcionado en el ejemplo de la diapositiva, debe crear el script MY_PKG que se encuentra en demo_10_33.sql. Este script de demostración crea el paquete MY_PKG.

En el ejemplo de la diapositiva, el procedimiento `compile_code` se designa para compilar un paquete PL/SQL con nombre. El código suprime las advertencias de la categoría PERFORMANCE. La configuración de advertencia del entorno de llamada se debe restaurar después de realizar la compilación. El código no sabe cuál es la configuración de advertencia del entorno de llamada; se utiliza la función `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_code` modifica el nivel de advertencia de sesión actual desactivando las advertencias para la categoría PERFORMANCE. El código también imprime la configuración de advertencia original, modificada y restaurada.

Uso de DBMS_WARNING: Ejemplo

```
EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING(-  
'ENABLE:ALL', 'SESSION');
```

```
anonymous block completed
```

```
@code_10_32_s.sql
```

```
PROCEDURE compile_code(p_pkg_name Compiled.
```

```
@code_10_33_cs.sql -- compiles the DEPT_PKG package
```

```
anonymous block completed  
Current warning settings: ENABLE:ALL  
Modified warning settings: ENABLE:INFORMATIONAL,DISABLE:PERFORMANCE,ENABLE:SEVERE  
Restored warning settings: ENABLE:ALL
```

ORACLE

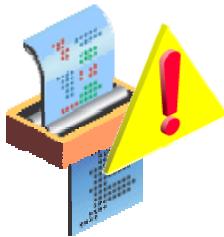
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de DBMS_WARNING: Ejemplo (continuación)

En el ejemplo de la diapositiva, se prueba el ejemplo proporcionado en la diapositiva anterior. En primer lugar, active todas las advertencias del compilador. A continuación, ejecute el script de la página anterior. Finalmente, llame al procedimiento `compile_code` y transfírele un nombre de paquete existente, `DEPT_PKG`, como parámetro.

Uso del Mensaje de Advertencia PLW 06009

- La advertencia PLW indica que el manejador OTHERS de la subrutina PL/SQL puede salir sin ejecutar:
 - Alguna forma de RAISE, o bien
 - Una llamada al procedimiento estándar RAISE_APPLICATION_ERROR
- Una buena práctica de programación sugiere que los manejadores OTHERS siempre deben transferir una excepción en sentido ascendente.



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

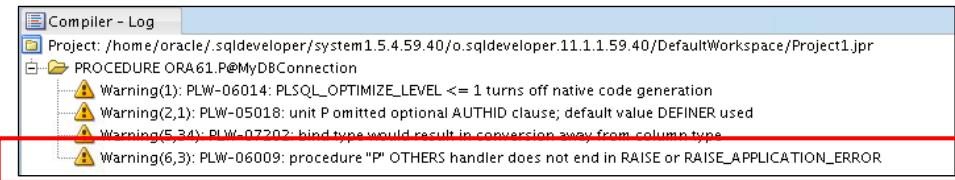
Uso de la Nueva Advertencia PLW 06009

Como buena práctica de programación, debe hacer que el manejador de excepciones OTHERS transfiera la excepción en sentido ascendente a la subrutina de llamada. Si no puede agregar esta funcionalidad, corre el riesgo de que las excepciones pasen desapercibidas. Para evitar este error en el código, puede activar advertencias para la sesión y recompilar el código que desea verificar. Si el manejador OTHERS no maneja la excepción, la advertencia PLW 06009 le informará. Tenga en cuenta que esta advertencia es específica de Oracle Database 11g.

Nota: PLW 06009 no es el único mensaje de advertencia nuevo de Oracle Database 11g. Para obtener una lista completa de todas las advertencias PLW, consulte *Oracle Database Error Messages 11g Release 2 (11.2)* [Mensajes de Error de Oracle Database 11g Versión 2 (11.2)].

Advertencia PLW 06009: Ejemplo

```
CREATE OR REPLACE PROCEDURE p(i IN VARCHAR2)
IS
BEGIN
    INSERT INTO t(col_a) VALUES (i);
EXCEPTION
    WHEN OTHERS THEN null;
END p;
/
ALTER PROCEDURE P COMPILE
PLSQL_warnings = 'enable:all' REUSE SETTINGS;
```



```
SELECT *
FROM user_errors
WHERE name = 'P'
```

ID	NAME	TYPE	SEQUEN...	LINE	POSITION	TEXT	ATTR...	MESSAGE_NUMBER
1	P	PROCEDURE	1	4	34	PLW-07202: bind type would result in conversion away from column type	WARNING	7202
2	P	PROCEDURE	2	1	1	PLW-05018: unit P omitted optional AUTHID clause; default value DEFINER used	WARNING	5018
3	P	PROCEDURE	3	0	0	PLW-06014: PLSQL_OPTIMIZE_LEVEL <= 1 turns off native code generation	WARNING	6014
4	P	PROCEDURE	4	5	3	PLW-06009: procedure "P" OTHERS handler does not end in RAISE or RAISE_APPLICATION... WARNING	WARNING	6009

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Advertencia PLW 06009: Ejemplo

Después de ejecutar el primer ejemplo de código de la diapositiva y después de compilar el procedimiento mediante el árbol Object Navigation, el separador **Compiler – Log** muestra la advertencia PLW-06009.

También puede utilizar la vista de diccionario de datos `user_error` para mostrar el error. La definición de la tabla `t` utilizada en el ejemplo de la diapositiva es la siguiente:

```
CREATE TABLE t (col_a NUMBER);
```

Prueba

Las categorías de mensajes de advertencia de tiempo de compilación PL/SQL son:

1. SEVERE
2. PERFORMANCE
3. INFORMATIONAL
4. All
5. CRITICAL



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: 1, 2, 3, 4

Los mensajes de advertencia PL/SQL se dividen en categorías, por lo que puede suprimir o mostrar grupos de advertencias similares durante la compilación. Las categorías son:

- SEVERE: mensajes para condiciones que pueden provocar un comportamiento inesperado o resultados incorrectos, como problemas de alias con parámetros.
- PERFORMANCE: mensajes para condiciones que pueden provocar problemas de rendimiento, como la transferencia de un valor VARCHAR2 a una columna NUMBER en una sentencia INSERT.
- INFORMATIONAL: mensajes para condiciones que no tienen un efecto en el rendimiento o corrección, pero que puede que desee cambiar para que el código se pueda mantener mejor, como código inaccesible que nunca se puede ejecutar.
- ALL: muestra todas las categorías.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Utilizar los parámetros de inicialización del compilador PL/SQL
- Utilizar las advertencias de tiempo de compilación de PL/SQL



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica 10: Visión General

En esta práctica se abordan los siguientes temas:

- Visualización de los parámetros de inicialización del compilador
- Activación de la compilación nativa para la sesión y compilación de un procedimiento
- Desactivación de las advertencias del compilador y, a continuación, restauración de la configuración de advertencia de sesión original
- Identificación de las categorías de algunos números de mensaje de advertencia del compilador



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica 10: Visión General

En esta práctica, visualizará los parámetros de inicialización del compilador. A continuación, activará la compilación nativa para la sesión y compilará un procedimiento. Después, suprimirá todas las categorías de advertencias del compilador para, a continuación, restaurar la configuración de advertencia de sesión original. Por último, identificará las categorías de algunos números de mensaje de advertencia del compilador.

11

Gestión del Código PL/SQL

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

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

- Describir y utilizar la compilación condicional
- Ocultar el código fuente PL/SQL mediante ocultación dinámica y la utilidad Wrap



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos de la Lección

En esta lección se presenta la compilación condicional y la ocultación o encapsulamiento de código PL/SQL.

Agenda de la Lección

- Uso de la compilación condicional
- Ocultación del código PL/SQL

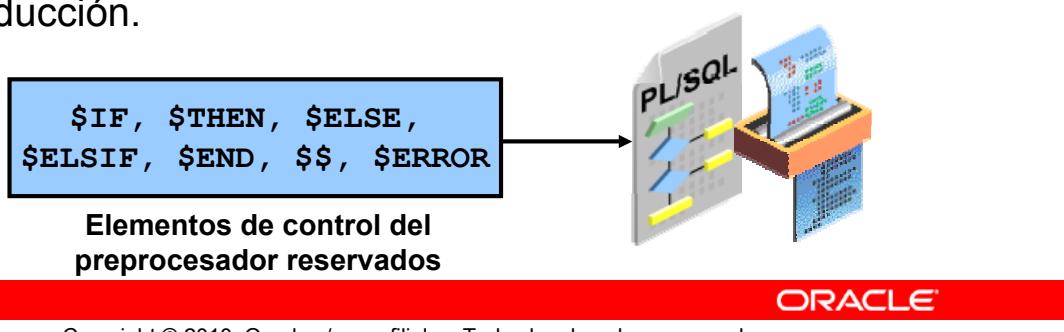
ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

¿Qué es la Compilación Condicional?

Le permite personalizar la funcionalidad de una aplicación PL/SQL sin eliminar ningún código fuente:

- Utilice la última funcionalidad con la última versión de la base de datos o desactive las nuevas funciones para ejecutar la aplicación con versiones anteriores de la base de datos.
- Active la funcionalidad de depuración o rastreo en el entorno de desarrollo y oculte dicha funcionalidad en la aplicación mientras se ejecuta en una ubicación de producción.



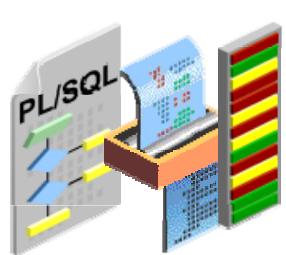
¿Qué es la Compilación Condicional?

La compilación condicional permite incluir selectivamente código, según los valores de las condiciones evaluadas durante la compilación. Por ejemplo, la compilación condicional permite determinar qué funciones PL/SQL de una aplicación PL/SQL se utilizan para versiones de base de datos concretas. Las últimas funciones PL/SQL de una aplicación se pueden ejecutar en una nueva versión de la base de datos y, al mismo tiempo, dichas funciones pueden ser condicionales para que la misma aplicación sea compatible con una versión de la base de datos anterior. La compilación condicional también resulta útil cuando desea ejecutar procedimientos de depuración en un entorno de desarrollo, pero desea desactivar las rutinas de depuración en un entorno de producción.

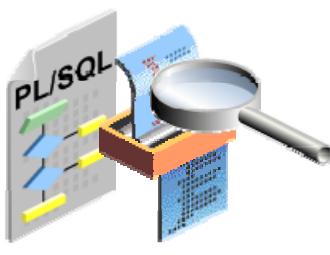
Ventajas de la Compilación Condicional

- Soporte de varias versiones del mismo programa en un código fuente
- Fácil mantenimiento y depuración del código
- Fácil migración del código a una versión distinta de la base de datos

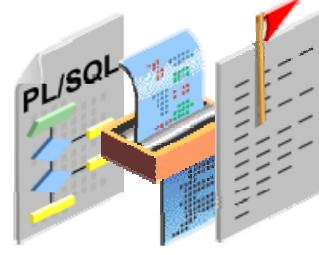
Funcionamiento de la Compilación Condicional



Directivas de selección:
utilice el elemento `$IF`.



Directivas de consulta:
utilice el elemento `$$`.



Directivas de error:
utilice el elemento
`$ERROR`.



Paquete
`DBMS_PREPROCESSOR`



Paquete
`DBMS_DB_VERSION`

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Funcionamiento de la Compilación Condicional

Puede utilizar la compilación condicional embebiendo las directivas en los programas de origen PL/SQL. Cuando se envía el programa PL/SQL para la compilación, un preprocesador evalúa estas directivas y selecciona partes del programa que se va a compilar. A continuación, el origen del programa seleccionado se cede al compilador para su compilación.

Las directivas de consulta utilizan el elemento `$$` para realizar consultas sobre el entorno de compilación, como el valor de los parámetros de inicialización de un compilador PL/SQL `PLSQL_CCFLAGS` o `PLSQL_OPTIMIZE_LEVEL` para la unidad compilada. Esta directiva se puede utilizar junto con la directiva de selección condicional para seleccionar las partes del programa que compilar.

Las directivas de selección pueden probar directivas de consulta o constantes estáticas del paquete mediante la construcción `$IF` para bifurcar secciones de código para la posible compilación si se cumple una condición.

Las directivas de error emiten un error de compilación si se encuentra una condición inesperada durante la compilación condicional mediante el elemento `$ERROR`.

El paquete `DBMS_DB_VERSION` proporciona constantes de versión y de versión final de la base de datos que se pueden utilizar para la compilación condicional.

El paquete `DBMS_PREPROCESSOR` proporciona subprogramas para acceder al texto del código postprocesado seleccionado por las directivas de compilación condicional en una unidad PL/SQL.

Uso de Directivas de Selección

```
$IF <Boolean-expression> $THEN Text
$ELSEIF <Boolean-expression> $THEN Text
.
.
.
$ELSE Text
$END
```

```
DECLARE
CURSOR cur IS SELECT employee_id FROM
employees WHERE
$IF myapp_tax_package.new_tax_code $THEN
    salary > 20000;
$ELSE
    salary > 50000;
$END
BEGIN
    OPEN cur;
.
.
.
END;
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

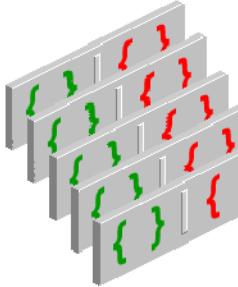
Uso de Directivas de Selección

La directiva de selección condicional se parece y funciona como el mecanismo IF-THEN-ELSE en el PL/SQL adecuado. Cuando el preprocesador encuentra \$THEN, verifica que el texto entre \$IF y \$THEN es una expresión estática. Si la comprobación es correcta y el resultado de la evaluación es TRUE, el texto del programa PL/SQL entre \$THEN y \$ELSE (o \$ELSIF) se selecciona para la compilación.

La condición de selección (la expresión entre \$IF y \$THEN) se puede construir haciendo referencia a constantes definidas en otro paquete, una directiva de consulta o una combinación de las dos.

En el ejemplo de la diapositiva, la directiva de selección condicional elige entre dos versiones del cursor, `cur`, en la base del valor de `MYAPP_TAX_PACKAGE.NEW_TAX_CODE`. Si el valor es TRUE, se seleccionan los empleados con `salary > 20000`; de lo contrario, se seleccionan los empleados con `salary > 50000`.

Uso de Directivas de Consulta Predefinidas y Definidas por el Usuario



```
PLSQL_CCFLAGS
PLSQL_CODE_TYPE
PLSQL_OPTIMIZE_LEVEL
PLSQL_WARNINGS
NLS_LENGTH_SEMANTICS
PLSQL_LINE
PLSQL_UNIT
```

Directivas de consulta predefinidas

```
PLSQL_CCFLAGS = 'plsql_ccflags:true,debug:true,debug:0';
```

Directivas de consulta definidas por el usuario

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Directivas de Consulta Predefinidas y Definidas por el Usuario

Una directiva de consulta puede ser predefinida o definida por el usuario. A continuación, se describe el orden del flujo de procesamiento cuando la compilación condicional intenta resolver una directiva de consulta:

1. El identificador se utiliza como directiva de consulta en forma de `$$id` para la clave de búsqueda.
2. El algoritmo de dos pasos continúa de la siguiente forma:
 - a. La cadena del parámetro de inicialización `PLSQL_CCFLAGS` se explora de derecha a izquierda, buscando con el identificador un nombre coincidente (no sensible a mayúsculas/minúsculas); termina si se encuentra.
 - b. Se buscan las directivas de consulta predefinidas; termina si se encuentran.
3. Si `$$ID` no se puede resolver en un valor, se informa del mensaje de advertencia `PLW-6003` si el texto de origen no está encapsulado. El literal `NUL` se sustituye como valor para directivas de consulta no definidas. Tenga en cuenta que si se encapsula el código PL/SQL, el mensaje de advertencia se desactiva de forma que la directiva de consulta no definida no se revela.

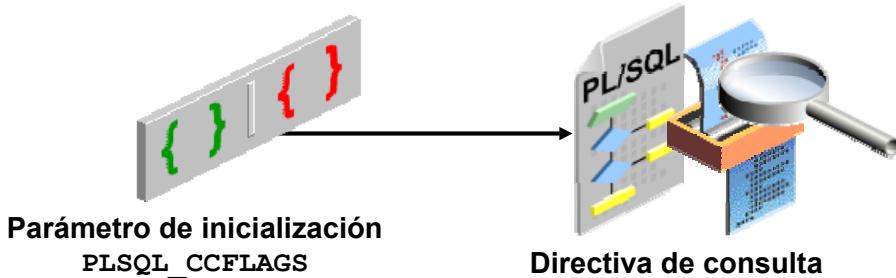
En el ejemplo de la diapositiva, el valor de `$$debug` es 0 y el valor de `$$plsql_ccflags` es TRUE. Tenga en cuenta que el valor de `$$plsql_ccflags` se resuelve en `plsql_ccflags` definido por el usuario en el valor del parámetro del compilador `PLSQL_CCFLAGS`. Esto se produce porque una directiva definida por el usuario sustituye la predefinida.

Parámetro PLSQL_CCFLAGS y Directiva de Consulta

Utilice el parámetro `PLSQL_CCFLAGS` para controlar la compilación condicional de cada unidad de biblioteca PL/SQL independientemente.

```
PLSQL_CCFLAGS = '<v1>:<c1>,<v2>:<c2>,...,<vn>:<cn>'
```

```
ALTER SESSION SET
  PLSQL_CCFLAGS = 'plsql_ccflags:true, debug:true, debug:0' ;
```



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Parámetro `PLSQL_CCFLAGS` y Directiva de Consulta

Oracle Database 10g Versión 2 introdujo un nuevo parámetro de inicialización de Oracle `PLSQL_CCFLAGS` para su uso con la compilación condicional. Este parámetro dinámico permite configurar pares de nombres-valores. A continuación, se puede hacer referencia a los nombres (denominados nombres de indicadores) en directivas de consulta. `PLSQL_CCFLAGS` proporciona un mecanismo para permitir a los programadores de PL/SQL controlar la compilación condicional de cada unidad de biblioteca PL/SQL independientemente.

Valores

- `vi`: tiene la forma de un identificador PL/SQL sin comillas. No está restringido y puede ser una palabra reservada o una palabra clave. El texto no es sensible a mayúsculas/minúsculas. Cada uno se denomina indicador o nombre del indicador. Cada `vi` se puede producir más de una vez en la cadena, cada incidencia puede tener un valor de indicador diferente y los valores de indicadores pueden ser de tipos distintos.
- `ci`: puede ser uno de los siguientes:
 - Un literal booleano PL/SQL
 - Un literal `PLS_INTEGER`
 - El literal es `NULL` (valor por defecto). El texto no es sensible a mayúsculas/minúsculas.
 Cada uno se denomina valor de indicador y corresponde a un nombre de indicador.

Visualización del Valor del Parámetro de Inicialización `PLSQL_CCFLAGS`

```
SELECT name, type, plsql_ccflags
FROM user_plsql_object_settings
```

Results:

NAME	TYPE	PLSQL_CCFLAGS
1 DEPT_PKG	PACKAGE	(null)
2 DEPT_PKG	PACKAGE BODY	(null)
3 TAXES_PKG	PACKAGE	(null)
4 TAXES_PKG	PACKAGE BODY	(null)
5 EMP_PKG	PACKAGE	(null)
6 EMP_PKG	PACKAGE BODY	(null)
7 SECURE_DML	PROCEDURE	(null)
8 SECURE_EMPLOYEES	TRIGGER	(null)
9 ADD_JOB_HISTORY	PROCEDURE	plsql_ccflags:true, debug:true, debug:0
10 UPDATE_JOB_HISTORY	TRIGGER	(null)

...

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización del Valor del Parámetro de Inicialización `PLSQL_CCFLAGS`

Utilice las vistas del diccionario de datos `USER|ALL|DBA_PLSQL_OBJECT_SETTINGS` para visualizar la configuración de un objeto PL/SQL.

Puede definir cualquier valor permitido para `PLSQL_CCFLAGS`. Sin embargo, Oracle recomienda que este parámetro se utilice para controlar la compilación condicional de depuración o rastreo de código.

Los nombres de indicador se pueden definir en cualquier identificador, incluidas palabras reservadas y palabras clave. Los valores deben ser los literales `TRUE`, `FALSE` o `NULL`, o un literal `PLS_INTEGER`. Los nombres y valores de indicador no son sensibles a mayúsculas y minúsculas. El parámetro `PLSQL_CCFLAGS` es un parámetro del compilador PL/SQL (como otros parámetros del compilador) y se almacena con la unidad de programa PL/SQL. Por lo tanto, si el programa PL/SQL se recompila posteriormente con la cláusula `REUSE SETTINGS` (ejemplo, `ALTER PACKAGE ...REUSE SETTINGS`), se utiliza el mismo valor de `PLSQL_CCFLAGS` para la recompilación. Puesto que el parámetro `PLSQL_CCFLAGS` se puede definir en un valor distinto para cada unidad PL/SQL, proporciona un método práctico para controlar la compilación condicional por unidad.

Parámetro PLSQL_CCFLAGS y Directiva de Consulta: Ejemplo

```
ALTER SESSION SET PLSQL_CCFLAGS = 'Tracing:true';
CREATE OR REPLACE PROCEDURE P IS
BEGIN
  $IF $$tracing $THEN
    DBMS_OUTPUT.PUT_LINE ('TRACING');
  $END
END P;
```

```
ALTER SESSION SET succeeded.
PROCEDURE P Compiled.
```

```
SELECT name, plsql_ccflags
FROM USER_PLSQL_OBJECT_SETTINGS
WHERE name = 'P';
```

Results:

	NAME	PLSQL_CCFLAGS
1	P	Tracing:true

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Parámetro PLSQL_CCFLAGS y Directiva de Consulta: Ejemplo

En el ejemplo de la diapositiva, se define el parámetro y, a continuación, se crea el procedimiento. El valor se almacena con cada unidad PL/SQL.

Uso de Directivas de Error de Compilación Condicional para Emitir Errores Definidos por el Usuario

```
$ERROR varchar2_static_expression $END

ALTER SESSION SET Plsql_CCFlags = ' Trace_Level:3 '
/ CREATE PROCEDURE P IS
BEGIN
  $IF $$Trace_Level = 0 $THEN ...
  $ELSIF $$Trace_Level = 1 $THEN ...
  $ELSIF $$Trace_Level = 2 $THEN ...
  $Else $error 'Bad: '||$$Trace_Level $END -- error
                                -- directive
  $END -- selection directive ends
END P;

SHOW ERRORS
Errors for PROCEDURE P:
LINE/COL ERROR
-----
6/9      PLS-00179: $ERROR: Bad: 3
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Directivas de Error de Compilación Condicional para Emitir Errores Definidos por el Usuario

La directiva de error \$ERROR emite un error definido por el usuario y tiene el formato:

```
$ERROR varchar2_static_expression $END
```

Nota: varchar2_static_expression debe ser una expresión estática VARCHAR2.

Uso de Expresiones Estáticas con la Compilación Condicional

- **Expresiones estáticas booleanas:**
 - TRUE, FALSE, NULL, IS NULL, IS NOT NULL
 - >, <, >=, <=, =, <>, NOT, AND, OR
- **Expresiones estáticas PLS_INTEGER:**
 - -2147483648 a 2147483647, NULL
- **Las expresiones estáticas VARCHAR2 incluyen:**
 - ||, NULL, TO_CHAR
- **Constantes estáticas:**

```
static_constant CONSTANT datatype := static_expression;
```

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Expresiones Estáticas con la Compilación Condicional

Como se ha descrito anteriormente, un preprocesador procesa directivas condicionales antes de empezar la compilación correcta. Por lo tanto, en las directivas de compilación condicional sólo se permiten las expresiones que se pueden evaluar completamente en tiempo de compilación. Cualquier expresión que contenga referencias a variables o funciones que necesiten la ejecución de PL/SQL no están disponibles durante la compilación y no se pueden evaluar.

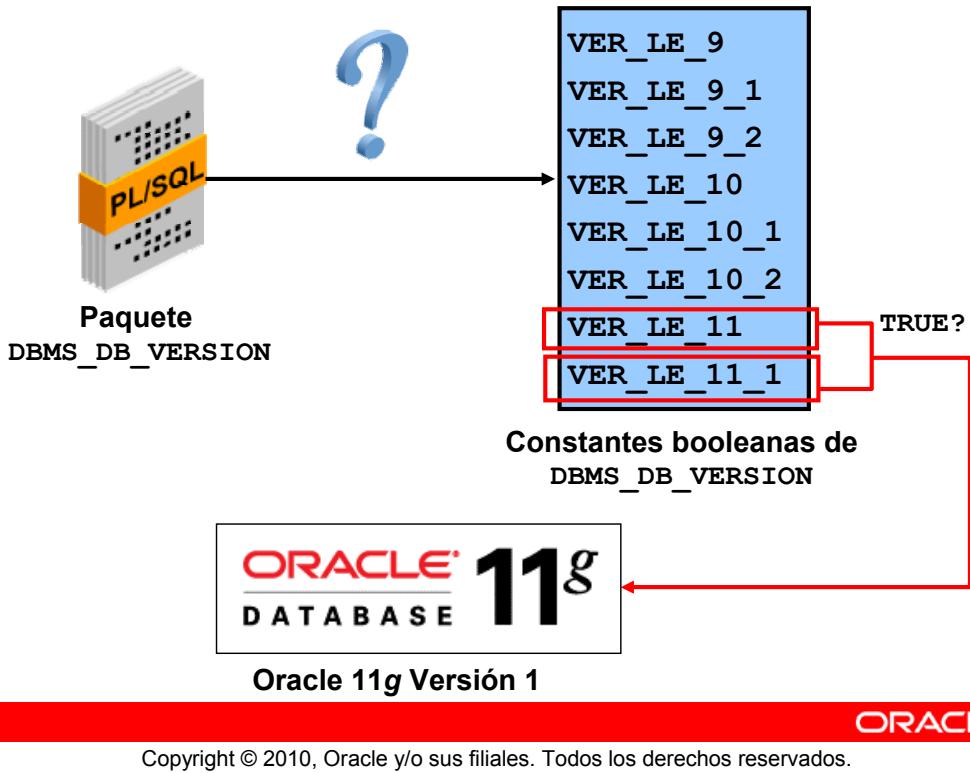
Este subjuego de expresiones PL/SQL permitidas en las directivas de compilación condicional se denomina expresiones estáticas. Las expresiones estáticas se definen con detenimiento para garantizar que si se recompila automáticamente una unidad sin ningún cambio en los valores de los que depende, las expresiones se evalúan de la misma forma y se compila el mismo origen.

Normalmente, las expresiones estáticas están compuestas de tres orígenes:

- Directivas de consulta marcadas con \$\$.
- Constantes definidas en paquetes PL/SQL como DBMS_DB_VERSION. Estos valores se pueden combinar y comparar mediante las operaciones ordinarias de PL/SQL.
- Literales como TRUE, FALSE, 'CA', 123, NULL.

Las expresiones estáticas también pueden contener operaciones que incluyan comparaciones, operaciones booleanas lógicas (como OR y AND) o concatenaciones de expresiones de caracteres estáticas.

Paquete DBMS_DB_VERSION: Constantes Booleanas



Paquete DBMS_DB_VERSION

Oracle Database 10g Versión 2 introdujo el paquete DBMS_DB_VERSION. Este paquete especifica los números de versión y de versión final de la base de datos Oracle que son útiles al realizar selecciones simples para la compilación condicional.

Las constantes representan una condición booleana que se evalúa en un valor menor o igual que la versión y la versión final, si está presente.

Ejemplo

VER_LE_11 indica que la versión de la base de datos ≤ 11 . Los valores de las constantes son TRUE o FALSE. Por ejemplo, en una base de datos Oracle Database 11g Versión 1, VER_LE_11 y VER_LE_11_1 son TRUE y todas las demás constantes son FALSE.

Constantes del Paquete DBMS_DB_VERSION

Nombre	Descripción
VER_LE_9	Versión <= 9.
VER_LE_9_1	Versión <= 9 y versión final <= 1.
VER_LE_9_2	Versión <= 9 y versión final <= 2.
VER_LE_10	Versión <= 10.
VER_LE_10_1	Versión <= 10 y versión final <= 1.
VER_LE_10_2	Versión <= 10 y versión final <= 2.
VER_LE_11	Versión <= 11.
VER_LE_11_1	Versión <= 11 y versión final <= 1.

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Paquete DBMS_DB_VERSION (continuación)

El paquete de Oracle Database 11g Versión 1 se muestra a continuación:

```
PACKAGE DBMS_DB_VERSION IS
    VERSION CONSTANT PLS_INTEGER := 11; -- RDBMS version
                                              -- number
    RELEASE CONSTANT PLS_INTEGER := 1;   -- RDBMS release
                                              -- number
    ver_le_9_1      CONSTANT BOOLEAN := FALSE;
    ver_le_9_2      CONSTANT BOOLEAN := FALSE;
    ver_le_9        CONSTANT BOOLEAN := FALSE;
    ver_le_10_1     CONSTANT BOOLEAN := FALSE;
    ver_le_10_2     CONSTANT BOOLEAN := FALSE;
    ver_le_10       CONSTANT BOOLEAN := FALSE;
    ver_le_11_1     CONSTANT BOOLEAN := TRUE;
    ver_le_11       CONSTANT BOOLEAN := TRUE;
END DBMS_DB_VERSION;
```

El paquete DBMS_DB_VERSION contiene constantes diferentes para las distintas versiones de Oracle Database. La versión del paquete DBMS_DB_VERSION de Oracle Database 11g Versión 1 utiliza las constantes mostradas en la diapositiva.

Uso de la Compilación Condicional con Versiones de la Base de Datos: Ejemplo

```

ALTER SESSION SET PLSQL_CCFLAGS = 'my_debug:FALSE, my_tracing:FALSE';
CREATE PACKAGE my_pkg AS
  SUBTYPE my_real IS
    -- Check the database version, if >= 10g, use BINARY_DOUBLE data type,
    -- else use NUMBER data type
    $IF DBMS_DB_VERSION.VERSION < 10 $THEN    NUMBER;
    $ELSE    BINARY_DOUBLE;
    $END
    my_pi my_real; my_e my_real;
  END my_pkg;
/
CREATE PACKAGE BODY my_pkg AS
BEGIN
  $IF DBMS_DB_VERSION.VERSION < 10 $THEN
    my_pi := 3.14016408289008292431940027343666863227;
    my_e  := 2.71828182845904523536028747135266249775;
  $ELSE
    my_pi := 3.14016408289008292431940027343666863227d;
    my_e  := 2.71828182845904523536028747135266249775d;
  $END
END my_pkg;
/

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de la Compilación Condicional con Versiones de la Base de Datos: Ejemplo

En este ejemplo también se muestra el uso del parámetro PLSQL_CCFLAGS. En primer lugar, defina el indicador de parámetro PLSQL_CCFLAGS para mostrar el código de depuración y la información de rastreo.

En el ejemplo de la diapositiva de esta página y de la siguiente, la compilación condicional se utiliza para especificar código para versiones de la base de datos. La compilación condicional se utiliza para determinar si el tipo de dato BINARY_DOUBLE se puede utilizar en los cálculos de unidades PL/SQL en la base de datos. El tipo de dato BINARY_DOUBLE sólo se puede utilizar en Oracle Database 10g o versiones posteriores. Si utiliza Oracle Database 10g, el tipo de dato para my_real es BINARY_DOUBLE; de lo contrario, el tipo de dato para my_real es NUMBER.

En la especificación del nuevo paquete, my_pkg, la compilación condicional se utiliza para comprobar la versión de la base de datos. En la definición del cuerpo del paquete, la compilación condicional se vuelve a utilizar para definir los valores de my_pi y my_e para cálculos futuros según la versión de la base de datos.

El resultado del código de ejemplo de la diapositiva es el siguiente:

```

ALTER SESSION SET succeeded.
PACKAGE my_pkg Compiled.
PACKAGE BODY my_pkg Compiled.

```

Uso de la Compilación Condicional con Versiones de la Base de Datos: Ejemplo

```

CREATE OR REPLACE PROCEDURE circle_area(p_radius my_pkg.my_real) IS
    v_my_area my_pkg.my_real;
    v_my_datatype VARCHAR2(30);
BEGIN
    v_my_area := my_pkg.my_pi * p_radius * p_radius;
    DBMS_OUTPUT.PUT_LINE('Radius: ' || TO_CHAR(p_radius)
        || ' Area: ' || TO_CHAR(v_my_area));
    $IF $$my_debug $THEN -- if my_debug is TRUE, run some debugging code
        SELECT DATA_TYPE INTO v_my_datatype FROM USER_ARGUMENTS
        WHERE OBJECT_NAME = 'CIRCLE_AREA' AND ARGUMENT_NAME = 'P_RADIUS';
        DBMS_OUTPUT.PUT_LINE('Datatype of the RADIUS argument is: ' ||
            v_my_datatype);
    $END
END;

```

PROCEDURE circle_area(p_radius Compiled.

CALL circle_area(50); -- Using Oracle Database 11g Release 2

CALL circle_area(50) succeeded.
Radius: 5.0E+001 Area: 7.8504102072252062E+003

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de la Compilación Condicional con Versiones de la Base de Datos: Ejemplo (continuación)

En el ejemplo de la diapositiva, se define un nuevo procedimiento denominado `circle_area`. Este procedimiento calcula el área de un círculo según los valores de las variables del paquete `my_pkg` definido en la página anterior. El procedimiento tiene un parámetro formal `IN`, `radius`.

El procedimiento declara un par de variables: `my_area`, que es del mismo tipo de dato que `my_real` en `my_pkg`, y `my_datatype`, que es `VARCHAR2(30)`.

En el cuerpo del procedimiento, `my_area` equivale al valor de `my_pi` definido en `my_pkg` multiplicado por el valor transferido al procedimiento como radio. Aparece un mensaje con el radio y el área del círculo, como se muestra en el segundo ejemplo de código de la diapositiva.

Nota: si desea definir `my_debug` en `TRUE`, puede realizar este cambio sólo para el procedimiento `circle_area` con la cláusula `REUSE SETTINGS`, de la siguiente forma:

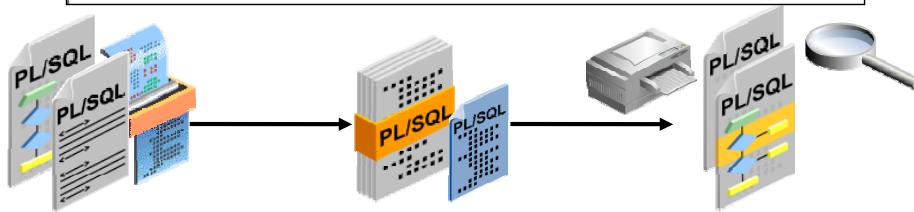
```
ALTER PROCEDURE circle_area COMPILE PLSQL_CCFLAGS =
    'my_debug:TRUE' REUSE SETTINGS;
```

Uso de Procedimientos DBMS_PREPROCESSOR para Imprimir o Recuperar Texto de Origen

CALL

```
DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE('PACKAGE'
, 'ORA61', 'MY_PKG');
```

```
CALL DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE('PACKAGE', succeeded.
PACKAGE my_pkg AS
SUBTYPE my_real IS
-- Check the database version, if >= 10g, use BINARY_DOUBLE data type,
-- else use NUMBER data type
BINARY_DOUBLE;
my_pi my_real; my_e my_real;
END my_pkg;
```



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de Procedimientos DBMS_PREPROCESSOR para Imprimir o Recuperar Texto de Origen

Los subprogramas DBMS_PREPROCESSOR imprimen o recuperan el texto de origen postprocesado de una unidad PL/SQL después de procesar las directivas de compilación condicional. Este texto postprocesado es el origen real utilizado para compilar una unidad PL/SQL válida. El ejemplo de la diapositiva muestra cómo imprimir la forma postprocesada de `my_pkg` mediante el procedimiento `PRINT_POST_PROCESSED_SOURCE`.

Cuando se compila `my_pkg` en una base de datos Oracle Database 10g o versión posterior con la cuenta HR, la salida resultante se muestra en la diapositiva anterior.

`PRINT_POST_PROCESSED_SOURCE` elimina texto no seleccionado. Se eliminan las líneas de código que no se incluyen en el texto postprocesado. Los argumentos del procedimiento `PRINT_POST_PROCESSED_SOURCE` son: tipo de objeto, nombre del esquema (utilizando la cuenta de estudiante ORA61) y nombre del objeto.

Nota: para obtener más información sobre el paquete DBMS_PREPROCESSOR, consulte *Oracle Database PL/SQL Packages and Types Reference* (Referencia de Tipos y Paquetes PL/SQL de Oracle Database).

Agenda de la Lección

- Uso de la compilación condicional
- Ocultación del código PL/SQL

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

¿Qué es la Ocultación?

- La ocultación (o encapsulamiento) de una unidad PL/SQL es el proceso de ocultar el código fuente PL/SQL.
- El encapsulamiento se puede realizar con la utilidad Wrap y los subprogramas DBMS_DDL.
- La utilidad Wrap se ejecuta desde la línea de comando y procesa un archivo SQL de entrada, como un script de instalación de SQL*Plus.
- Los subprogramas DBMS_DDL encapsulan una única unidad PL/SQL, como un único comando CREATE PROCEDURE, que se ha generado dinámicamente.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Nota

Para obtener más información sobre la ocultación, consulte *Oracle Database PL/SQL Language Reference* (Referencia del Lenguaje PL/SQL de Oracle Database).

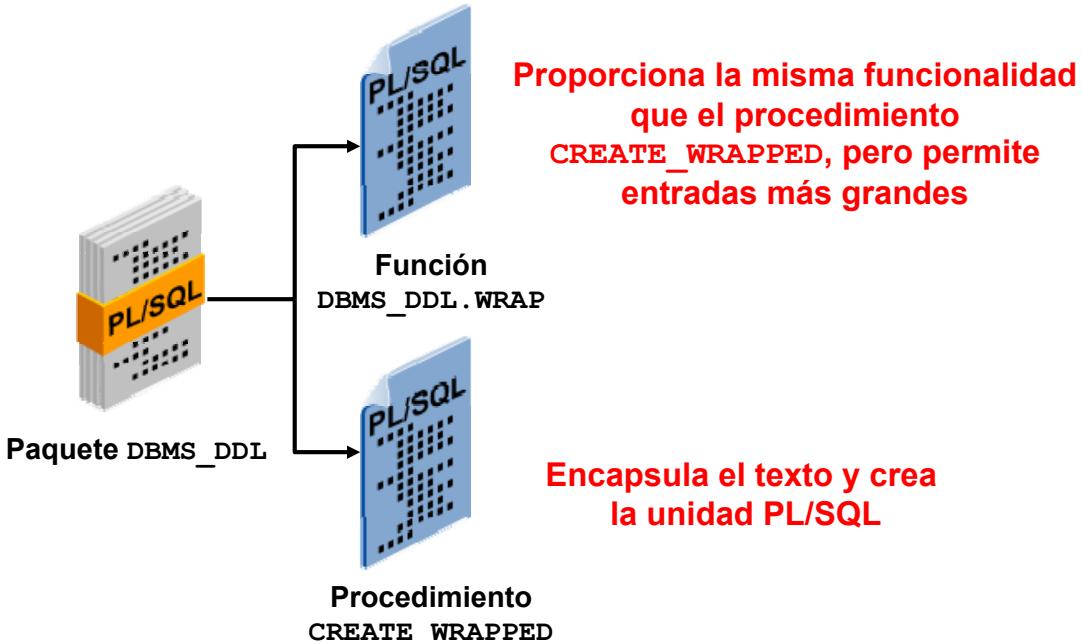
Ventajas de la Ocultación

- Impide que otros usuarios vean el código fuente.
- El código fuente no está visible mediante las vistas del diccionario de datos `USER_SOURCE`, `ALL_SOURCE` o `DBA_SOURCE`.
- SQL*Plus puede procesar los archivos de origen ocultos.
- Las utilidades Import y Export aceptan archivos encapsulados.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Novedades de la Ocultación Dinámica desde Oracle 10g



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Novedades de la Ocultación Dinámica desde Oracle 10g

Procedimiento `CREATE_WWRAPPED`

Toma como entrada una única sentencia `CREATE OR REPLACE` que especifica la creación de una especificación de paquete PL/SQL, cuerpo del paquete, función, procedimiento, especificación del tipo o cuerpo del tipo, genera una sentencia `CREATE OR REPLACE` con el texto de origen de PL/SQL oculto y ejecuta la sentencia generada.

Función `WRAP`

Toma como entrada una sentencia `CREATE OR REPLACE` que especifica la creación de una especificación de paquete PL/SQL, cuerpo del paquete, función, procedimiento, especificación del tipo o cuerpo del tipo y devuelve una sentencia `CREATE OR REPLACE`, donde el texto de la unidad PL/SQL se ha ocultado.

Código PL/SQL No Oculto: Ejemplo

```
SET SERVEROUTPUT ON
BEGIN -- The ALL_SOURCE view family shows source code
  EXECUTE IMMEDIATE '
    CREATE OR REPLACE PROCEDURE P1 IS
      BEGIN
        DBMS_OUTPUT.PUT_LINE (''I am not wrapped'');
      END P1;
    ';
END;
/
CALL p1();
```

```
anonymous block completed
CALL p1() succeeded.
I'm not wrapped
```

```
SELECT text FROM user_source
WHERE name = 'P1' ORDER BY line;
```

TEXT
1 PROCEDURE P1 IS
2 BEGIN
3 DBMS_OUTPUT.PUT_LINE ('I am not wrapped');
4 END P1;

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Código PL/SQL No Oculto: Ejemplo

En el primer ejemplo de la diapositiva, la sentencia EXECUTE IMMEDIATE se utiliza para crear el procedimiento P1. El código del procedimiento creado no está encapsulado. El código no está oculto cuando utiliza cualquiera de las vistas de la familia de vistas ALL_SOURCE para visualizar el código del procedimiento, como se muestra en la diapositiva.

Código PL/SQL Oculto: Ejemplo

```
BEGIN -- ALL_SOURCE view family obfuscates source code
DBMS_DDL.CREATE_WRAPPED (
    CREATE OR REPLACE PROCEDURE P1 IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE (''I am wrapped now'');
    END P1;
    );
END;
/
CALL p1();
```

```
anonymous block completed
call pl() succeeded.
I am wrapped now
```

```
SELECT text FROM user_source
WHERE name = 'P1' ORDER BY line;
```

```
TEXT
-----
PROCEDURE P1 wrapped
a00000
b2
abcd
```

...

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Código PL/SQL Oculto: Ejemplo

En el ejemplo de la diapositiva, el procedimiento del paquete DBMS_DDL.CREATE_WRAPPED se utiliza para crear el procedimiento P1.

El código está oculto cuando utiliza cualquiera de las vistas de la familia de vistas ALL_SOURCE para visualizar el código del procedimiento, como se muestra en la siguiente página. Al comprobar las vistas *_SOURCE, el origen se encapsula, u oculta, para que otros usuarios no puedan ver los detalles de código, como se muestra en la salida del comando de la diapositiva.

Ocultación Dinámica: Ejemplo

```

SET SERVEROUTPUT ON

DECLARE
  c_code CONSTANT VARCHAR2 (32767) := 
    ' CREATE OR REPLACE PROCEDURE new_proc AS
      v_VDATE DATE;
    BEGIN
      v_VDATE := SYSDATE;
      DBMS_OUTPUT.PUT_LINE(v_VDATE) ;
    END; ' ;
BEGIN
  DBMS_DDL.CREATE_WRAPPED (c_CODE);
END;
/

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Ocultación Dinámica: Ejemplo

En el ejemplo de la diapositiva se muestra la creación de un procedimiento oculto dinámicamente denominado

NEW_PROC. Para verificar que el código de NEW_PROC está oculto, puede consultar las vistas del diccionario DBA|ALL|USER_SOURCE, como se muestra a continuación:

```

SELECT text FROM user_source
WHERE name = 'NEW_PROC';

```

```

TEXT
-----
PROCEDURE new_proc wrapped
a0000000

```

...

```

7
71 9e
hBWmpGeSsd58b4jCP3/0d04rof0wg5nnm7+fMr2ywFyFDGLQlhaXriu4dCuPCWnnx1J0U1xp
pvc8nsr7Seq/riQvHRsXAQovdoh0K6ZvM1Kbskr+KLK957KzHQYwLK4k6rJLCS5EyJ7qJB/2
RDmm3j79Uw==


```

1 rows selected

Utilidad Encapsuladora PL/SQL

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

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Encapsulador PL/SQL

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

El código encapsulado, 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 reiniciar para agregar 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 encapsulados.

Ejecución de la Utilidad Encapsuladora

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

- No utilice espacios delante ni detrás del signo igual.
- Se necesita el argumento `INAME`.
- La extensión por defecto del archivo de entrada es `.sql`, a menos que se especifique con 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=demo_04_hello.sql
WRAP INAME=demo_04_hello
WRAP INAME=demo_04_hello.sql ONAME=demo_04_hello.plb
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Ejecución del Encapsulador

El encapsulador es un ejecutable del sistema operativo denominado WRAP. *Se trata de código de sólo lenguaje de compilación.*

Para ejecutar el encapsulador, introduzca el siguiente comando en la petición de datos del sistema operativo:

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

Cada uno de los ejemplos mostrados en la diapositiva toma un archivo denominado `demo_04_hello.sql` como entrada y crea un archivo de salida denominado `demo_04_hello.plb`.

Después de crear el archivo encapsulado, ejecute el archivo `.plb` de SQL*Plus para compilar y almacenar la versión encapsulada del código fuente, tal y como ejecutaría los archivos de script 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`.
- El archivo de entrada 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 Encapsulamiento

```
-- Original PL/SQL source code in input file:  
  
CREATE PACKAGE banking IS  
    min_bal := 100;  
    no_funds EXCEPTION;  
    ...  
END banking;  
/
```

```
-- Wrapped code in output file:  
  
CREATE PACKAGE banking  
    wrapped  
012abc463e ...  
/
```



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Resultados del Encapsulamiento

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

El archivo de entrada puede contener cualquier combinación de sentencias SQL. Sin embargo, el encapsulador PL/SQL sólo encapsula 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 Encapsulamiento

- Debe encapsular sólo el cuerpo del paquete, no su especificación.
- El encapsulador puede detectar errores sintácticos, no semánticos.
- El archivo de salida no se debe editar. Puede mantener el código fuente original y volverlo a encapsular como sea necesario.
- Para garantizar que todas las partes importantes del código fuente están ocultas, visualice el archivo encapsulado en un editor de texto antes de distribuirlo.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Instrucciones para el Encapsulamiento

Entre estas instrucciones se incluyen:

- Al encapsular el paquete o el tipo de objeto, encapsule sólo el cuerpo, no la especificación. Por lo tanto, proporcione a otros desarrolladores la información que necesiten para utilizar el paquete sin exponer su implantación.
- Si su archivo de entrada contiene errores sintácticos, el encapsulador PL/SQL los detecta y notifica. Sin embargo, el encapsulador no puede detectar errores semánticos porque no resuelve referencias externas. Por ejemplo, el encapsulador 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 encapsulador (archivo .plb).

- Puesto que no se puede leer su contenido, el archivo de salida no se debe editar. Para cambiar un objeto encapsulado, necesita modificar el código fuente original y volver a encapsular el código.

Paquete DBMS_DDL frente a utilidad Wrap

Funcionalidad	DBMS_DDL	Utilidad Wrap
Ocultación de código	Sí	Sí
Ocultación dinámica	Sí	No
Ocultación de varios programas a la vez	No	Sí

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

DBMS_DDL frente a Utilidad Wrap

La utilidad Wrap y el paquete DBMS_DDL tienen distintos usos:

La utilidad Wrap resulta útil para ocultar varios programas con una sola ejecución. Básicamente, se puede ajustar una aplicación completa. Sin embargo, la utilidad Wrap no se puede utilizar para ocultar dinámicamente código generado en tiempo de ejecución. La utilidad Wrap procesa un archivo SQL de entrada y oculta sólo las unidades PL/SQL del archivo, como:

- Especificación y cuerpo del paquete
- Función y procedimiento
- Especificación y cuerpo del tipo

La utilidad Wrap no oculta el contenido PL/SQL en:

- Bloques anónimos
- Disparadores
- Código que no sea PL/SQL

El paquete DBMS_DDL está destinado a ocultar una unidad de programa generada dinámicamente desde otra unidad de programa. Los métodos del paquete DBMS_DDL no pueden ocultar varias unidades de programa con una ejecución. Cada ejecución de estos métodos sólo acepta una sentencia CREATE OR REPLACE a la vez como argumento.

Prueba

La compilación condicional le permite personalizar la funcionalidad en una aplicación PL/SQL sin eliminar ningún código fuente.

1. Verdadero
2. Falso



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: 1

Compilación Condicional

La compilación condicional le permite personalizar la funcionalidad de una aplicación PL/SQL sin eliminar ningún código fuente.

Utilice la última funcionalidad con la última versión de la base de datos o desactive las nuevas funciones para ejecutar la aplicación con versiones anteriores de la base de datos.

Active la funcionalidad de depuración o rastreo en el entorno de desarrollo y oculte dicha funcionalidad en la aplicación mientras se ejecuta en una ubicación de producción.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Describir y utilizar la compilación condicional
- Ocultar el código fuente PL/SQL mediante ocultación dinámica y la utilidad Wrap

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Resumen

En esta lección se ha presentado la compilación condicional y la ocultación (o encapsulamiento) de código PL/SQL.

Práctica 11: Visión General

En esta práctica se abordan los siguientes temas:

- Creación de un paquete y un procedimiento en los que se utiliza la compilación condicional
- Uso del paquete adecuado para recuperar el texto de origen postprocesado de la unidad PL/SQL
- Ocultación de parte del código PL/SQL



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica 11: Visión General

En esta práctica, creará un paquete y un procedimiento en los que se utilice la compilación condicional. Además, utilizará el paquete adecuado para recuperar el texto de origen post-procesado de la unidad PL/SQL. También ocultará parte del código PL/SQL.

12

Gestión de Dependencias

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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
- Gestionar dependencias de procedimiento



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos de la Lección

Esta lección ofrece una introducción a las dependencias de objetos y a la recompilación implícita y explícita de objetos no válidos.

Visión General de Dependencias de Objetos de Esquema

Tipo de Objeto	Puede Ser Dependiente o al que Se Hace Referencia
Cuerpo del paquete	Sólo dependiente
Especificación del paquete	Ambos
Secuencia	Sólo se le hace referencia
Subprograma	Ambos
Sinónimo	Ambos
Tabla	Ambos
Disparador	Ambos
Objeto definido por el usuario	Ambos
Recopilación definida por el usuario	Ambos
Vista	Ambos

ORACLE

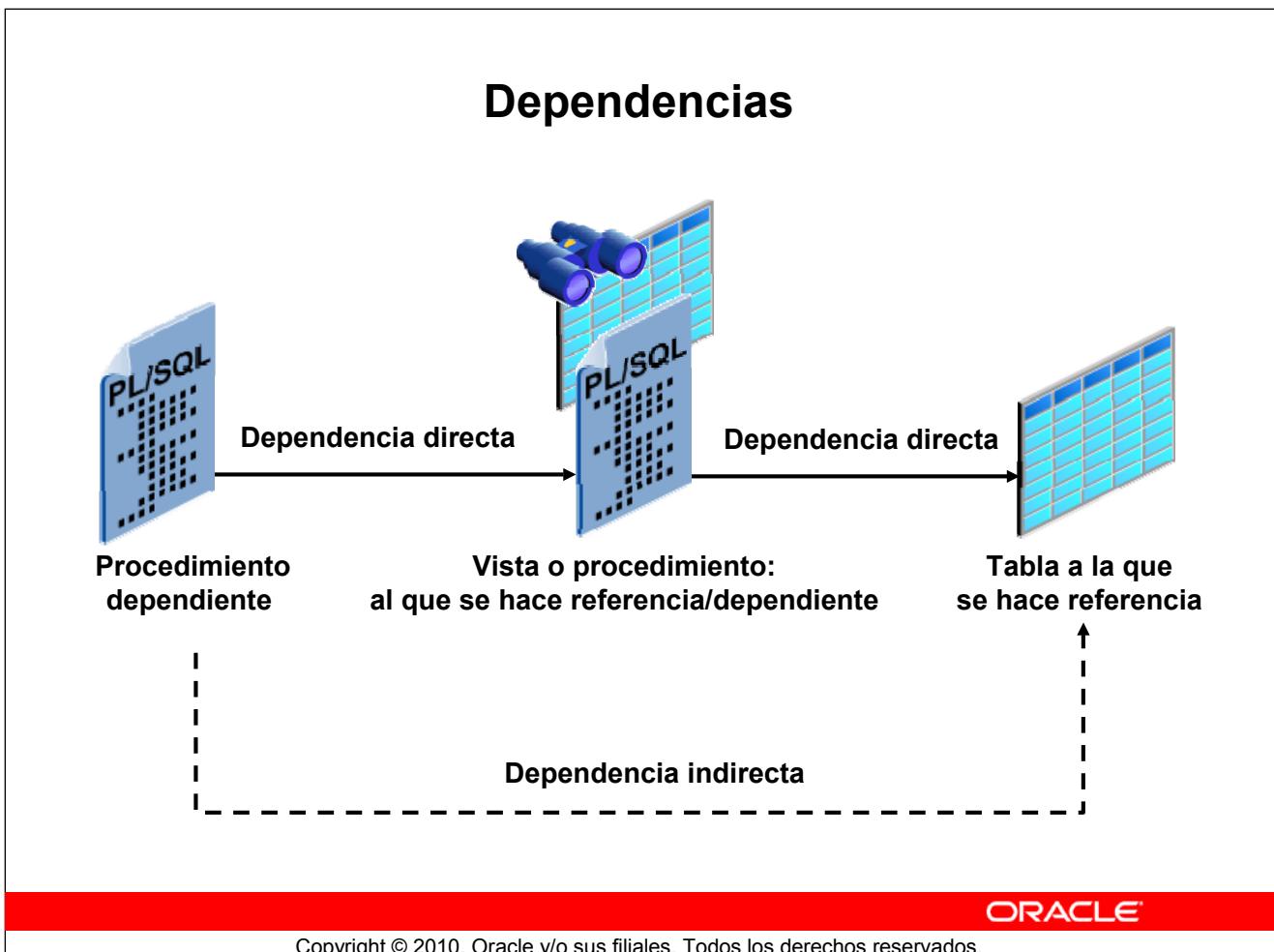
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Objetos Dependientes y a los que Se Hace Referencia

Algunos tipos de objetos de esquema hacen referencia a otros objetos en sus definiciones. Por ejemplo, una vista definida por una consulta que hace referencia a tablas o a otras vistas, así como el cuerpo de un subprograma, pueden incluir sentencias SQL que hagan referencia a otros objetos. Si la definición del objeto A hace referencia al objeto B, A es un objeto dependiente (con respecto a B) y B es un objeto al que se hace referencia (con respecto a A).

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`.
- Si el estado de un objeto de esquema es `VALID`, el objeto se ha compilado y se puede utilizar inmediatamente cuando se haga referencia al mismo.
- Si el estado de un objeto de esquema es `INVALID`, el objeto se debe compilar para que se pueda utilizar.

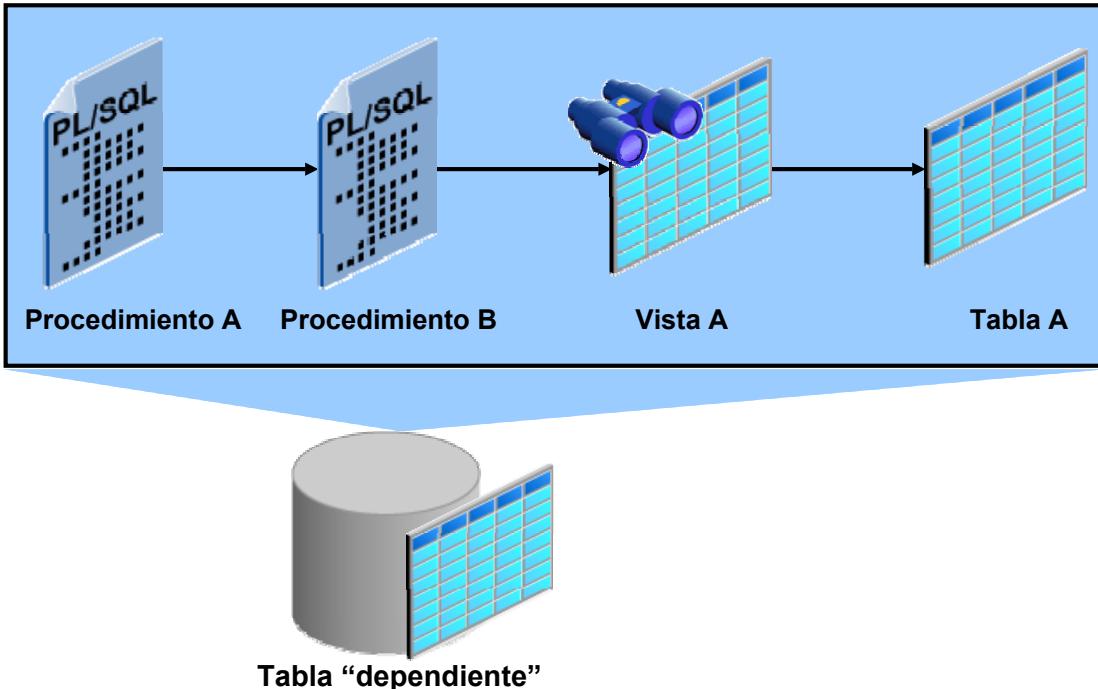


Objetos Dependientes y a los que Se Hace Referencia (continuación)

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 Directas



ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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” interna de la base de datos. Cuando se modifica un objeto al que se hace referencia, los objetos dependientes a veces se invalidan. La próxima vez que se llame a un objeto invalidado, el servidor de Oracle lo recompilará de forma automática.

Si modifica la definición de un objeto al que se hace referencia, los objetos dependientes pueden o no seguir funcionando sin errores, según el tipo de modificación. Por ejemplo, si borra una tabla, no se puede utilizar ninguna vista basada en la tabla borrada.

A partir de Oracle Database 10g, el comando `CREATE OR REPLACE SYNONYM` se ha mejorado para minimizar las invalidaciones para vistas y unidades de programa PL/SQL dependientes que hacen referencia a él. Esto se trata más adelante en esta lección.

A partir de Oracle Database 11g, el seguimiento de las dependencias se realiza a nivel de elemento dentro de una unidad. Esto se denomina dependencia detallada. Las dependencias detalladas se tratan más adelante en esta lección.

Consulta de Dependencias Directas de Objeto: mediante la Vista USER_DEPENDENCIES

Name	Null	Type
NAME	NOT NULL	VARCHAR2(30)
TYPE		VARCHAR2(17)
REFERENCED_OWNER		VARCHAR2(30)
REFERENCED_NAME		VARCHAR2(64)
REFERENCED_TYPE		VARCHAR2(17)
REFERENCED_LINK_NAME		VARCHAR2(128)
SCHEMAD		NUMBER
DEPENDENCY_TYPE		VARCHAR2(4)

8 rows selected

```
SELECT name, type, referenced_name, referenced_type
FROM user_dependencies
WHERE referenced_name IN ('EMPLOYEES', 'EMP_VW');
```

NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE
1 QUERY_EMP	PROCEDURE	EMPLOYEES	TABLE
2 DML_CALL_SQL	FUNCTION	EMPLOYEES	TABLE
3 QUERY_CALL_SQL	FUNCTION	EMPLOYEES	TABLE
4 COMM_PKG	PACKAGE BODY	EMPLOYEES	TABLE
5 CURS_PKG	PACKAGE BODY	EMPLOYEES	TABLE
6 EMP_PKG	PACKAGE	EMPLOYEES	TABLE
• • 7 EMP_PKG	PACKAGE BODY	EMPLOYEES	TABLE

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Consulta de Dependencias Directas de Objeto: mediante la Vista USER_DEPENDENCIES

Puede determinar 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.

Las vistas ALL_DEPENDENCIES y DBA_DEPENDENCIES contienen la columna adicional OWNER, que hace referencia al propietario del objeto.

Columnas de la Vista del Diccionario de Datos USER_DEPENDENCIES

Las columnas de la vista del diccionario de datos USER_DEPENDENCIES son las siguientes:

- 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

Consulta del Estado de un Objeto

Todos los objetos de base de datos tienen uno de los siguientes valores de estado:

Estado	Descripción
VALID	El objeto se ha compilado correctamente, mediante la definición actual del diccionario de datos.
COMPILED WITH ERRORS	El intento más reciente de compilación del objeto ha producido errores.
INVALID	El objeto se ha marcado como no válido porque un objeto al que hace referencia ha cambiado. (Sólo un objeto dependiente puede ser no válido.)
UNAUTHORIZED	Se ha revocado un privilegio de acceso en un objeto al que se hace referencia. (Sólo un objeto dependiente puede ser no autorizado.)

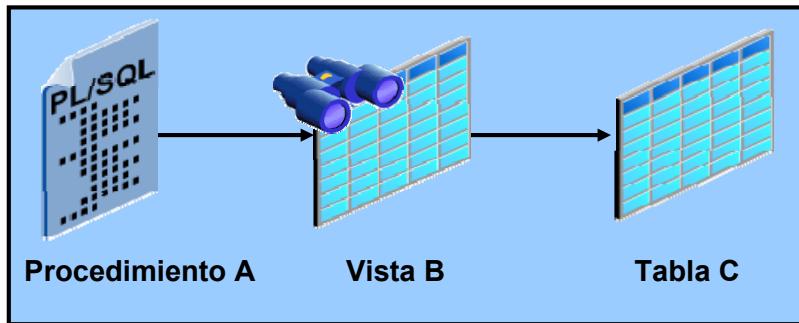
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Consulta del Estado de un Objeto

Todos los objetos de base de datos tienen uno de los valores de estado mostrados en la tabla de la diapositiva.

Nota: las vistas estáticas del diccionario de datos USER_OBJECTS, ALL_OBJECTS y DBA_OBJECTS no distinguen entre Compiled with errors, Invalid y Unauthorized; en su lugar, se describen todas como INVALID.

Invalidación de Objetos Dependientes



- El procedimiento A es un dependiente directo de la vista B. La vista B es un dependiente directo de la tabla C. El procedimiento A es un dependiente indirecto de la tabla C.
- Los dependientes directos se invalidan sólo mediante cambios en el objeto al que hacen referencia que les afectan.
- Los dependientes indirectos se pueden invalidar mediante cambios en el objeto al que hacen referencia que no les afectan.

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Invalidación de Objetos Dependientes

Si el objeto A depende del objeto B, que depende del objeto C, A es un dependiente directo de B, B es un dependiente directo de C y A es un dependiente indirecto de C.

En Oracle Database 11g, los dependientes directos se invalidan sólo mediante cambios en el objeto al que hacen referencia que les afectan (cambios en la firma del objeto al que hacen referencia).

Los dependientes indirectos se pueden invalidar mediante cambios en el objeto al que hacen referencia que no les afectan: si un cambio en la tabla C invalida la vista B, también invalida el procedimiento A (y todas las dependencias directo e indirectas de la vista B). Esto se denomina invalidación en cascada.

Si se modifica la estructura de la tabla en la que se basa una vista, cuando describa la vista con el comando SQL*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; 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.

Cambio del Objeto de Esquema que Invalida Algunos Dependientes: Ejemplo

```
CREATE VIEW commissioned AS
SELECT first_name, last_name, commission_pct FROM employees
WHERE commission_pct > 0.00;
```

```
CREATE VIEW six_figure_salary AS
SELECT * FROM employees
WHERE salary >= 100000;
```

```
SELECT object_name, status
FROM user_objects
WHERE object_type = 'VIEW';
```

Results:		
	OBJECT_NAME	STATUS
1	EMP_DETAILS	VALID
2	COMMISSIONED	VALID
3	SIX FIGURE SALARY	VALID
4	EMP_DETAILS_VIEW	VALID



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Cambio del Objeto de Esquema que Invalida Algunos Dependientes: Ejemplo

En la diapositiva se muestra un ejemplo de un cambio del objeto de esquema que invalida algunos dependientes, pero no otros. Las dos vistas recién creadas se basan en la tabla EMPLOYEES del esquema HR. El estado de las vistas recién creadas es VALID.

Cambio del Objeto de Esquema que Invalida Algunos Dependientes: Ejemplo

```
ALTER TABLE employees MODIFY email VARCHAR2(50);

SELECT object_name, status
FROM user_objects
WHERE object_type = 'VIEW';
```

Results:	
OBJECT_NAME	STATUS
1 EMP_DETAILS	VALID
2 COMMISSIONED	VALID
3 SIX FIGURE_SALARY	INVALID
4 EMP_DETAILS_VIEW	VALID

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Cambio del Objeto de Esquema que Invalida Algunos Dependientes: Ejemplo (continuación)

Supongamos que determina que la columna EMAIL de la tabla EMPLOYEES tiene que aumentarse de longitud de 25 a 50 y modifica la tabla como se muestra en la diapositiva anterior.

Puesto que la vista COMMISSIONED no incluye la columna EMAIL en su lista de selección, no se invalida. Sin embargo, la vista SIXFIGURES se invalida porque se han seleccionado todas las columnas de la tabla.

Visualización de Dependencias Directas e Indirectas

- Ejecute el script `utldtree.sql` para crear los objetos que permiten mostrar las dependencias directas e indirectas.

```
@/home/oracle/labs/plpu/labs/utldtree.sql
```

- Ejecute el procedimiento `DEPTREE_FILL`.

```
EXECUTE deptree_fill('TABLE', 'ORA61', 'EMPLOYEES')
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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` e `IDEPTREE` que proporciona Oracle.

Ejemplo

- Asegúrese de que se ha ejecutado el script `utldtree.sql`. Este script está ubicado en la carpeta `$ORACLE_HOME/labs/plpu/labs`. Puede ejecutar el script de la siguiente forma:
`@?/labs/plpu/labs/utldtree.sql`

Nota: en esta clase, este script está en la carpeta `labs` de los archivos de clase. El ejemplo de código anterior utiliza la cuenta de estudiante `ORA61`. (Esto se aplica a un entorno Linux. Si no se encuentra el archivo, localícelo en el subdirectorio `labs`.)

- Rellene la tabla `DEPTREE_TEMP TAB` con información para un objeto determinado al que se hace referencia, llamando al procedimiento `DEPTREE_FILL`. Hay tres parámetros para este procedimiento:

<code>object_type</code>	Type of the referenced object
<code>object_owner</code>	Schema of the referenced object
<code>object_name</code>	Name of the referenced object

Visualización de Dependencias mediante la 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	SECURE_EMPLOYEES
1	TRIGGER	UPDATE_JOB_HISTORY
1	PROCEDURE	RAISE_SALARY
2	PROCEDURE	PROCESS_EMPLOYEES
1	PROCEDURE	QUERY_EMP
1	PROCEDURE	PROCESS_EMPLOYEES
1	FUNCTION	DML_CALL_SQL
1	FUNCTION	QUERY_CALL_SQL
1	PACKAGE BODY	COMM_PKG
1	PACKAGE BODY	CURS_PKG
1	PACKAGE	EMP_PKG
2	PACKAGE BODY	EMP_PKG
1	PACKAGE BODY	EMP_PKG
1	PROCEDURE	SAL_STATUS

...



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Visualización de Dependencias mediante la Vista DEPTREE

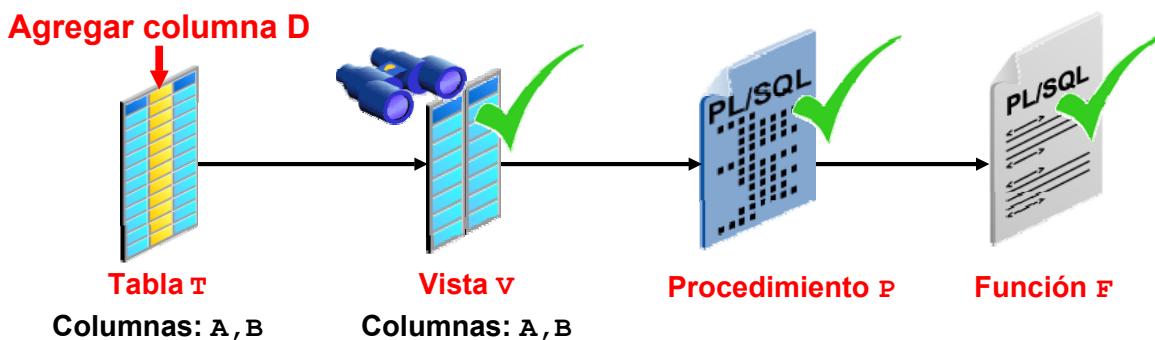
Puede visualizar una representación tabular de todos los objetos dependientes consultando la vista DEPTREE. Puede visualizar una representación sangrada de la misma información consultando la vista IDEPTREE, que consta de una sola columna denominada DEPENDENCIES:

```
SELECT *
FROM    ideptree;
```

DEPENDENCIES
TRIGGER ORA61.SECURE_EMPLOYEES
PROCEDURE ORA61.UPDATE_SALARY
TRIGGER ORA61.AUDIT_EMP_VALUES
PROCEDURE ORA61.EMP_LIST
PROCEDURE ORA61.PROCESS_EMPLOYEES
PACKAGE BODY ORA61.CURS_PKG
PACKAGE BODY ORA61.EMP_PKG
VIEW ORA61.EMP_DETAILS
TRIGGER ORA61.NEW_EMP_DEPT
TRIGGER ORA61.UPDATE_JOB_HISTORY
PACKAGE BODY ORA61.EMP_PKG
FUNCTION ORA61.EMP_HIRE_DATE
PROCEDURE ORA61.SAL_STATUS
TRIGGER ORA61.DERIVE_COMMISSION_PCT
TRIGGER ORA61.CHECK_SALARY
PROCEDURE ORA61.PROCESS_EMPLOYEES
PROCEDURE ORA61.QUERY_EMP
TRIGGER ORA61.RESTRICT_SALARY
VIEW ORA61.COMMISSIONED

Metadatos de Dependencia más Precisos en Oracle Database 11g

- Antes de 11g, al agregar la columna D a la tabla T, se invalidaban los objetos dependientes.
- Oracle Database 11g registra la gestión de dependencias detalladas adicionales:
 - Al agregar la columna D a la tabla T, la vista V no se ve afectada ni se invalidan los objetos dependientes.



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Dependencias Detalladas

A partir de Oracle Database 11g, tiene acceso a registros que describen con más precisión los metadatos de las dependencias. Es lo que se denomina dependencia detallada y permite ver qué objetos dependientes no están invalidados sin requisitos lógicos.

En versiones anteriores de la base de datos Oracle, se registraban los metadatos de dependencias (por ejemplo, que la unidad PL/SQL P depende de la unidad PL/SQL F o que la vista V depende de la tabla T) con la precisión del objeto completo. Esto significa que, en ocasiones, los objetos dependientes se invalidan sin requisitos lógicos. Por ejemplo, si la vista V depende en exclusiva de las columnas A y B de la tabla T y se agrega la columna D a la tabla T, la validez de la vista V no se ve afectada lógicamente. Sin embargo, antes de Oracle Database Versión 11.1, la vistas V se invalidaba al agregar la columna D a la tabla T. Con Oracle Database Versión 11.1, al agregar la columna D a la tabla T, no se invalida la vista V. De igual modo, si el procedimiento P depende sólo de los elementos E1 y E2 de un paquete, al agregar el elemento E99 al paquete, no se invalida el procedimiento P.

Al reducir la invalidación de los objetos dependientes en respuesta a los cambios realizados en los objetos de los que dependen, aumenta la disponibilidad de la aplicación tanto en el entorno de desarrollo como durante su actualización en línea.

Gestión de Dependencias Detalladas

- En Oracle Database 11g, el seguimiento de las dependencias se realiza a nivel de *elemento dentro de una unidad*.
- El seguimiento de dependencias basado en elementos abarca lo siguiente:
 - Dependencia de una vista de tabla única de su tabla base
 - Dependencia de una unidad de programa PL/SQL (especificación del paquete, cuerpo del paquete o subprograma) de lo siguiente:
 - Otras unidades de programa PL/SQL
 - Tablas
 - Vistas



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Gestión de Dependencias Detalladas: Ejemplo 1

```
CREATE TABLE t2 (col_a NUMBER, col_b NUMBER, col_c NUMBER);
CREATE VIEW v AS SELECT col_a, col_b FROM t2;
```

```
SELECT ud.name, ud.type, ud.referenced_name,
       ud.referenced_type, uo.status
  FROM user_dependencies ud, user_objects uo
 WHERE ud.name = uo.object_name AND ud.name = 'V';
```

Results:					
	NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE	STATUS
1	V	VIEW	T2	TABLE	VALID

```
ALTER TABLE t2 ADD (col_d VARCHAR2(20));
```

```
SELECT ud.name, ud.type, ud.referenced_name,
       ud.referenced_type, uo.status
  FROM user_dependencies ud, user_objects uo
 WHERE ud.name = uo.object_name AND ud.name = 'V';
```

Results:					
	NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE	STATUS
1	V	VIEW	T2	TABLE	VALID

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Gestión de Dependencias Detalladas: Ejemplo 1

Ejemplo de Dependencia de una Vista de Tabla Única de su Tabla Base

En el primer ejemplo de la diapositiva, se crea la tabla T2 con tres columnas: COL_A, COL_B y COL_C. Se crea una vista denominada V basada en las columnas COL_A y COL_B de la tabla T2. Se consultan las vistas del diccionario y la vista V depende de la tabla T y su estado es válido.

En el tercer ejemplo, se modifica la tabla T2. Se agrega una nueva columna denominada COL_D. En las vistas del diccionario sigue apareciendo que la vista V es dependiente, porque el seguimiento de dependencias basado en elementos detecta que las columnas COL_A y COL_B no están modificadas y, por lo tanto, no hace falta invalidar la vista.

Gestión de Dependencias Detalladas: Ejemplo 1

```
ALTER TABLE t2 MODIFY (col_a VARCHAR2(20));
SELECT ud.name, ud.referenced_name, ud.referenced_type,
       uo.status
  FROM user_dependencies ud, user_objects uo
 WHERE ud.name = uo.object_name AND ud.name = 'V';
```

Results:

	NAME	REFERENCED_NAME	REFERENCED_TYPE	STATUS
1	V	T2	TABLE	INVALID

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Gestión de Dependencias Detalladas: Ejemplo 1 (continuación)

En el ejemplo de la diapositiva, la vista se invalida porque su elemento (COL_A) se ha modificado en la tabla de la que depende la vista.

Gestión de Dependencias Detalladas: Ejemplo 2

```

CREATE PACKAGE pkg IS
  PROCEDURE proc_1;
END pkg;
/
CREATE OR REPLACE PROCEDURE p IS
BEGIN
  pkg.proc_1();
END p;
/
CREATE OR REPLACE PACKAGE pkg
IS
  PROCEDURE proc_1;
  PROCEDURE unheard_of;
END pkg;
/

```

```

PACKAGE pkg Compiled.
PROCEDURE p Compiled.
PACKAGE pkg Compiled.

```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Gestión de Dependencias Detalladas: Ejemplo 2

En el ejemplo de la diapositiva, se crea un paquete denominado PKG que tiene el procedimiento PROC_1 declarado.

Otro procedimiento denominado P llama a PKG.PROC_1.

Se modifica la definición del paquete PKG y se agrega otra subrutina a la declaración del paquete.

Al consultar en la vista del diccionario USER_OBJECTS el estado del procedimiento P, sigue teniendo validez porque no se hace referencia al elemento agregado a la definición de PKG mediante el procedimiento P.

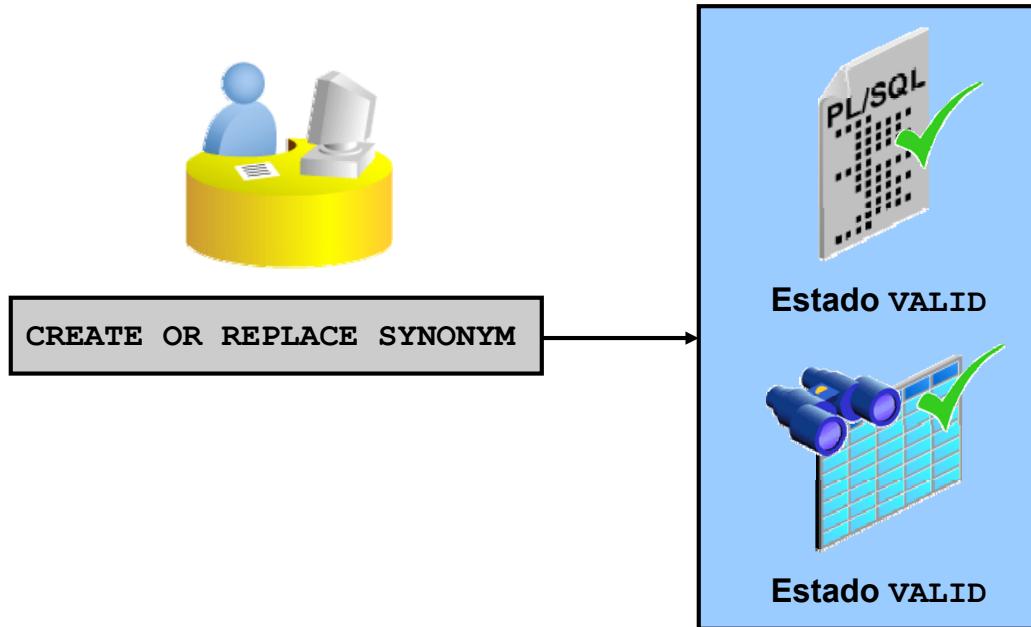
```

SELECT status FROM user_objects
WHERE object_name = 'P';

```

Results:	
	STATUS
1	VALID

Cambios en las Dependencias de Sinónimos



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Cambios en las Dependencias de Sinónimos

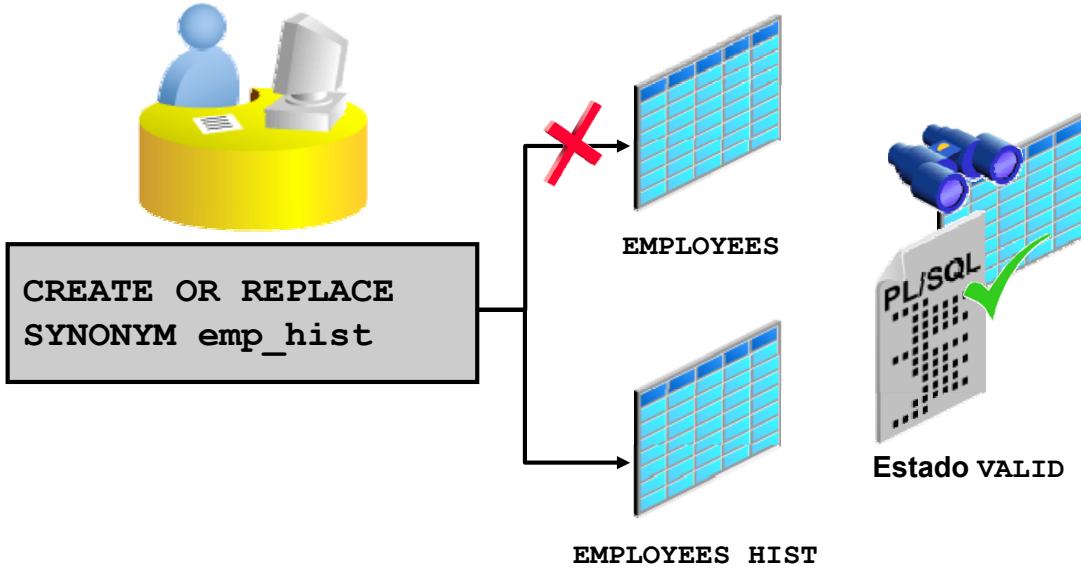
Oracle Database minimiza el tiempo de inactividad durante las actualizaciones de código o fusiones de esquemas.

Cuando se cumplen determinas condiciones en las columnas, privilegios, particiones, etc., las tablas o tipos de objeto que se consideran objetos equivalentes y dependientes ya no se invalidan.

En Oracle Database 10g, el comando CREATE OR REPLACE SYNONYM se ha mejorado para minimizar las invalidaciones para vistas y unidades de programa PL/SQL dependientes que hacen referencia a él. Esto elimina la necesidad de la recompilación que requiere mucho tiempo de las unidades de programa después de volver a definir los sinónimos o durante la ejecución. No tiene que definir ningún parámetro ni emitir ningún comando especial para activar esta funcionalidad; las invalidaciones se minimizan automáticamente.

Nota: esta mejora sólo se aplica a sinónimos que apuntan a tablas.

Mantenimiento de Vistas y Unidades de Programa PL/SQL Válidas



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Mantenimiento de Unidades de Programa PL/SQL Válidas

A partir de Oracle Database 10g Versión 2, puede cambiar la definición de un sinónimo y las unidades de programa PL/SQL dependientes no se invalidan cuando se dan las siguientes condiciones:

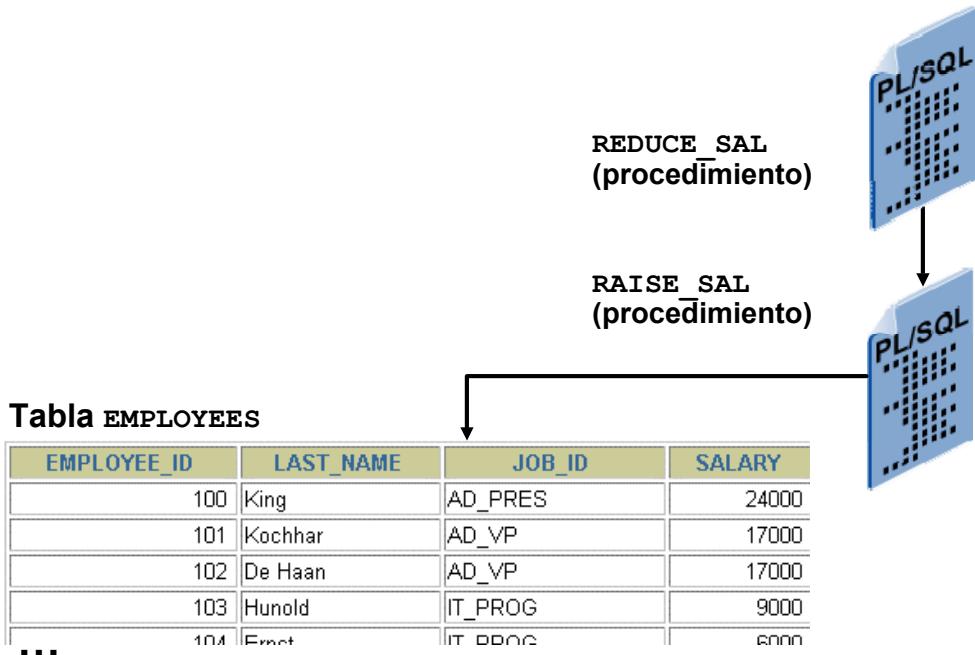
- El orden de las columnas, los nombres de las columnas y los tipos de dato de las columnas de las tablas son idénticos.
- Los privilegios de la tabla a la que se ha hecho referencia recientemente y sus columnas son un superjuego del juego de privilegios de la tabla original. Estos privilegios no se deben derivar de roles solos.
- Los nombres y tipos de particiones y subparticiones son idénticos.
- Las tablas son del mismo tipo de organización.
- Las columnas de tipo de objeto son del mismo tipo.

Mantenimiento de Vistas Válidas

Al igual que con las unidades de programa PL/SQL dependientes, puede cambiar la definición de un sinónimo y las vistas dependientes no se invalidan cuando se dan las condiciones indicadas en el apartado anterior. Además, se debe cumplir lo siguiente para mantener el estado VALID de las vistas dependientes, pero no las unidades de programa PL/SQL dependientes, al volver a definir un sinónimo:

- Las columnas y el orden de las columnas definidos para índices de clave primaria y únicos, restricciones NOT NULL y restricciones de clave primaria y únicas deben ser idénticos.
- La vista dependiente no puede tener ninguna restricción de referencia.

Otro Supuesto de Dependencias Locales



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Otro Supuesto de Dependencias Locales

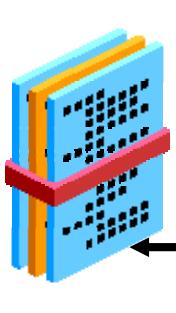
En el ejemplo de la diapositiva se muestra el efecto que produce un cambio en la definición de un procedimiento en la recompilación de un procedimiento dependiente. Si el procedimiento RAISE_SAL actualiza la tabla EMPLOYEES directamente y el procedimiento REDUCE_SAL actualiza la tabla EMPLOYEES

indirectamente mediante RAISE_SAL.

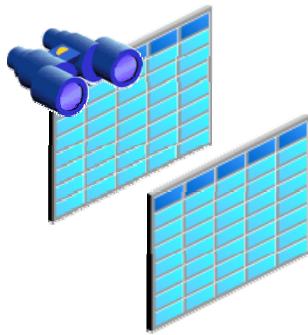
- Si se modifica la lógica interna del procedimiento RAISE_SAL, REDUCE_SAL se recompilará correctamente si RAISE_SAL se ha compilado correctamente.
- Si se eliminan los parámetros formales del procedimiento RAISE_SAL, REDUCE_SAL no se recompilará correctamente.

Instrucciones para Reducir la Invalidación

Para reducir la invalidación de objetos dependientes:



Agregue nuevos elementos al final del paquete



Haga referencia a cada tabla mediante una vista

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Instrucciones para Reducir la Invalidación

Adición de Nuevos Elementos al Final del Paquete

Al agregar nuevos elementos a un paquete, agréguelos al final del paquete. De esta forma, se mantienen los números de ranura y los números de punto de entrada de elementos del paquete de nivel superior existentes, lo que evita su invalidación. Por ejemplo, considere el siguiente paquete:

```
CREATE OR REPLACE PACKAGE pkg1 IS
  FUNCTION get_var RETURN VARCHAR2;
  PROCEDURE set_var (v VARCHAR2);
END;
```

Al agregar un elemento al final de `pkg1`, no se invalidan los dependientes que hacen referencia a `get_var`. Al insertar un elemento entre la función `get_var` y el procedimiento `set_var` se invalidan los dependientes que hacen referencia a la función `set_var`.

Referencia a Cada Tabla mediante una Vista

Haga referencia a las tablas indirectamente, mediante vistas. Esto permite realizar lo siguiente:

- Agregar columnas a la tabla sin invalidar vistas u objetos PL/SQL dependientes
- Modificar o suprimir columnas a las que no hace referencia la vista sin invalidar los objetos dependientes

Revalidación de Objetos

- Un objeto que no es válido cuando se hace referencia a él se debe validar para poder utilizarlo.
- La validación se produce automáticamente cuando se hace referencia a un objeto; no necesita una acción del usuario explícita.
- Si un objeto no es válido, su estado es COMPILED WITH ERRORS, UNAUTHORIZED o INVALID.



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Revalidación de Objetos

El compilador no puede revalidar automáticamente un objeto compilado con errores. El compilador recompila el objeto y, si se recompila sin errores, se revalida; de lo contrario, sigue siendo no válido.

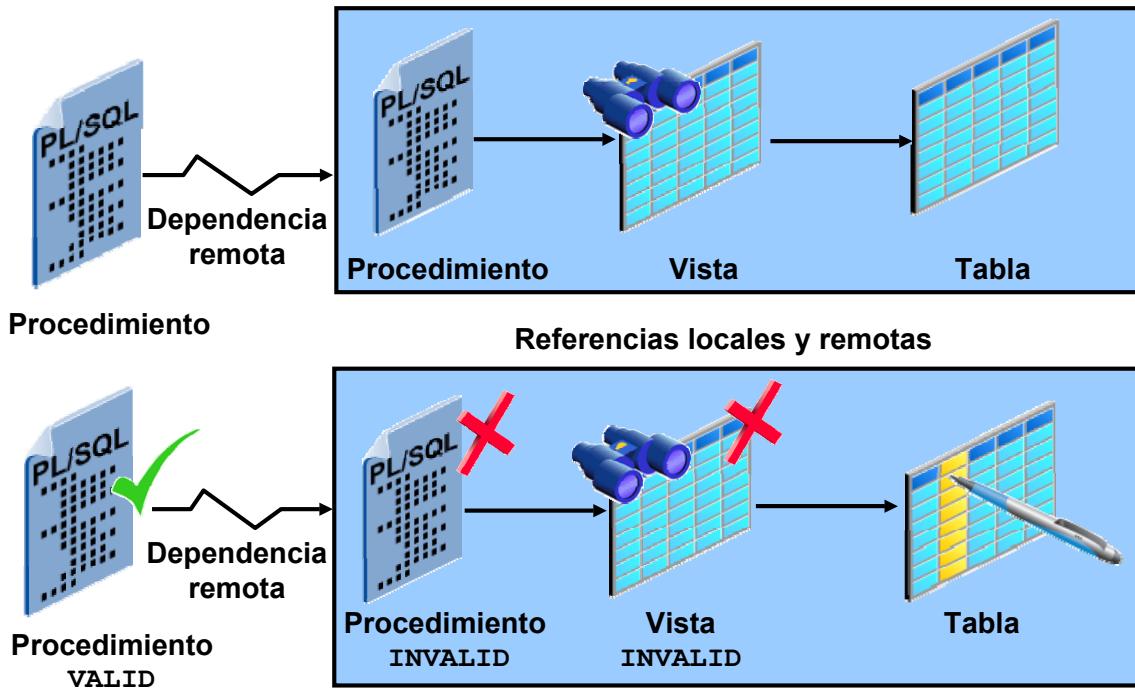
El compilador comprueba si el objeto no autorizado tiene privilegios de acceso a todos los objetos a los que hace referencia. Si es así, el compilador revalida el objeto no autorizado sin recompilarlo. De lo contrario, el compilador emite los mensajes de error adecuados.

El compilador SQL recompila el objeto no válido. Si el objeto se recompila sin errores, se revalida; de lo contrario, sigue siendo no válido.

Para una unidad de programa PL/SQL no válida (procedimiento, función o paquete), el compilador PL/SQL comprueba si cualquier objeto al que hace referencia se ha cambiado de una forma que afecta al objeto no válido.

- Si es así, el compilador recompila el objeto no válido. Si el objeto se recompila sin errores, se revalida; de lo contrario, sigue siendo no válido. De lo contrario, el compilador revalida el objeto no válido sin recompilarlo.
- De lo contrario, el compilador revalida el objeto no válido sin recompilarlo. La revalidación rápida se realiza normalmente en objetos invalidados debido a la invalidación en cascada.

Dependencias Remotas



Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Dependencias Remotas

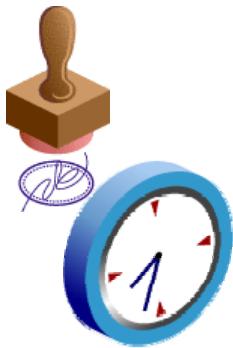
En el caso de dependencias remotas, los objetos están en nodos independientes. El servidor de Oracle no gestiona dependencias entre objetos remotos de esquema distintas de las dependencias de procedimiento local a procedimiento remoto (incluidas las funciones, los paquetes y los disparadores). El procedimiento local almacenado y todos sus objetos dependientes se invalidan pero no se recompilan de forma automática cuando se llaman por primera vez.

Recompilación de Objetos Dependientes: Locales y Remotos

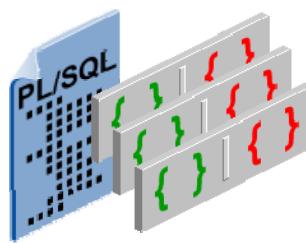
- Verifique la recompilación explícita correcta de los procedimientos remotos dependientes y la recompilación implícita de los procedimientos locales dependientes, comprobando el estado de estos procedimientos en la vista `USER_OBJECTS`.
- Si falla una recompilación implícita automática de los procedimientos 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 © 2010, Oracle y/o sus filiales. 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.

Definición del Parámetro REMOTE_DEPENDENCIES_MODE

- Como parámetro init.ora:

```
REMOTE_DEPENDENCIES_MODE = TIMESTAMP |  
SIGNATURE
```

- A nivel de sistema:

```
ALTER SYSTEM SET REMOTE_DEPENDENCIES_MODE =  
TIMESTAMP | SIGNATURE
```

- A nivel de sesión:

```
ALTER SESSION SET REMOTE_DEPENDENCIES_MODE =  
TIMESTAMP | SIGNATURE
```

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Parámetro REMOTE_DEPENDENCIES_MODE

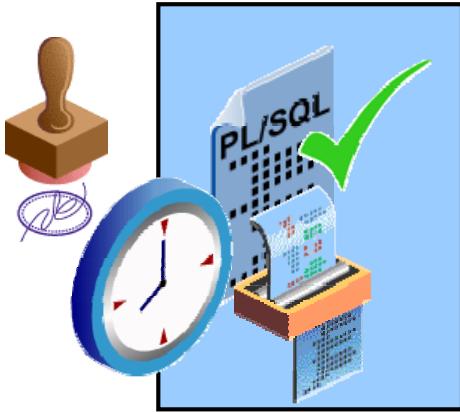
Definición de REMOTE_DEPENDENCIES_MODE

value TIMESTAMP
 SIGNATURE

Especifique el valor del parámetro REMOTE_DEPENDENCIES_MODE utilizando uno de los tres métodos que se describen en la diapositiva.

Nota: el sitio de llamada determina el modelo de dependencia.

El Procedimiento Remoto B Se Compila a las 8:00 a.m.



**Procedimiento remoto B:
Se compila y es VALID
a las 8:00 a.m.**

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Procedimientos Locales que Hacen Referencia a Procedimientos Remotos

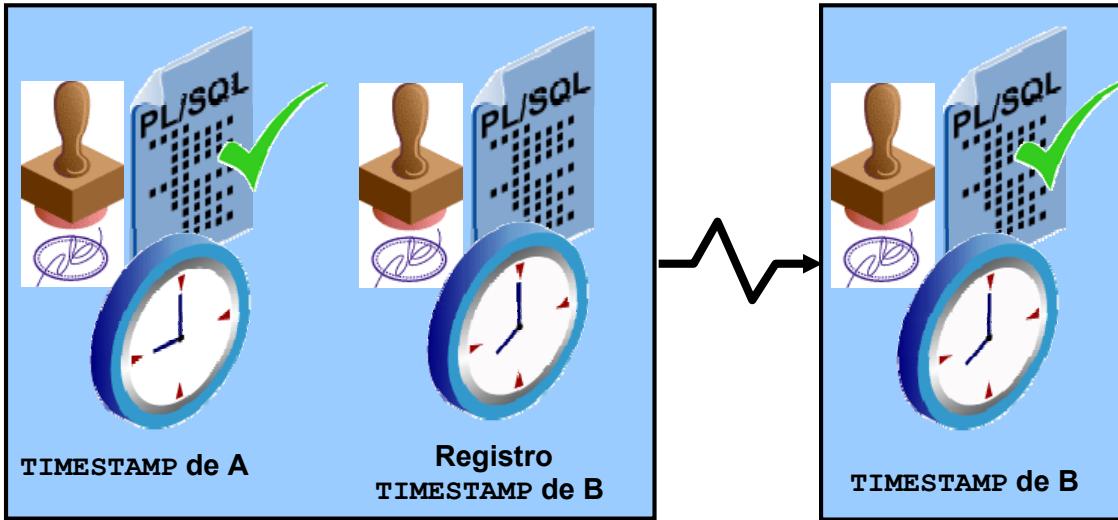
El servidor de Oracle invalidará un procedimiento local que haga referencia a un procedimiento remoto, si el procedimiento remoto se recompila después de compilar el procedimiento local.

Mecanismo de Dependencia Remota Automática

Cuando se compila un procedimiento, el servidor de Oracle realiza el registro de hora de dicha compilación en el código P del procedimiento.

En la diapositiva, 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.



**Procedimiento local A:
VALID**

**Procedimiento
remoto B: VALID**

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

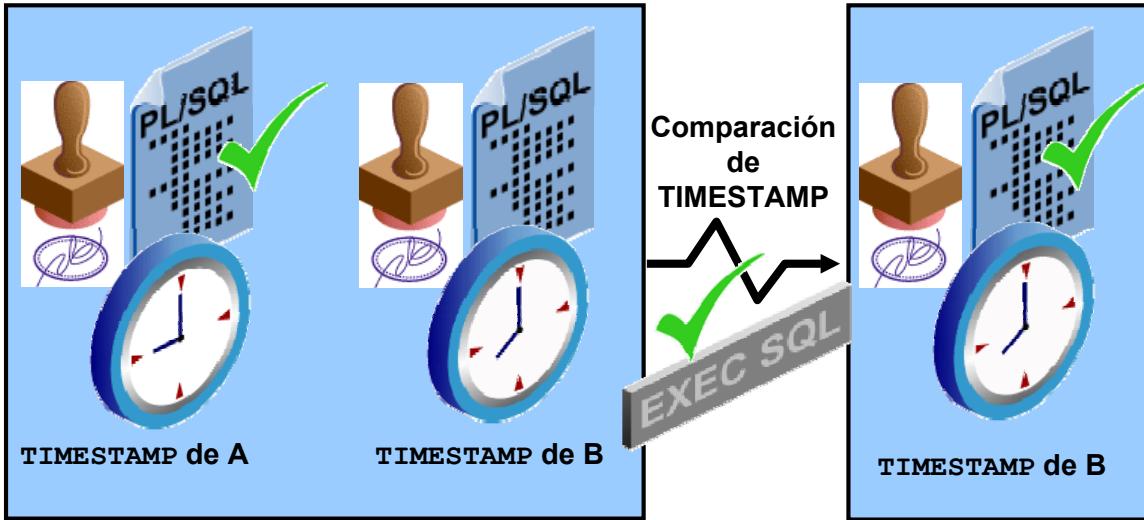
Procedimientos Locales que Hacen Referencia a Procedimientos Remotos (continuación)

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 diapositiva, 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



Procedimiento local A:
VALID

Procedimiento
remoto B: VALID

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

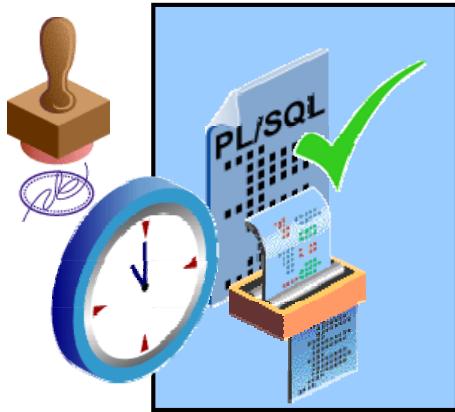
Dependencia Remota Automática

Cuando se llama al procedimiento local en tiempo de ejecución, el servidor de Oracle compara los dos registros de hora del procedimiento remoto al que se hace referencia.

Si los registros de hora son iguales (lo que indica que el procedimiento remoto no se ha recompilado), el servidor de Oracle ejecutará el procedimiento local.

En el ejemplo de la diapositiva, 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:
Se recompila y es VALID
a las 11:00 a.m.**

ORACLE

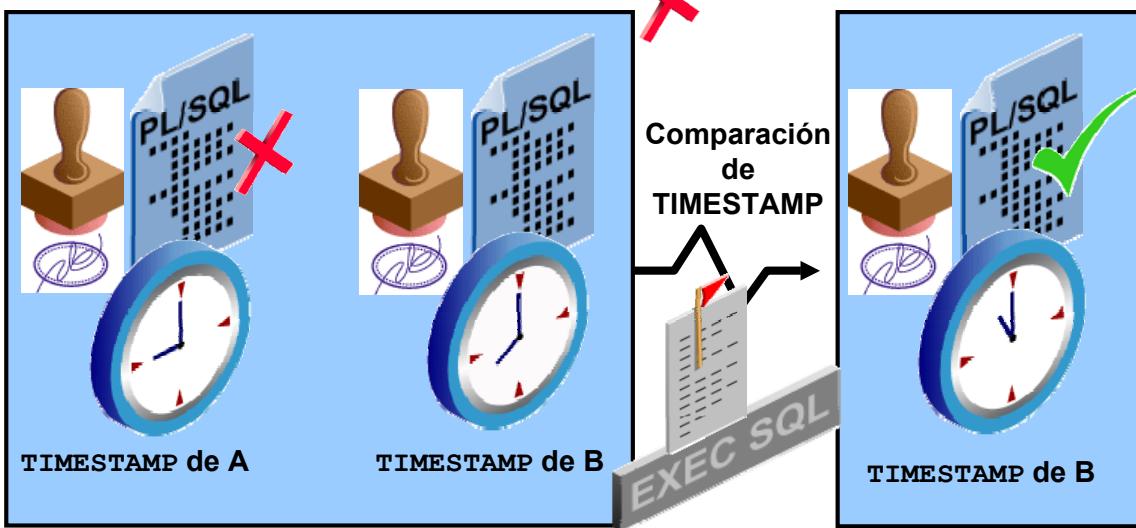
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

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 su código P.

Ejecución del Procedimiento A

TIMESTAMP guardado de B != TIEMPO DE COMPILACIÓN de B



Procedimiento local A:
INVALID

**Procedimiento
remoto B: VALID**

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Dependencia Remota Automática

Si los registros de hora no son iguales (lo que indica 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 (lo que indica 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 diapositiva, 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.



Copyright © 2010, Oracle y/o sus filiales. 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 © 2010, Oracle y/o sus filiales. 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 recompilar 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



Copyright © 2010, Oracle y/o sus filiales. 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 recomilar 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



Copyright © 2010, Oracle y/o sus filiales. 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 contuviera 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

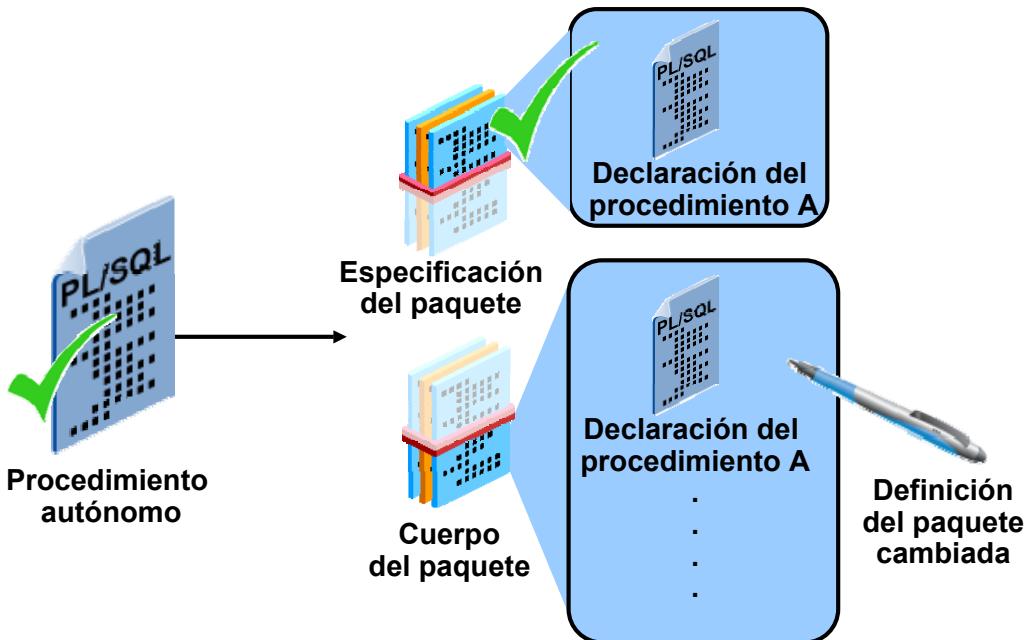


Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Recompilación de Procedimientos

Puede minimizar los fallos de recompilación siguiendo las instrucciones que se muestran en la diapositiva.

Paquetes y Dependencias: El Subprograma Hace Referencia al Paquete



ORACLE

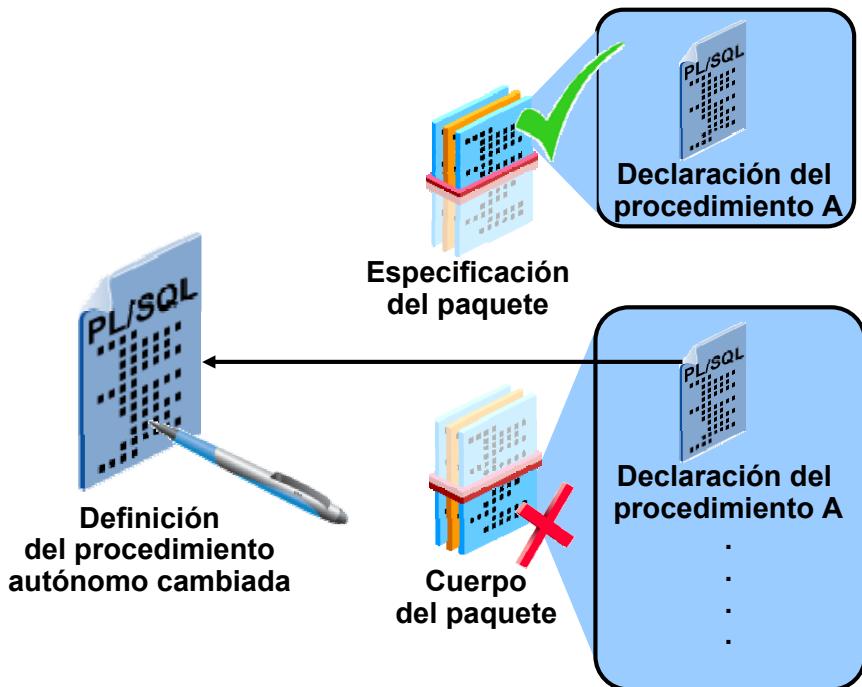
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Paquetes y Dependencias: El Subprograma Hace Referencia al Paquete

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: El Subprograma del Paquete Hace Referencia al Procedimiento



ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Paquetes y Dependencias: El Subprograma del Paquete Hace Referencia al Procedimiento

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.

Prueba

Puede visualizar dependencias directas e indirectas ejecutando el script `utldtree.sql`, rellenando la tabla `DEPTREE_TEMP TAB` con información para un objeto concreto al que se hace referencia y consultando las vistas `DEPTREE` o `I DEPTREE`.

1. Verdadero
2. Falso

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: 1

Visualización de Dependencias Directas e Indirectas

Puede visualizar dependencias directas e indirectas de la siguiente forma:

1. Ejecute el script `utldtree.sql` que crea los objetos que permiten mostrar las dependencias directas e indirectas.
2. Rellene la tabla `DEPTREE_TEMP TAB` con información para un objeto determinado al que se hace referencia, ejecutando al procedimiento `DEPTREE_FILL`.
3. Consulte las vistas `DEPTREE` o `I DEPTREE`.

Resumen

En esta lección, debe haber aprendido 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
- Gestionar dependencias de procedimiento



Copyright © 2010, Oracle y/o sus filiales. 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.

Visión General de la Práctica 12: Gestión de Dependencias en el Esquema

En esta práctica se abordan los siguientes temas:

- Uso de DEPTREE_FILL e IDEPTREE para ver las dependencias
- Recompilación de procedimientos, funciones y paquetes

ORACLE

Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

Práctica 12: 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.