

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

**Volumen 2: Guía del Alumno**

D17169CS11

Edición 1.1

Agosto de 2004

D21977

**ORACLE®**

## **Autores**

Glenn Stokol  
Aniket Raut

## **Colaboradores y Revisores Técnicos**

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

## **Editor**

Joseph Fernandez

## **Copyright © 2004, Oracle. Todos los Derechos Reservados.**

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

### **Restricted Rights Legend**

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

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

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

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

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

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

Oracle Internal & OAI Use Only

# Contenido

## Prefacio

### I Introducción

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

### 1 Creación de Procedimientos Almacenados

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

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

## **2 Creación de Funciones Almacenadas**

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

## **3 Creación de Paquetes**

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

#### **4 Uso de Más Conceptos de Paquete**

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

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

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

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

## **6 SQL Dinámico y Metadatos**

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

## **7 Consideraciones de Diseño para Código PL/SQL**

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

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

## 8 Gestión de Dependencias

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

## **9 Manipulación de Objetos Grandes**

Objetivos 9-2

¿Qué es un LOB? 9-3

Comparación de los Tipos de Dato LONG y LOB 9-5

Anatomía de un LOB 9-6

LOB Internos 9-7

Gestión de los LOB Internos 9-8

¿Qué son los BFILE? 9-9

Protección de BFILE 9-10

Nuevo Objeto de Base de Datos: DIRECTORY 9-11

Instrucciones para la Creación de Objetos DIRECTORY 9-12

Gestión de los BFILE 9-13

Preparación para Utilizar BFILE 9-15

Relleno de Columnas BFILE con SQL 9-16

Relleno de Columnas BFILE con PL/SQL 9-17

Uso de Rutinas DBMS\_LOB con BFILEs 9-18

Migración de LONG a LOB 9-19

Paquete DBMS\_LOB 9-21

DBMS\_LOB.READ y DBMS\_LOB.WRITE 9-24

Inicialización de Columnas LOB Agregadas a una Tabla 9-25

Relleno de Columnas LOB 9-26

Actualización de LOB con DBMS\_LOB en PL/SQL 9-27

Selección de Valores CLOB con SQL 9-28

Selección de Valores CLOB con DBMS\_LOB 9-29

Selección de Valores CLOB en PL/SQL 9-30

Eliminación de LOB 9-31

LOB Temporales 9-32

Creación de un LOB Temporal 9-33

Resumen 9-34

Práctica 9: Visión General 9-35

## **10 Creación de Disparadores**

Objetivos 10-2

Tipos de Disparadores 10-3

Instrucciones para el Diseño de Disparadores 10-5

Creación de Disparadores DML 10-7

Tipos de Disparadores DML 10-8

Temporización de Disparadores 10-9

Secuencia de Arranque de Disparadores 10-10

Tipos de Evento y Cuerpo del Disparador 10-12

Creación de un Disparador de Sentencia DML 10-13

Prueba de SECURE\_EMP 10-14

Uso de Predicados Condicionales 10-15

Creación de un Disparador de Fila DML 10-16

Uso de los Cualificadores OLD y NEW 10-17



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

## **11 Aplicaciones para Disparadores**

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

## **12 Descripción e Influencia del Compilador PL/SQL**

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

Uso de Procedimientos DBMS\_WARNING 12-14

Uso de Funciones DBMS\_WARNING 12-15

Uso de DBMS\_WARNING: Ejemplo 12-16

Resumen 12-18

Práctica 12: Visión General 12-19

## **Apéndice A: Soluciones a la Práctica**

## **Apéndice B: Descripciones de las Tablas y Datos**

## **Apéndice C: Estudios para Implementación de Disparadores**

Objetivos C-2

Control de la Seguridad en el Servidor C-3

Control de la Seguridad con un Disparador de Base de Datos C-4

Uso de la Utilidad del Servidor para Auditoría de Operaciones de Datos C-5

Auditoría con un Disparador C-6

Auditoría de Disparadores con Construcciones de Paquetes C-7

Paquete AUDIT\_PKG C-9

Tabla AUDIT\_TABLE y Procedimiento AUDIT\_EMP C-10

Forzado de Integridad de Datos en el Servidor C-11

Protección de la Integridad de los Datos con un Disparador C-12

Forzado de la Integridad Referencial en el Servidor C-13

Protección de la Integridad Referencial con un Disparador C-14

Replicación de Tablas en el Servidor C-15

Replicación de Tablas con un Disparador C-16

Cálculo de Datos Derivados en el Servidor C-17

Cálculo de Valores Derivados con un Disparador C-18

Registro de Eventos con un Disparador C-19

Resumen C-21

## **Apéndice D: Revisión de PL/SQL**

Estructura en Bloque para Bloques PL/SQL Anónimos D-2

Declaración de Variables PL/SQL D-3

Declaración de Variables con el Atributo %TYPE D-4

Creación de un Registro PL/SQL D-5

Atributo %ROWTYPE D-6

Creación de una Tabla PL/SQL D-7

Sentencias SELECT en PL/SQL D-8

Inserción de Datos D-9

Actualización de Datos D-10

Supresión de Datos D-11

Sentencias COMMIT y ROLLBACK D-12

Atributos de Cursor SQL D-13

Sentencias IF, THEN y ELSIF D-14

Bucle Básico D-18

Bucle FOR D-16

Bucle WHILE D-17

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

## **Apéndice E: JDeveloper**

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

## **Índice**

### **Prácticas Adicionales**

#### **Prácticas Adicionales: Soluciones**

#### **Prácticas Adicionales: Descripciones de las Tablas y Datos**

Oracle Internal & OAI Use Only

---

## Prefacio

---

Oracle Internal & OAI Use Only

Oracle Internal & OAI Use Only

## Perfil

### Cualificación Necesaria para el Curso

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

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

### Organización del Curso

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

Oracle Internal & OAI Use Only

## Publicaciones Relacionadas

### Publicaciones de Oracle

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

### Publicaciones Adicionales

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

Oracle Internal & OAI Use Only



## Convenciones Tipográficas

### Convenciones Tipográficas en el Texto

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

## Convenciones Tipográficas (continuación)

### Convenciones Tipográficas en el Texto (continuación)

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

## **Convenciones Tipográficas (continuación)**

### **Convenciones Tipográficas en las Rutas de Acceso de Navegación**

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

#### **Invoice Batch Summary**

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

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

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

#### **Notación:**

(N) = Navegador

(M) = Menú

(T) = Separador

(I) = Icono

(H) = Enlace de hipertexto

(B) = Botón

Oracle Internal & OAI Use Only

Oracle Internal & OAI Use Only

---

**A**

## **Soluciones a las Prácticas**

---

Oracle Internal & OAI Use Only

## Práctica I: Soluciones

1. Inicie *iSQL\*Plus* con el icono que aparece en el escritorio.
  - a. Conéctese a la base de datos utilizando el nombre de usuario y los datos de la cadena de conexión de la base de datos que le ha proporcionado el instructor (también puede escribir aquí la información para tenerla guardada):  
Username: **ora**\_\_  
Password: **oracle**  
Database Connect String/Tnsname: **T**\_\_
  - b. Ejecute sentencias básicas de `SELECT` para consultar los datos de las tablas `DEPARTMENTS`, `EMPLOYEES` y `JOBS`. Dedique unos minutos a familiarizarse con los datos o consulte el Apéndice B que proporciona una descripción y algunos datos de cada tabla del esquema Human Resources.

```
SELECT * FROM departments;  
SELECT * FROM employees;
```

2. Cree un procedimiento denominado `HELLO` para mostrar el texto “Hello World”.
  - a. Cree un procedimiento denominado `HELLO`.
  - b. En la sección ejecutable, utilice el procedimiento `DBMS_OUTPUT.PUT_LINE` para imprimir el texto “Hello Word” y guardar el código en la base de datos.  
**Nota:** Si obtiene errores en tiempo de compilación, edite PL/SQL para corregir el código y sustituir la palabra clave `CREATE` con el texto `CREATE OR REPLACE`.

```
CREATE PROCEDURE hello IS  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('Hello World');  
END;  
/  
  
Procedure created.
```

- c. Ejecute el comando `SET SERVEROUTPUT ON` para asegurarse de que la salida del procedimiento `DBMS_OUTPUT.PUT_LINE` se mostrará en *iSQL\*Plus*.

```
SET SERVEROUTPUT ON
```

- d. Cree un bloque anónimo para llamar al procedimiento almacenado.

```
BEGIN  
    hello;  
END;  
/  
  
Hello World  
PL/SQL procedure successfully completed.
```

## Práctica I: Soluciones (continuación)

3. Cree una función denominada TOTAL\_SALARY para calcular la suma de los sueldos de todos los empleados.
  - a. Cree una función denominada TOTAL\_SALARY que devuelve un NUMBER.
  - b. En la sección ejecutable, ejecute una consulta para almacenar el salario total de todos los empleados en una variable local que se declara en la sección de declaraciones. Devuelva el valor almacenado en la variable local. Guarde y compile el código.

```
CREATE FUNCTION total_salary RETURN NUMBER IS
    total employees.salary%type;
BEGIN
    SELECT SUM(salary) INTO total
    FROM employees;
    RETURN total;
END;
/

Function created.
```

- c. Utilice un bloque anónimo para llamar a la función. Para mostrar el resultado calculado por la función utilice el procedimiento DBMS\_OUTPUT.PUT\_LINE.  
**Indicación:** Anide la llamada de función dentro del parámetro DBMS\_OUTPUT.PUT\_LINE o almacene el resultado de la función en una variable local del bloque anónimo y utilice la variable local en el procedimiento DBMS\_OUTPUT.PUT\_LINE.

```
DECLARE
    total number := total_salary;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Total Salary: ' || total);
END;
/

-- OR ...
BEGIN
    DBMS_OUTPUT.PUT_LINE('Total Salary: ' || total_salary);
END;
/

Total Salary: 691400
PL/SQL procedure successfully completed.

Total Salary: 691400
PL/SQL procedure successfully completed.
```

## Práctica I: Soluciones (continuación)

Si tiene tiempo, realice el siguiente ejercicio:

4. Inicie SQL\*Plus con el icono que aparece en el escritorio.
  - a. Llame al procedimiento y a la función que ha creado en los ejercicios 2 y 3.

```
SET SERVEROUTPUT ON
EXECUTE hello;

Hello World
PL/SQL procedure successfully completed.

EXECUTE DBMS_OUTPUT.PUT_LINE('Total Salary: ' || total_salary);

Total Salary: 691400
PL/SQL procedure successfully completed.
```

- b. Cree un procedimiento nuevo denominado HELLO\_AGAIN para imprimir “Hello World again”.

```
CREATE PROCEDURE hello_again IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello World again');
END;
/

Procedure created.
```

- c. Llame al procedimiento HELLO\_AGAIN con un bloque anónimo.

```
SET SERVEROUTPUT ON
BEGIN
    hello_again;
END;
/

Hello World again
PL/SQL procedure successfully completed.
```



## Práctica 1: Soluciones

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

No olvide activar SERVEROUTPUT si lo ha desactivado previamente.

1. Cree y llame al procedimiento ADD\_JOB y tenga en cuenta los resultados.
  - a. Cree un procedimiento denominado ADD\_JOB para insertar un nuevo trabajo en la tabla JOBS. Proporcione el identificador y el título del trabajo utilizando dos parámetros.

```
CREATE OR REPLACE PROCEDURE add_job (  
  jobid jobs.job_id%TYPE,  
  jobtitle jobs.job_title%TYPE) IS  
BEGIN  
  INSERT INTO jobs (job_id, job_title)  
  VALUES (jobid, jobtitle);  
  COMMIT;  
END add_job;  
/  
Procedure created.
```

- b. Compile el código y llame al procedimiento con IT\_DBA como identificador de trabajo y Database Administrator como título. Consulte la tabla JOBS para ver los resultados.

```
EXECUTE add_job ('IT_DBA', 'Database Administrator')  
SELECT * FROM jobs WHERE job_id = 'IT_DBA';
```

PL/SQL Procedure Successfully Completed.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Database Administrator		

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

```
EXECUTE add_job ('ST_MAN', 'Stock Manager')  
  
BEGIN add_job ('ST_MAN', 'Stock Manager'); END;  
  
*  
  
ERROR at line 1:  
ORA-00001: unique constraint (ORA1.JOB_ID_PK) violated  
ORA-06512: at "ORA1.ADD_JOB", line 5  
ORA-06512: at line 1
```

**Se ha producido una excepción porque hay una restricción de integridad de clave primaria en la columna JOB\_ID.**

## Práctica 1: Soluciones (continuación)

2. Cree un procedimiento denominado UPD\_JOB para modificar un trabajo en la tabla JOBS.
  - a. Cree un procedimiento denominado UPD\_JOB para actualizar el título. Proporcione el identificador de trabajo y un título nuevo utilizando dos parámetros. Incluya el manejo de excepciones necesario si no se ha producido la actualización.

```
CREATE OR REPLACE PROCEDURE upd_job(  
  jobid   IN jobs.job_id%TYPE,  
  jobtitle IN jobs.job_title%TYPE) IS  
BEGIN  
  UPDATE jobs  
  SET    job_title = jobtitle  
  WHERE job_id = jobid;  
  IF SQL%NOTFOUND THEN  
    RAISE_APPLICATION_ERROR(-20202, 'No job updated.');  END IF;  
END upd_job;  
/  
  
Procedure created.
```

- b. Compile el código, llame al procedimiento para cambiar el título del identificador de trabajo IT\_DBA a Data Administrator. Consulte la tabla JOBS para ver los resultados.

```
EXECUTE upd_job ('IT_DBA', 'Data Administrator')  
SELECT * FROM jobs WHERE job_id = 'IT_DBA';
```

PL/SQL Procedure Successfully Completed.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Data Administrator		

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

```
EXECUTE upd_job ('IT_WEB', 'Web Master')  
  
BEGIN upd_job ('IT_WEB', 'Web Master'); END;  
  
*  
  
ERROR at line 1:  
ORA-20202: No job updated.  
ORA-06512: at "ORA1.UPD_JOB", line 9  
ORA-06512: at line 1
```

## Práctica 1: Soluciones (continuación)

3. Cree un procedimiento denominado DEL\_JOB para suprimir un trabajo en la tabla JOBS.
  - a. Cree un procedimiento denominado DEL\_JOB para suprimir un trabajo. Incluya el manejo de excepciones necesario si no se suprime ningún trabajo.

```
CREATE OR REPLACE PROCEDURE del_job (jobid jobs.job_id%TYPE) IS
BEGIN
    DELETE FROM jobs
    WHERE job_id = jobid;
    IF SQL%NOTFOUND THEN
        RAISE_APPLICATION_ERROR(-20203, 'No jobs deleted.');
```

Procedure created.

- b. Compile el código; llame al procedimiento utilizando el identificador de trabajo IT\_DBA. Consulte la tabla JOBS para ver los resultados.

```
EXECUTE del_job ('IT_DBA')
SELECT * FROM jobs WHERE job_id = 'IT_DBA';

PL/SQL procedure successfully completed.

no rows selected
```

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

```
EXECUTE del_job ('IT_WEB')

BEGIN del_job ('IT_WEB'); END;

*

ERROR at line 1:
ORA-20203: No jobs deleted.
ORA-06512: at "ORA1.DEL_JOB", line 6
ORA-06512: at line 1
```

4. Cree un procedimiento denominado GET\_EMPLOYEE para consultar la tabla EMPLOYEES, lo que devuelve el salario y el identificador de trabajo de un empleado cuando se proporciona el identificador de empleado.
  - a. Cree un procedimiento que devuelva un valor de las columnas SALARY y JOB\_ID para el identificador de empleado especificado. Compile el código y elimine los errores de sintaxis.

## Práctica 1: Soluciones (continuación)

```
CREATE OR REPLACE PROCEDURE get_employee
(empid IN employees.employee_id%TYPE,
 sal   OUT employees.salary%TYPE,
 job   OUT employees.job_id%TYPE) IS
BEGIN
  SELECT salary, job_id
  INTO   sal, job
  FROM   employees
  WHERE  employee_id = empid;
END get_employee;
/
```

Procedure created.

- b. Ejecute el procedimiento utilizando las variables del host para los dos parámetros OUT uno para el salario y el otro para el identificador de trabajo. Muestre el salario y el identificador de trabajo para el identificador de empleado 120.

```
VARIABLE salary NUMBER
VARIABLE job      VARCHAR2(15)
EXECUTE get_employee(120, :salary, :job)
PRINT salary job
```

PL/SQL procedure successfully completed.

SALARY	
	8000

JOB	
ST_MAN	

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

```
EXECUTE get_employee(300, :salary, :job)

BEGIN get_employee(300, :salary, :job); END;

*

ERROR at line 1:
ORA-01403: no data found
ORA-06512: at "ORA1.GET_EMPLOYEE", line 6
ORA-06512: at line 1
```

Ningún empleado de la tabla **EMPLOYEES** tiene un **EMPLOYEE\_ID** de 300. La sentencia **SELECT** no recuperó ningún dato de la base de datos y generó un error **PL/SQL fatal: NO\_DATA\_FOUND**.

## Práctica 2: Soluciones

1. Cree y llame a la función GET\_JOB para devolver un título.

a. Cree y compile la función denominada GET\_JOB para devolver un título.

```
CREATE OR REPLACE FUNCTION get_job (jobid IN jobs.job_id%type)
RETURN jobs.job_title%type IS
    title jobs.job_title%type;
BEGIN
    SELECT job_title
    INTO title
    FROM jobs
    WHERE job_id = jobid;
    RETURN title;
END get_job;
/

Function created.
```

b. Cree una variable de host VARCHAR2 denominada TITLE, que permita una longitud de 35 caracteres. Llame a la función con identificador de trabajo SA\_REP para que devuelva el valor de la variable del host. Imprima la variable del host para ver el resultado.

```
VARIABLE title VARCHAR2(35)
EXECUTE :title := get_job ('SA_REP');
PRINT title

PL/SQL procedure successfully completed.
```

TITLE
Sales Representative

2. Cree una función denominada GET\_ANNUAL\_COMP para devolver el salario anual de un empleado calculado a partir del salario mensual y la comisión transferidos como parámetros.

a. Desarrolle y almacene la función GET\_ANNUAL\_COMP, aceptando valores de parámetros para salario mensual y comisión. Uno o ambos valores transferidos pueden ser NULL, pero la función deberá devolver un salario anual no NULL. Utilice la siguiente fórmula básica para calcular el salario anual:

$$(\text{salary} * 12) + (\text{commission\_pct} * \text{salary} * 12)$$

```
CREATE OR REPLACE FUNCTION get_annual_comp(
    sal IN employees.salary%TYPE,
    comm IN employees.commission_pct%TYPE)
RETURN NUMBER IS
BEGIN
    RETURN (NVL(sal,0) * 12 + (NVL(comm,0) * nvl(sal,0) * 12));
END get_annual_comp;
/

Function created.
```

## Práctica 2: Soluciones (continuación)

- b. Utilice la función en una sentencia SELECT en la tabla EMPLOYEES para los empleados del departamento 30.

```
SELECT employee_id, last_name,  
       get_annual_comp(salary,commission_pct) "Annual Compensation"  
FROM   employees  
WHERE  department_id=30  
/
```

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

6 rows selected.

3. Cree un procedimiento, ADD\_EMPLOYEE, para insertar un nuevo empleado en la tabla EMPLOYEES. El procedimiento llamará a una función VALID\_DEPTID para comprobar si el identificador de departamento especificado para el nuevo empleado existe en la tabla DEPARTMENTS.
- a. Cree una función VALID\_DEPTID para validar el identificador de departamento especificado y devolver un valor BOOLEAN de TRUE si existe el departamento.

```
CREATE OR REPLACE FUNCTION valid_deptid(  
  deptid IN departments.department_id%TYPE)  
RETURN BOOLEAN IS  
  dummy PLS_INTEGER;  
BEGIN  
  SELECT 1  
  INTO    dummy  
  FROM    departments  
  WHERE   department_id = deptid;  
  RETURN TRUE;  
EXCEPTION  
  WHEN NO_DATA_FOUND THEN  
    RETURN FALSE;  
END valid_deptid;  
/
```

Function created.

## Práctica 2: Soluciones (continuación)

- b. Cree el procedimiento ADD\_EMPLOYEE para agregar un empleado a la tabla EMPLOYEES.

La fila se agregará a la tabla EMPLOYEES si la función VALID\_DEPTID devuelve TRUE, de lo contrario, alertará al usuario con un mensaje adecuado. Proporcione los siguientes parámetros (con los valores por defecto especificados entre paréntesis): first\_name, last\_name, email, job (SA\_REP), mgr (145), sal (1000), comm (0), y deptid (30). Utilice la secuencia EMPLOYEES\_SEQ para definir la columna employee\_id y definir hire\_date en TRUNC (SYSDATE).

```
CREATE OR REPLACE PROCEDURE add_employee(  
    first_name employees.first_name%TYPE,  
    last_name   employees.last_name%TYPE,  
    email       employees.email%TYPE,  
    job         employees.job_id%TYPE          DEFAULT 'SA_REP',  
    mgr         employees.manager_id%TYPE      DEFAULT 145,  
    sal         employees.salary%TYPE          DEFAULT 1000,  
    comm        employees.commission_pct%TYPE  DEFAULT 0,  
    deptid      employees.department_id%TYPE   DEFAULT 30) IS  
BEGIN  
    IF valid_deptid(deptid) THEN  
        INSERT INTO employees(employee_id, first_name, last_name, email,  
                                job_id, manager_id, hire_date, salary, commission_pct,  
                                department_id)  
        VALUES (employees_seq.NEXTVAL, first_name, last_name, email,  
                job, mgr, TRUNC(SYSDATE), sal, comm, deptid);  
    ELSE  
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID. Try again.');    END IF;  
END add_employee;  
/  
  
Procedure created.
```

- c. Llame a ADD\_EMPLOYEE para el nombre Jane Harris del departamento 15, dejando otros parámetros con los valores por defecto. ¿Cuál es el resultado?

```
EXECUTE add_employee('Jane', 'Harris', 'JAHARRIS', deptid=> 15)  
  
BEGIN add_employee('Jane', 'Harris', 'JAHARRIS', deptid=> 15); END;  
  
*  
  
ERROR at line 1:  
ORA-20204: Invalid department ID. Try again.  
ORA-06512: at "ORA1.ADD_EMPLOYEE", line 17  
ORA-06512: at line 1
```

## Práctica 2: Soluciones (continuación)

- d. Agregue otro empleado llamado Joe Harris en el departamento 80, dejando los parámetros restantes con sus valores por defecto. ¿Cuál es el resultado?

```
EXECUTE add_employee('Joe', 'Harris', 'JAHARRIS', deptid=> 80)
```

```
PL/SQL procedure successfully completed.
```

Oracle Internal & OAI Use Only



### Práctica 3: Soluciones

1. Cree una especificación y cuerpo del paquete denominado JOB\_PKG que contenga una copia de los procedimientos ADD\_JOB, UPD\_JOB y DEL\_JOB así como la función GET\_JOB.

**Consejo:** Puede guardar la especificación y el cuerpo del paquete en dos archivos separados (por ejemplo, p3q1\_s.sql y p3q1\_b.sql para la especificación y el cuerpo del paquete respectivamente). Incluya una sentencia SHOW ERRORS después de la sentencia CREATE PACKAGE en cada archivo. Asimismo, coloque todo el código en un sólo archivo.

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

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

```
CREATE OR REPLACE PACKAGE job_pkg IS
  PROCEDURE add_job (jobid jobs.job_id%TYPE, jobtitle
jobs.job_title%TYPE);
  PROCEDURE del_job (jobid jobs.job_id%TYPE);
  FUNCTION get_job (jobid IN jobs.job_id%type) RETURN
jobs.job_title%type;
  PROCEDURE upd_job(jobid IN jobs.job_id%TYPE, jobtitle IN
jobs.job_title%TYPE);
END job_pkg;
/
SHOW ERRORS

Package created.

No errors.
```

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

- b. Cree el cuerpo del paquete con las implementaciones de cada uno de los subprogramas.

```
CREATE OR REPLACE PACKAGE BODY job_pkg IS
  PROCEDURE add_job (
    jobid jobs.job_id%TYPE,
    jobtitle jobs.job_title%TYPE) IS
  BEGIN
    INSERT INTO jobs (job_id, job_title)
    VALUES (jobid, jobtitle);
    COMMIT;
  END add_job;

  PROCEDURE del_job (jobid jobs.job_id%TYPE) IS
  BEGIN
    DELETE FROM jobs
    WHERE job_id = jobid;
    IF SQL%NOTFOUND THEN
      RAISE_APPLICATION_ERROR(-20203, 'No jobs deleted.');
```

### Práctica 3: Soluciones (continuación)

```
FUNCTION get_job (jobid IN jobs.job_id%type)
  RETURN jobs.job_title%type IS
  title jobs.job_title%type;
BEGIN
  SELECT job_title
  INTO title
  FROM jobs
  WHERE job_id = jobid;
  RETURN title;
END get_job;

PROCEDURE upd_job(
  jobid IN jobs.job_id%TYPE,
  jobtitle IN jobs.job_title%TYPE) IS
BEGIN
  UPDATE jobs
  SET   job_title = jobtitle
  WHERE job_id = jobid;
  IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR(-20202, 'No job updated.');

END IF;



END upd_job;



END job_pkg;



/



SHOW ERRORS



Package body created.



No errors.


```

- c. Llame al procedimiento empaquetado ADD\_JOB transfiriendo como parámetros los valores IT\_SYSAN y Systems Analyst.

```
EXECUTE job_pkg.add_job('IT_SYSAN', 'Systems Analyst')

PL/SQL procedure successfully completed.
```

- d. Consulte la tabla JOBS para ver el resultado.

```
SELECT *
FROM jobs
WHERE job_id = 'IT_SYSAN';
```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_SYSAN	Systems Analyst		

### Práctica 3: Soluciones (continuación)

2. Cree y llame a un paquete que contenga construcciones públicas y privadas.
  - a. Cree una especificación del paquete y un cuerpo del paquete denominados EMP\_PKG que contengan los procedimientos ADD\_EMPLOYEE y GET\_EMPLOYEE como construcciones públicas e incluyan la función VALID\_DEPTID como construcción privada.

Especificación del Paquete:

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30);
  PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE);
END emp_pkg;
/
SHOW ERRORS

Package created.

No errors.
```

Cuerpo del Paquete:

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
    RETURN BOOLEAN IS
    dummy PLS_INTEGER;
  BEGIN
    SELECT 1
    INTO    dummy
    FROM    departments
    WHERE   department_id = deptid;
    RETURN TRUE;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN FALSE;
  END valid_deptid;
  -- ...
```

### Práctica 3: Soluciones (continuación)

Cuerpo del Paquete (continuación):

```
PROCEDURE add_employee(  
    first_name employees.first_name%TYPE,  
    last_name  employees.last_name%TYPE,  
    email      employees.email%TYPE,  
    job        employees.job_id%TYPE          DEFAULT 'SA_REP',  
    mgr        employees.manager_id%TYPE      DEFAULT 145,  
    sal        employees.salary%TYPE          DEFAULT 1000,  
    comm       employees.commission_pct%TYPE  DEFAULT 0,  
    deptid     employees.department_id%TYPE   DEFAULT 30) IS  
BEGIN  
    IF valid_deptid(deptid) THEN  
        INSERT INTO employees(employee_id, first_name, last_name, email,  
                                job_id, manager_id, hire_date, salary, commission_pct, department_id)  
        VALUES (employees_seq.NEXTVAL, first_name, last_name, email,  
                job, mgr, TRUNC(SYSDATE), sal, comm, deptid);  
    ELSE  
        RAISE_APPLICATION_ERROR (-20204,  
                                   'Invalid department ID. Try again.');    END IF;  
END add_employee;  
  
PROCEDURE get_employee(  
    empid IN employees.employee_id%TYPE,  
    sal    OUT employees.salary%TYPE,  
    job    OUT employees.job_id%TYPE) IS  
BEGIN  
    SELECT salary, job_id  
    INTO    sal, job  
    FROM    employees  
    WHERE   employee_id = empid;  
END get_employee;  
END emp_pkg;  
/  
SHOW ERRORS  
  
Package body created.  
  
No errors.
```

### Práctica 3: Soluciones (continuación)

- b. Llame al procedimiento `EMP_PKG.GET_EMPLOYEE` y utilice el identificador de departamento 15 para la empleada Jane Harris con correo electrónico JAHARRIS. Como el identificador de departamento 15 no existe recibirá un mensaje de error como se especifica en el manejador de excepciones del procedimiento.

```
EXECUTE emp_pkg.add_employee('Jane', 'Harris','JAHARRIS', deptid => 15)

BEGIN emp_pkg.add_employee('Jane', 'Harris','JAHARRIS', deptid => 15);
END;

*

ERROR at line 1:
ORA-20204: Invalid department ID. Try again.
ORA-06512: at "ORA1.EMP_PKG", line 31
ORA-06512: at line 1
```

- c. Llame al procedimiento empaquetado `GET_EMPLOYEE` utilizando el identificador de departamento 80 para el empleado David Smith con correo electrónico DASMITH.

```
EXECUTE emp_pkg.add_employee('David', 'Smith','DASMITH', deptid => 80)

PL/SQL procedure successfully completed.
```

Oracle Internal & OAI Use Only

## Práctica 4: Soluciones

1. Copie y modifique el código del paquete EMP\_PKG que creó en la Práctica 3, Ejercicio 2 y sobrecargue el procedimiento ADD\_EMPLOYEE.
  - a. En la especificación del paquete, agregue un nuevo procedimiento denominado ADD\_EMPLOYEE, que acepte tres parámetros: nombre, apellido e identificador de departamento. Guarde y compile los cambios.

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE DEFAULT 145,
    sal        employees.salary%TYPE DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30);
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    deptid     employees.department_id%TYPE);
  PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE);
END emp_pkg;
/
SHOW ERRORS

Package created.

No errors.
```

- b. Implemente el nuevo procedimiento ADD\_EMPLOYEE en el cuerpo del paquete para que formatee la dirección de correo electrónico en mayúsculas, utilizando la primera letra del nombre concatenada con las primeras siete letras del apellido. El procedimiento llamará al procedimiento ADD\_EMPLOYEE existente para realizar la operación INSERT utilizando los parámetros y el correo electrónico formateado para proporcionar los valores. Guarde y compile los cambios.

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
    RETURN BOOLEAN IS
    dummy PLS_INTEGER;
  BEGIN
    SELECT 1
    INTO    dummy
    FROM    departments
    WHERE   department_id = deptid;
    RETURN TRUE;
  END;
```

## Práctica 4: Soluciones (continuación)

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END valid_deptid;

PROCEDURE add_employee(
  first_name employees.first_name%TYPE,
  last_name  employees.last_name%TYPE,
  email      employees.email%TYPE,
  job employees.job_id%TYPE DEFAULT 'SA_REP',
  mgr employees.manager_id%TYPE DEFAULT 145,
  sal        employees.salary%TYPE          DEFAULT 1000,
  comm       employees.commission_pct%TYPE  DEFAULT 0,
  deptid     employees.department_id%TYPE   DEFAULT 30) IS
BEGIN
  IF valid_deptid(deptid) THEN
    INSERT INTO employees(employee_id, first_name, last_name, email,
      job_id, manager_id, hire_date, salary, commission_pct, department_id)
    VALUES (employees_seq.NEXTVAL, first_name, last_name, email,
      job, mgr, TRUNC(SYSDATE), sal, comm, deptid);
  ELSE
    RAISE_APPLICATION_ERROR (-20204,
      'Invalid department ID. Try again.');
```

**PROCEDURE add\_employee(**  
**first\_name employees.first\_name%TYPE,**  
**last\_name employees.last\_name%TYPE,**  
**deptid employees.department\_id%TYPE) IS**  
**email employees.email%type;**  
**BEGIN**  
**email := UPPER(SUBSTR(first\_name, 1, 1)||SUBSTR(last\_name, 1, 7));**  
**add\_employee(first\_name, last\_name, email, deptid => deptid);**  
**END;**

```
PROCEDURE get_employee(
  empid IN employees.employee_id%TYPE,
  sal    OUT employees.salary%TYPE,
  job    OUT employees.job_id%TYPE) IS
BEGIN
  SELECT salary, job_id
  INTO   sal, job
  FROM   employees
  WHERE  employee_id = empid;
END get_employee;
END emp_pkg;
/
SHOW ERRORS
```

## Práctica 4: Soluciones (continuación)

- c. Llame al nuevo procedimiento ADD\_EMPLOYEE utilizando el nombre Samuel Joplin para agregarlo al departamento 30.

```
EXECUTE emp_pkg.add_employee('Samuel', 'Joplin', 30)
```

```
PL/SQL procedure successfully completed.
```

2. En el paquete EMP\_PKG, cree dos funciones sobrecargadas denominadas GET\_EMPLOYEE.

- a. En la especificación, agregue una función GET\_EMPLOYEE que acepte el parámetro denominado emp\_id según el tipo employees.employee\_id%TYPE y una segunda función GET\_EMPLOYEE que acepte un parámetro denominado family\_name del tipo employees.last\_name%TYPE. Ambas funciones devolverán un EMPLOYEES%ROWTYPE. Guarde y compile los cambios.

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30);
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    deptid     employees.department_id%TYPE);
  PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE);
  FUNCTION get_employee(emp_id employees.employee_id%type)
    RETURN employees%rowtype;
  FUNCTION get_employee(family_name employees.last_name%type)
    RETURN employees%rowtype;
END emp_pkg;
/
SHOW ERRORS

Package created.

No errors.
```



## Práctica 4: Soluciones (continuación)

- b. En el cuerpo del paquete, implemente la primera función GET\_EMPLOYEE para realizar una consulta sobre un empleado según su identificador y la segunda para utilizar el operador de igualdad sobre el valor proporcionado en el parámetro family\_name. Guarde y compile los cambios.

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
    RETURN BOOLEAN IS
    dummy PLS_INTEGER;
  BEGIN
    SELECT 1
    INTO    dummy
    FROM    departments
    WHERE   department_id = deptid;
    RETURN TRUE;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN FALSE;
  END valid_deptid;

  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE          DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE      DEFAULT 145,
    sal        employees.salary%TYPE          DEFAULT 1000,
    comm       employees.commission_pct%TYPE  DEFAULT 0,
    deptid     employees.department_id%TYPE   DEFAULT 30) IS
  BEGIN
    IF valid_deptid(deptid) THEN
      INSERT INTO employees(employee_id, first_name, last_name, email,
        job_id, manager_id, hire_date, salary, commission_pct, department_id)
      VALUES (employees_seq.NEXTVAL, first_name, last_name, email,
        job, mgr, TRUNC(SYSDATE), sal, comm, deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204,
        'Invalid department ID. Try again.');
```

END IF;

```
END add_employee;

  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    deptid     employees.department_id%TYPE) IS
    email      employees.email%TYPE;
  BEGIN
    email := UPPER(SUBSTR(first_name, 1, 1)||SUBSTR(last_name, 1, 7));
    add_employee(first_name, last_name, email, deptid => deptid);
  END;
```

## Práctica 4: Soluciones (continuación)

```
PROCEDURE get_employee(  
    empid IN employees.employee_id%TYPE,  
    sal    OUT employees.salary%TYPE,  
    job    OUT employees.job_id%TYPE) IS  
BEGIN  
    SELECT  salary, job_id  
    INTO    sal, job  
    FROM    employees  
    WHERE   employee_id = empid;  
END get_employee;  
  
FUNCTION get_employee(emp_id employees.employee_id%type)  
    return employees%rowtype IS  
    emprec employees%rowtype;  
BEGIN  
    SELECT * INTO emprec  
    FROM employees  
    WHERE  employee_id = emp_id;  
    RETURN emprec;  
END;  
  
FUNCTION get_employee(family_name employees.last_name%type)  
    return employees%rowtype IS  
    emprec employees%rowtype;  
BEGIN  
    SELECT * INTO emprec  
    FROM employees  
    WHERE last_name = family_name;  
    RETURN emprec;  
END;  
  
END emp_pkg;  
/  
SHOW ERRORS  
  
Package body created.  
  
No errors.
```

## Práctica 4: Soluciones (continuación)

- c. Agregue un procedimiento de utilidad PRINT\_EMPLOYEE al paquete que acepte EMPLOYEES%ROWTYPE como parámetro y muestre department\_id, employee\_id, first\_name, last\_name, job\_id y salary para un empleado en una sola línea utilizando DBMS\_OUTPUT. Guarde y compile los cambios.

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30);
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    deptid     employees.department_id%TYPE);
  PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE);
  FUNCTION get_employee(emp_id employees.employee_id%type)
    return employees%rowtype;
  FUNCTION get_employee(family_name employees.last_name%type)
    return employees%rowtype;
  PROCEDURE print_employee(emprec employees%rowtype);
END emp_pkg;
/
SHOW ERRORS

Package created.

No Errors.

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
  RETURN BOOLEAN IS
    dummy PLS_INTEGER;
  BEGIN
    SELECT 1
    INTO   dummy
    FROM   departments
    WHERE  department_id = deptid;
    RETURN TRUE;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN FALSE;
  END valid_deptid;
```

## Práctica 4: Soluciones (continuación)

```
PROCEDURE add_employee(  
    first_name employees.first_name%TYPE,  
    last_name  employees.last_name%TYPE,  
    email      employees.email%TYPE,  
    job        employees.job_id%TYPE          DEFAULT 'SA_REP',  
    mgr        employees.manager_id%TYPE      DEFAULT 145,  
    sal        employees.salary%TYPE          DEFAULT 1000,  
    comm       employees.commission_pct%TYPE  DEFAULT 0,  
    deptid     employees.department_id%TYPE   DEFAULT 30) IS  
BEGIN  
    IF valid_deptid(deptid) THEN  
        INSERT INTO employees(employee_id, first_name, last_name, email,  
                                job_id, manager_id, hire_date, salary, commission_pct, department_id)  
        VALUES (employees_seq.NEXTVAL, first_name, last_name, email,  
                job, mgr, TRUNC(SYSDATE), sal, comm, deptid);  
    ELSE  
        RAISE_APPLICATION_ERROR (-20204,  
                                   'Invalid department ID. Try again.');    END IF;  
END add_employee;  
  
PROCEDURE add_employee(  
    first_name employees.first_name%TYPE,  
    last_name  employees.last_name%TYPE,  
    deptid     employees.department_id%TYPE) IS  
    email      employees.email%type;  
BEGIN  
    email := UPPER(SUBSTR(first_name, 1, 1)||SUBSTR(last_name, 1, 7));  
    add_employee(first_name, last_name, email, deptid => deptid);  
END;  
  
PROCEDURE get_employee(  
    empid IN employees.employee_id%TYPE,  
    sal    OUT employees.salary%TYPE,  
    job    OUT employees.job_id%TYPE) IS  
BEGIN  
    SELECT salary, job_id  
    INTO    sal, job  
    FROM employees  
    WHERE   employee_id = empid;  
END get_employee;  
  
FUNCTION get_employee(emp_id employees.employee_id%type)  
    return employees%rowtype IS  
    emprec employees%rowtype;  
BEGIN  
    SELECT * INTO emprec  
    FROM employees  
    WHERE   employee_id = emp_id;  
    RETURN emprec;  
END;
```

## Práctica 4: Soluciones (continuación)

```
FUNCTION get_employee(family_name employees.last_name%type)
  return employees%rowtype IS
  emprec employees%rowtype;
BEGIN
  SELECT * INTO emprec
  FROM employees
  WHERE last_name = family_name;
  RETURN emprec;
END;

PROCEDURE print_employee(emprec employees%rowtype) IS
BEGIN
  DBMS_OUTPUT.PUT_LINE(emprec.department_id || ' ' ||
    emprec.employee_id || ' ' ||
    emprec.first_name || ' ' ||
    emprec.last_name || ' ' ||
    emprec.job_id || ' ' ||
    emprec.salary);
END;
END emp_pkg;
/
SHOW ERRORS

Package body created.

No errors.
```

- d. Utilice un bloque anónimo para llamar a la función EMP\_PKG.GET\_EMPLOYEE con un identificador de empleado de 100 y con apellido 'Joplin'. Utilice el procedimiento PRINT\_EMPLOYEE para mostrar los resultados para cada fila devuelta.

```
BEGIN
  emp_pkg.print_employee(emp_pkg.get_employee(100));
  emp_pkg.print_employee(emp_pkg.get_employee('Joplin'));
END;
/

90 100 Steven King AD_PRE 24000
30 209 Samuel Joplin SA_REP 1000

PL/SQL procedure successfully completed.
```

**Nota:** El identificador de empleado 209 para Samuel Joplin se asigna utilizando un objeto de secuencia de Oracle. Puede que reciba un valor diferente cuando ejecute el bloque PL/SQL que se muestra en esta solución.

## Práctica 4: Soluciones (continuación)

3. Como la compañía no cambia con frecuencia sus datos de departamento, puede mejorar el rendimiento de EMP\_PKG agregando un procedimiento público INIT\_DEPARTMENTS para rellenar una tabla PL/SQL privada de identificadores de departamento válidos. Modifique la función VALID\_DEPTID para utilizar el contenido de la tabla PL/SQL privada para validar los valores de los identificadores de departamento.
  - a. En la especificación del paquete, cree un procedimiento denominado INIT\_DEPARTMENTS sin parámetros.

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE          DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE      DEFAULT 145,
    sal        employees.salary%TYPE          DEFAULT 1000,
    comm       employees.commission_pct%TYPE  DEFAULT 0,
    deptid     employees.department_id%TYPE   DEFAULT 30);
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    deptid     employees.department_id%TYPE);
  PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE);
  FUNCTION get_employee(emp_id employees.employee_id%type)
    return employees%rowtype;
  FUNCTION get_employee(family_name employees.last_name%type)
    return employees%rowtype;
  PROCEDURE init_departments;
  PROCEDURE print_employee(emprec employees%rowtype);
END emp_pkg;
/
SHOW ERRORS

Package created.

No errors.
```

- b. En el cuerpo del paquete, implemente el procedimiento INIT\_DEPARTMENTS para almacenar todos los identificadores de departamento en una tabla PL/SQL privada index-by denominada valid\_departments que contiene valores BOOLEAN. Utilice el valor de la columna department\_id como índice para crear la entrada en la tabla index-by para indicar su presencia y asignar a la entrada un valor de TRUE. Declare la variable valid\_departments y su definición de tipo boolean\_tabtype antes que todos los procedimientos del cuerpo.

## Práctica 4: Soluciones (continuación)

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  TYPE boolean_tabtype IS TABLE OF BOOLEAN
    INDEX BY BINARY_INTEGER;
  valid_departments boolean_tabtype;

  FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
    RETURN BOOLEAN IS
    dummy PLS_INTEGER;
  BEGIN
    ...
  END valid_deptid;

  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30) IS
  BEGIN
    ...
  END add_employee;

  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    deptid     employees.department_id%TYPE) IS
    email employees.email%type;
  BEGIN
    ...
  END;

  PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE) IS
  BEGIN
    ...
  END get_employee;

  FUNCTION get_employee(emp_id employees.employee_id%type)
    return employees%rowtype IS
    emprec employees%rowtype;
  BEGIN
    ...
  END;
```

## Práctica 4: Soluciones (continuación)

```
FUNCTION get_employee(family_name employees.last_name%type)
    return employees%rowtype IS
    emprec employees%rowtype;
BEGIN
    SELECT * INTO emprec
    FROM employees
    WHERE last_name = family_name;
    RETURN emprec;
END;

PROCEDURE print_employee(emprec employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(emprec.department_id || ' ' ||
                          emprec.employee_id || ' ' ||
                          emprec.first_name || ' ' ||
                          emprec.last_name || ' ' ||
                          emprec.job_id || ' ' ||
                          emprec.salary);
END;

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;
END emp_pkg;
/
SHOW ERRORS

Package body created.

No errors.
```

- c. En el cuerpo, cree un bloque de inicialización que llame al procedimiento INIT\_DEPARTMENTS para inicializar la tabla. Guarde y compile los cambios.

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
...
PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;
BEGIN
    init_departments;
END emp_pkg;
/
SHOW ERRORS
```



## Práctica 4: Soluciones (continuación)

4. Cambie el procesamiento de validación VALID\_DEPTID para utilizar la tabla PL/SQL privada de identificadores de departamento.
  - a. Modifique VALID\_DEPTID para realizar la validación utilizando la tabla PL/SQL de valores de identificadores de departamento. Guarde y compile los cambios.

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  TYPE boolean_tabtype IS TABLE OF BOOLEAN
    INDEX BY BINARY_INTEGER;
  valid_departments boolean_tabtype;

  FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
    RETURN BOOLEAN IS
    dummy PLS_INTEGER;
  BEGIN
    RETURN valid_departments.exists(deptid);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN FALSE;
  END valid_deptid;
  ...

  PROCEDURE init_departments IS
  BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
      valid_departments(rec.department_id) := TRUE;
    END LOOP;
  END;
BEGIN
  init_departments;
END emp_pkg;
/
SHOW ERRORS

Package body created.

No errors.
```

- b. Pruebe el código llamando a ADD\_EMPLOYEE con el nombre James Bond en el departamento 15. ¿Qué sucede?

```
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)

BEGIN emp_pkg.add_employee('James', 'Bond', 15); END;

*
ERROR at line 1:
ORA-20204: Invalid department ID. Try again.
ORA-06512: at "ORA1.EMP_PKG", line 32
ORA-06512: at "ORA1.EMP_PKG", line 43
ORA-06512: at line 1
```

## Práctica 4: Soluciones (continuación)

**La operación de inserción para agregar un empleado ha fallado con una excepción, porque el departamento 15 no existe.**

- c. Inserte un departamento nuevo con identificador 15 y nombre Security para confirmar los cambios.

```
INSERT INTO departments (department_id, department_name)
VALUES (15, 'Security');
COMMIT;

1 row created.

Commit complete.
```

- d. Pruebe el código de nuevo llamando a ADD\_EMPLOYEE con el nombre James Bond en el departamento 15. ¿Qué sucede?

```
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)

BEGIN emp_pkg.add_employee('James', 'Bond', 15); END;

*
ERROR at line 1:
ORA-20204: Invalid department ID. Try again.
ORA-06512: at "ORA1.EMP_PKG", line 32
ORA-06512: at "ORA1.EMP_PKG", line 43
ORA-06512: at line 1
```

**La operación de inserción para agregar un empleado ha fallado con una excepción porque el departamento 15 no existe como entrada en la variable de estado de paquete de la tabla PL/SQL index-by.**

- e. Ejecute el procedimiento EMP\_PKG.INIT\_DEPARTMENTS para actualizar la tabla interna PL/SQL con los últimos datos del departamento.

```
EXECUTE EMP_PKG.INIT_DEPARTMENTS

PL/SQL procedure successfully completed.
```

- f. Compruebe el código llamando a ADD\_EMPLOYEE con el nombre de empleado James Bond que trabaja en el departamento 15. ¿Qué sucede?

```
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)

PL/SQL procedure successfully completed.
```

**La fila se inserta finalmente porque el registro del departamento 15 existe en la base de datos y en la tabla PL/SQL index-by del paquete, debido a la llamada de EMP\_PKG.INIT\_DEPARTMENTS, que refresca los datos de estado del paquete.**

## Práctica 4: Soluciones (continuación)

- g. Suprima al empleado James Bond y el departamento 15 de sus respectivas tablas, confirme los cambios y refresque los datos del departamento llamando al procedimiento EMP\_PKG.INIT\_DEPARTMENTS.

```
DELETE FROM employees
WHERE first_name = James AND last_name = Bond;
DELETE FROM departments WHERE department_id = 15;
COMMIT;
EXECUTE EMP_PKG.INIT_DEPARTMENTS

1 row deleted.
1 row deleted.
Commit complete.
PL/SQL procedure successfully completed.
```

5. Reorganice los subprogramas en el cuerpo de la especificación del paquete para que estén en secuencia alfabética.
- a. Edite la especificación del paquete y reorganice los subprogramas de forma alfabética. En *iSQL\*Plus*, cargue y compile la especificación del paquete. ¿Qué sucede?

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30);
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    deptid     employees.department_id%TYPE);
  PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE);
  FUNCTION get_employee(emp_id employees.employee_id%type)
    return employees%rowtype;
  FUNCTION get_employee(family_name employees.last_name%type)
    return employees%rowtype;
  PROCEDURE init_departments;
  PROCEDURE print_employee(emprec employees%rowtype);
END emp_pkg;
/
SHOW ERRORS
```

**Compila correctamente.**

**Nota:** Puede que el paquete ya tenga los subprogramas en orden alfabético.

## Práctica 4: Soluciones (continuación)

- b. Edite el cuerpo del paquete y reorganice todos los subprogramas de forma alfabética. En iSQL\*Plus, cargue y compile la especificación del paquete. ¿Qué sucede?

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  TYPE boolean_tabtype IS TABLE OF BOOLEAN
    INDEX BY BINARY_INTEGER;
  valid_departments boolean_tabtype;

  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30) IS
  BEGIN
    IF valid_deptid(deptid) THEN
      INSERT INTO employees(employee_id, first_name, last_name, email,
        job_id, manager_id, hire_date, salary, commission_pct,
        department_id)
        VALUES (employees_seq.NEXTVAL, first_name, last_name, email,
          job, mgr, TRUNC(SYSDATE), sal, comm, deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID. Try
again.');
```

again.');

```
    END IF;
  END add_employee;

  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    deptid     employees.department_id%TYPE) IS
    email      employees.email%TYPE;
  BEGIN
    email := UPPER(SUBSTR(first_name, 1, 1)||SUBSTR(last_name, 1, 7));
    add_employee(first_name, last_name, email, deptid => deptid);
  END;

  PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE) IS
  BEGIN
    SELECT salary, job_id
    INTO   sal, job
    FROM   employees
    WHERE  employee_id = empid;
  END get_employee;
```

## Práctica 4: Soluciones (continuación)

```
FUNCTION get_employee(emp_id employees.employee_id%type)
  return employees%rowtype IS
  emprec employees%rowtype;
BEGIN
  SELECT * INTO emprec
  FROM employees
  WHERE employee_id = emp_id;
  RETURN emprec;
END;

FUNCTION get_employee(family_name employees.last_name%type)
  return employees%rowtype IS
  emprec employees%rowtype;
BEGIN
  SELECT * INTO emprec
  FROM employees
  WHERE last_name = family_name;
  RETURN emprec;
END;

PROCEDURE init_departments IS
BEGIN
  FOR rec IN (SELECT department_id FROM departments)
  LOOP
    valid_departments(rec.department_id) := TRUE;
  END LOOP;
END;

PROCEDURE print_employee(emprec employees%rowtype) IS
BEGIN
  DBMS_OUTPUT.PUT_LINE(emprec.department_id || ' ' ||
    emprec.employee_id || ' ' ||
    emprec.first_name || ' ' ||
    emprec.last_name || ' ' ||
    emprec.job_id || ' ' ||
    emprec.salary);
END;

FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
  RETURN BOOLEAN IS
  dummy PLS_INTEGER;
BEGIN
  RETURN valid_departments.exists(deptid);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END valid_deptid;
```

## Práctica 4: Soluciones (continuación)

```
BEGIN
  init_departments;
END emp_pkg;
/
SHOW ERRORS
```

Warning: Package Body created with compilation errors.

Errors for PACKAGE BODY EMP\_PKG:

LINE/COL ERROR

```
-----
16/5      PL/SQL: Statement ignored
16/8      PLS-00313: 'VALID_DEPTID' not declared in this scope
```

**No compila correctamente porque se hace referencia a la función `VALID_DEPTID` antes de que se declare.**

- c. Corrija el error de compilación utilizando una declaración anticipada en el cuerpo para la referencia de subprograma incorrecta. Cargue y vuelva a crear el cuerpo del paquete.  
¿Qué sucede?

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  TYPE boolean_tabtype IS TABLE OF BOOLEAN
    INDEX BY BINARY_INTEGER;
  valid_departments boolean_tabtype;

  FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
    RETURN BOOLEAN;

  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30) IS
  BEGIN
    IF valid_deptid(deptid) THEN
      INSERT INTO employees(employee_id, first_name, last_name, email,
        job_id, manager_id, hire_date, salary, commission_pct, department_id)
        VALUES (employees_seq.NEXTVAL, first_name, last_name, email,
        job, mgr, TRUNC(SYSDATE), sal, comm, deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204,
        'Invalid department ID. Try again.');

END IF;



END add_employee;



...


```

## Práctica 4: Soluciones (continuación)

```
FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
RETURN BOOLEAN IS
    dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

BEGIN
    init_departments;
END emp_pkg;
/
SHOW ERRORS

Package body created.

No errors.
```

Una declaración anticipada para la función **VALID\_DEPTID** permite que el cuerpo del paquete se compile correctamente.

Si tiene tiempo, realice el siguiente ejercicio:

6. Ajuste el cuerpo del paquete EMP\_PKG y vuelva a crearlo.
  - a. Consulte el diccionario de datos para ver el origen para el cuerpo EMP\_PKG.

```
SELECT text
FROM user_source
WHERE name = 'EMP_PKG'
AND type = 'PACKAGE BODY'
ORDER BY line;
```

TEXT
PACKAGE BODY emp_pkg IS
TYPE boolean_tabtype IS TABLE OF BOOLEAN
INDEX BY BINARY_INTEGER;
valid_departments boolean_tabtype;
BEGIN
init_departments;
END emp_pkg;

100 rows selected.

## Práctica 4: Soluciones (continuación)

- b. Inicie una ventana de comandos y ejecute la utilidad de línea de comandos WRAP para ajustar el cuerpo del paquete EMP\_PKG. Asigne una extensión .plb al archivo de salida.

**Indicación:** Copie el archivo (que ha guardado en el paso 5c) que contiene el cuerpo del paquete en un archivo denominado emp\_pkb\_b.sql.

```
WRAP INAME=emp_pkg_b.sql
```

```
PL/SQL Wrapper: Release 9,2.00,4.0 - Production on Mon Feb 16 20:25:29
2004
```

```
Copyright (c) Oracle Corporation 1993, 2001. All Rights Reserved.
```

```
Processing emp_pkg_b.sql to emp_pkg_b.plb
```

- c. Con iSQL\*Plus, cargue y ejecute el archivo .plb que contiene el origen ajustado.

```
CREATE OR REPLACE PACKAGE BODY emp_pkg wrapped
```

```
0
```

```
abcd
```

```
abcd
```

```
abcd
```

```
abcd
```

```
abcd
```

```
abcd
```

```
abcd
```

```
abcd
```

```
abcd
```

```
abcd
```

```
abcd
```

```
abcd
```

```
abcd
```

```
abcd
```

```
abcd
```

```
:
```

```
:
```

```
be 4 0
```

```
67 3 0
```

```
15 2 0
```

```
133 6 0
```

```
5 1 0
```

```
5d 3 0
```

```
193 1 8
```

```
0
```

```
/
```

```
SHOW ERRORS
```

```
Package body created.
```

```
No errors.
```



## Práctica 4: Soluciones (continuación)

- d. Consulte el diccionario de datos para mostrar de nuevo el origen para el cuerpo del paquete EMP\_PKG. ¿Son legibles las líneas del código de origen original?

```
SELECT text
FROM user_source
WHERE name = 'EMP_PKG'
AND type = 'PACKAGE BODY'
ORDER BY line;
```

TEXT
PACKAGE BODY emp_pkg wrapped 0 abcd abcd abcd abcd abcd abcd abcd abcd abcd abcd abcd abcd abcd abcd abcd abcd 3 b 9200000 1 4 0 44 2 :e;
121 2 11b 11c 1 129 1 133 1 13b 1 143 1 14c 1 150 2 14f 150 1 14b 2 158 15b 1 14c 1 150 1 151 151 1 152 152 1 153 153 1 154 154 1 155 155 1 156 156 1 157 157 1 158 158 1 159 159 1 160 160 1 161 161 1 162 162 1 163 163 1 164 164 1 165 165 1 166 166 1 167 167 1 168 168 1 169 169 1 170 170 1 171 171 1 172 172 1 173 173 1 174 174 1 175 175 1 176 176 1 177 177 1 178 178 1 179 179 1 180 180 1 181 181 1 182 182 1 183 183 1 184 184 1 185 185 1 186 186 1 187 187 1 188 188 1 189 189 1 190 190 1 191 191 1 192 192 1 193 193 1 194 194 1 195 195 1 196 196 1 197 197 1 198 198 1 199 199 1 200 200 1 201 201 1 202 202 1 203 203 1 204 204 1 205 205 1 206 206 1 207 207 1 208 208 1 209 209 1 210 210 1 211 211 1 212 212 1 213 213 1 214 214 1 215 215 1 216 216 1 217 217 1 218 218 1 219 219 1 220 220 1 221 221 1 222 222 1 223 223 1 224 224 1 225 225 1 226 226 1 227 227 1 228 228 1 229 229 1 230 230 1 231 231 1 232 232 1 233 233 1 234 234 1 235 235 1 236 236 1 237 237 1 238 238 1 239 239 1 240 240 1 241 241 1 242 242 1 243 243 1 244 244 1 245 245 1 246 246 1 247 247 1 248 248 1 249 249 1 250 250 1 251 251 1 252 252 1 253 253 1 254 254 1 255 255 1 256 256 1 257 257 1 258 258 1 259 259 1 260 260 1 261 261 1 262 262 1 263 263 1 264 264 1 265 265 1 266 266 1 267 267 1 268 268 1 269 269 1 270 270 1 271 271 1 272 272 1 273 273 1 274 274 1 275 275 1 276 276 1 277 277 1 278 278 1 279 279 1 280 280 1 281 281 1 282 282 1 283 283 1 284 284 1 285 285 1 286 286 1 287 287 1 288 288 1 289 289 1 290 290 1 291 291 1 292 292 1 293 293 1 294 294 1 295 295 1 296 296 1 297 297 1 298 298 1 299 299 1 300 300 1 301 301 1 302 302 1 303 303 1 304 304 1 305 305 1 306 306 1 307 307 1 308 308 1 309 309 1 310 310 1 311 311 1 312 312 1 313 313 1 314 314 1 315 315 1 316 316 1 317 317 1 318 318 1 319 319 1 320 320 1 321 321 1 322 322 1 323 323 1 324 324 1 325 325 1 326 326 1 327 327 1 328 328 1 329 329 1 330 330 1 331 331 1 332 332 1 333 333 1 334 334 1 335 335 1 336 336 1 337 337 1 338 338 1 339 339 1 340 340 1 341 341 1 342 342 1 343 343 1 344 344 1 345 345 1 346 346 1 347 347 1 348 348 1 349 349 1 350 350 1 351 351 1 352 352 1 353 353 1 354 354 1 355 355 1 356 356 1 357 357 1 358 358 1 359 359 1 360 360 1 361 361 1 362 362 1 363 363 1 364 364 1 365 365 1 366 366 1 367 367 1 368 368 1 369 369 1 370 370 1 371 371 1 372 372 1 373 373 1 374 374 1 375 375 1 376 376 1 377 377 1 378 378 1 379 379 1 380 380 1 381 381 1 382 382 1 383 383 1 384 384 1 385 385 1 386 386 1 387 387 1 388 388 1 389 389 1 390 390 1 391 391 1 392 392 1 393 393 1 394 394 1 395 395 1 396 396 1 397 397 1 398 398 1 399 399 1 400 400 1 401 401 1 402 402 1 403 403 1 404 404 1 405 405 1 406 406 1 407 407 1 408 408 1 409 409 1 410 410 1 411 411 1 412 412 1 413 413 1 414 414 1 415 415 1 416 416 1 417 417 1 418 418 1 419 419 1 420 420 1 421 421 1 422 422 1 423 423 1 424 424 1 425 425 1 426 426 1 427 427 1 428 428 1 429 429 1 430 430 1 431 431 1 432 432 1 433 433 1 434 434 1 435 435 1 436 436 1 437 437 1 438 438 1 439 439 1 440 440 1 441 441 1 442 442 1 443 443 1 444 444 1 445 445 1 446 446 1 447 447 1 448 448 1 449 449 1 450 450 1 451 451 1 452 452 1 453 453 1 454 454 1 455 455 1 456 456 1 457 457 1 458 458 1 459 459 1 460 460 1 461 461 1 462 462 1 463 463 1 464 464 1 465 465 1 466 466 1 467 467 1 468 468 1 469 469 1 470 470 1 471 471 1 472 472 1 473 473 1 474 474 1 475 475 1 476 476 1 477 477 1 478 478 1 479 479 1 480 480 1 481 481 1 482 482 1 483 483 1 484 484 1 485 485 1 486 486 1 487 487 1 488 488 1 489 489 1 490 490 1 491 491 1 492 492 1 493 493 1 494 494 1 495 495 1 496 496 1 497 497 1 498 498 1 499 499 1 500 500 1 501 501 1 502 502 1 503 503 1 504 504 1 505 505 1 506 506 1 507 507 1 508 508 1 509 509 1 510 510 1 511 511 1 512 512 1 513 513 1 514 514 1 515 515 1 516 516 1 517 517 1 518 518 1 519 519 1 520 520 1 521 521 1 522 522 1 523 523 1 524 524 1 525 525 1 526 526 1 527 527 1 528 528 1 529 529 1 530 530 1 531 531 1 532 532 1 533 533 1 534 534 1 535 535 1 536 536 1 537 537 1 538 538 1 539 539 1 540 540 1 541 541 1 542 542 1 543 543 1 544 544 1 545 545 1 546 546 1 547 547 1 548 548 1 549 549 1 550 550 1 551 551 1 552 552 1 553 553 1 554 554 1 555 555 1 556 556 1 557 557 1 558 558 1 559 559 1 560 560 1 561 561 1 562 562 1 563 563 1 564 564 1 565 565 1 566 566 1 567 567 1 568 568 1 569 569 1 570 570 1 571 571 1 572 572 1 573 573 1 574 574 1 575 575 1 576 576 1 577 577 1 578 578 1 579 579 1 580 580 1 581 581 1 582 582 1 583 583 1 584 584 1 585 585 1 586 586 1 587 587 1 588 588 1 589 589 1 590 590 1 591 591 1 592 592 1 593 593 1 594 594 1 595 595 1 596 596 1 597 597 1 598 598 1 599 599 1 600 600 1 601 601 1 602 602 1 603 603 1 604 604 1 605 605 1 606 606 1 607 607 1 608 608 1 609 609 1 610 610 1 611 611 1 612 612 1 613 613 1 614 614 1 615 615 1 616 616 1 617 617 1 618 618 1 619 619 1 620 620 1 621 621 1 622 622 1 623 623 1 624 624 1 625 625 1 626 626 1 627 627 1 628 628 1 629 629 1 630 630 1 631 631 1 632 632 1 633 633 1 634 634 1 635 635 1 636 636 1 637 637 1 638 638 1 639 639 1 640 640 1 641 641 1 642 642 1 643 643 1 644 644 1 645 645 1 646 646 1 647 647 1 648 648 1 649 649 1 650 650 1 651 651 1 652 652 1 653 653 1 654 654 1 655 655 1 656 656 1 657 657 1 658 658 1 659 659 1 660 660 1 661 661 1 662 662 1 663 663 1 664 664 1 665 665 1 666 666 1 667 667 1 668 668 1 669 669 1 670 670 1 671 671 1 672 672 1 673 673 1 674 674 1 675 675 1 676 676 1 677 677 1 678 678 1 679 679 1 680 680 1 681 681 1 682 682 1 683 683 1 684 684 1 685 685 1 686 686 1 687 687 1 688 688 1 689 689 1 690 690 1 691 691 1 692 692 1 693 693 1 694 694 1 695 695 1 696 696 1 697 697 1 698 698 1 699 699 1 700 700 1 701 701 1 702 702 1 703 703 1 704 704 1 705 705 1 706 706 1 707 707 1 708 708 1 709 709 1 710 710 1 711 711 1 712 712 1 713 713 1 714 714 1 715 715 1 716 716 1 717 717 1 718 718 1 719 719 1 720 720 1 721 721 1 722 722 1 723 723 1 724 724 1 725 725 1 726 726 1 727 727 1 728 728 1 729 729 1 730 730 1 731 731 1 732 732 1 733 733 1 734 734 1 735 735 1 736 736 1 737 737 1 738 738 1 739 739 1 740 740 1 741 741 1 742 742 1 743 743 1 744 744 1 745 745 1 746 746 1 747 747 1 748 748 1 749 749 1 750 750 1 751 751 1 752 752 1 753 753 1 754 754 1 755 755 1 756 756 1 757 757 1 758 758 1 759 759 1 760 760 1 761 761 1 762 762 1 763 763 1 764 764 1 765 765 1 766 766 1 767 767 1 768 768 1 769 769 1 770 770 1 771 771 1 772 772 1 773 773 1 774 774 1 775 775 1 776 776 1 777 777 1 778 778 1 779 779 1 780 780 1 781 781 1 782 782 1 783 783 1 784 784 1 785 785 1 786 786 1 787 787 1 788 788 1 789 789 1 790 790 1 791 791 1 792 792 1 793 793 1 794 794 1 795 795 1 796 796 1 797 797 1 798 798 1 799 799 1 800 800 1 801 801 1 802 802 1 803 803 1 804 804 1 805 805 1 806 806 1 807 807 1 808 808 1 809 809 1 810 810 1 811 811 1 812 812 1 813 813 1 814 814 1 815 815 1 816 816 1 817 817 1 818 818 1 819 819 1 820 820 1 821 821 1 822 822 1 823 823 1 824 824 1 825 825 1 826 826 1 827 827 1 828 828 1 829 829 1 830 830 1 831 831 1 832 832 1 833 833 1 834 834 1 835 835 1 836 836 1 837 837 1 838 838 1 839 839 1 840 840 1 841 841 1 842 842 1 843 843 1 844 844 1 845 845 1 846 846 1 847 847 1 848 848 1 849 849 1 850 850 1 851 851 1 852 852 1 853 853 1 854 854 1 855 855 1 856 856 1 857 857 1 858 858 1 859 859 1 860 860 1 861 861 1 862 862 1 863 863 1 864 864 1 865 865 1 866 866 1 867 867 1 868 868 1 869 869 1 870 870 1 871 871 1 872 872 1 873 873 1 874 874 1 875 875 1 876 876 1 877 877 1 878 878 1 879 879 1 880 880 1 881 881 1 882 882 1 883 883 1 884 884 1 885 885 1 886 886 1 887 887 1 888 888 1 889 889 1 890 890 1 891 891 1 892 892 1 893 893 1 894 894 1 895 895 1 896 896 1 897 897 1 898 898 1 899 899 1 900 900 1 901 901 1 902 902 1 903 903 1 904 904 1 905 905 1 906 906 1 907 907 1 908 908 1 909 909 1 910 910 1 911 911 1 912 912 1 913 913 1 914 914 1 915 915 1 916 916 1 917 917 1 918 918 1 919 919 1 920 920 1 921 921 1 922 922 1 923 923 1 924 924 1 925 925 1 926 926 1 927 927 1 928 928 1 929 929 1 930 930 1 931 931 1 932 932 1 933 933 1 934 934 1 935 935 1 936 936 1 937 937 1 938 938 1 939 939 1 940 940 1 941 941 1 942 942 1 943 943 1 944 944 1 945 945 1 946 946 1 947 947 1 948 948 1 949 949 1 950 950 1 951 951 1 952 952 1 953 953 1 954 954 1 955 955 1 956 956 1 957 957 1 958 958 1 959 959 1 960 960 1 961 961 1 962 962 1 963 963 1 964 964 1 965 965 1 966 966 1 967 967 1 968 968 1 969 969 1 970 970 1 971 971 1 972 972 1 973 973 1 974 974 1 975 975 1 976 976 1 977 977 1 978 978 1 979 979 1 980 980 1 981 981 1 982 982 1 983 983 1 984 984 1 985 985 1 986 986 1 987 987 1 988 988 1 989 989 1 990 990 1 991 991 1 992 992 1 993 993 1 994 994 1 995 995 1 996 996 1 997 997 1 998 998 1 999 999 1 1000 1000 1 1001 1001 1 1002 1002 1 1003 1003 1 1004 1004 1 1005 1005 1 1006 1006 1 1007 1007 1 1008 1008 1 1009 1009 1 1010 1010 1 1011 1011 1 1012 1012 1 1013 1013 1 1014 1014 1 1015 1015 1 1016 1016 1 1017 1017 1 1018 1018 1 1019 1019 1 1020 1020 1 1021 1021 1 1022 1022 1 1023 1023 1 1024 1024 1 1025 1025 1 1026 1026 1 1027 1027 1 1028 1028 1 1029 1029 1 1030 1030 1 1031 1031 1 1032 1032 1 1033 1033 1 1034 1034 1 1035 1035 1 1036 1036 1 1037 1037 1 1038 1038 1 1039 1039 1 1040 1040 1 1041 1041 1 1042 1042 1 1043 1043 1 1044 1044 1 1045 1045 1 1046 1046 1 1047 1047 1 1048 1048 1 1049 1049 1 1050 1050 1 1051 1051 1 1052 1052 1 1053 1053 1 1054 1054 1 1055 1055 1 1056 1056 1 1057 1057 1 1058 1058 1 1059 1059 1 1060 1060 1 1061 1061 1 1062 1062 1 1063 1063 1 1064 1064 1 1065 1065 1 1066 1066 1 1067 1067 1 1068 1068 1 1069 1069 1 1070 1070 1 1071 1071 1 1072 1072 1 1073 1073 1 1074 1074 1 1075 1075 1 1076 1076 1 1077 1077 1 1078 1078 1 1079 1079 1 1080 1080 1 1081 1081 1 1082 1082 1 1083 1083 1 1084 1084 1 1085 1085 1 1086 1086 1 1087 1087 1 1088 1088 1 1089 1089 1 1090 1090 1 1091 1091 1 1092 1092 1 1093 1093 1 1094 1094 1 1095 1095 1 1096 1096 1 1097 1097 1 1098 1098 1 1099 1099 1 1100 1100 1 1101 1101 1 1102 1102 1 1103 1103 1 1104 1104 1 1105 1105 1 1106 1106 1 1107 1107 1 1108 1108 1 1109 1109 1 1110 1110 1 1111 1111 1 1112 1112 1 1113 1113 1 1114 1114 1 1115 1115 1 1116 1116 1 1117 1117 1 1118 1118 1 1119 1119 1 1120 1120 1 1121 1121 1 1122 1122 1 1123 1123 1 1124 1124 1 1125 1125 1 1126 1126 1 1127 1127 1 1128 1128 1 1129 1129 1 1130 1130 1 1131 1131 1 1132 1132 1 1133 1133 1 1134 1134 1 1135 1135 1 1136 1136 1 1137 1137 1 1138 1138 1 1139 1139 1 1140 1140 1 1141 1141 1 1142 1142 1 1143 1143 1 1144 1144 1 1145 1145 1 1146 1146 1 1147 1147 1 1148 1148 1 1149 1149 1 1150 1150 1 1151 1151 1 1152 1152 1 1153 1153 1 1154 1154 1 1155 1155 1 1156 1156 1 1157 1157 1 1158 1158 1 1159 1159 1 1160 1160 1 1161 1161 1 1162 1162 1 1163 1163 1 1164 1164 1 1165 1165 1 1166 1166 1 1167 1167 1 1168 1168 1 1169 1169 1 1170 1170 1 1171 1171 1 1172 1172 1 1173 1173 1 1174 1174 1 1175 1175 1 1176 1176 1 1177 1177 1 1178 1178 1 1179 1179 1 1180 1180 1 1181 1181 1 1182 1182 1 1183 1183 1 1184 1184 1 1185 1185 1 1186 1186 1 1187 1187 1 1188 1188 1 1189 1189 1 1190 1190 1 1191 1191 1 1192 1192 1 1193 1193 1 1194 1194 1 1195 1195 1 1196 1196 1 1197 1197 1 1198 1198 1 1199 1199 1 1200 1200 1 1201 1201 1 1202 1202 1 1203 1203 1 1204 1204 1 1205 1205 1 1206 1206 1 1207 1207 1 1208 1208 1 1209 1209 1 1210 1210 1 1211 1211 1 1212 1212 1 1213 1213 1 1214 1214 1 1215 1215 1 1216 1216 1 1217 1217 1 1218 1218 1 1219 1219 1 1220 1220 1 1221 1221 1 1222 1222 1 1223 1223 1 1224 1224 1 1225 1225 1 1226 1226 1 1227 1227 1 1228 1228 1 1229 1229 1 1230 1230 1 1231 1231 1 1232 1232 1 1233 1233 1 1234 1234 1 1235 1235 1 1236 1236 1 1237 1237 1 1238 1238 1 1239 1239 1 1240 1240 1 1241 1241 1 1242 1242 1 1243 1243 1 1244 1244 1 1245 1245 1 1246 1246 1 1247 1247 1 1248 1248 1 1249 1249 1 1250 1250 1 1251 1251 1 1252 1252 1 1253 1253 1 1254 1254 1 1255 1255 1 1256 1256 1 1257 1257 1 1258 1258 1 1259 1259 1 1260 1260 1 1261 1261 1 1262 1262 1 1263 1263 1 1264 1264 1 1265 1265 1 1266 1266 1 1267 1267 1 1268 1268 1 1269 1269 1 1270 1270 1 1271 1271 1 1272 1272 1 1273 1273 1 1274 1274 1 1275 1275 1 1276 1276 1 1277 1277 1 1278 1278 1 1279 1279 1 1280 1280 1 1281 1281 1 1282 1282 1 1283 1283 1 1284 1284 1 1285 1285 1 1286 1286 1 1287 1287 1 1288 1288 1 1289 1289 1 1290 1290 1 1291 1291 1 129

## Práctica 5: Soluciones

1. Cree un procedimiento denominado `EMPLOYEE_REPORT` que genera un informe de empleados en un archivo del sistema operativo utilizando el paquete `UTL_FILE`. Este informe generará una lista de los empleados que han excedido el salario medio de su departamento.
  - a. El programa debe aceptar dos parámetros. El primero es el directorio de salida. El segundo es el nombre del archivo de texto escrito.  
**Nota:** Utilice el valor de la ubicación del directorio `UTL_FILE`. Agregue una sección de manejo de excepciones para manejar los errores que se pueden encontrar al utilizar el paquete `UTL_FILE`.

A continuación se muestra una salida de ejemplo del archivo de informe:

```
Employees who earn more than average salary:
REPORT GENERATED ON 26-FEB-04
Hartstein                20          $13,000.00
Raphaely                 30          $11,000.00
Marvis                   40          $6,500.00
...
*** END OF REPORT ***
```

```
CREATE OR REPLACE PROCEDURE employee_report(
  dir IN VARCHAR2, filename IN VARCHAR2) IS
  f UTL_FILE.FILE_TYPE;
  CURSOR avg_csr IS
    SELECT last_name, department_id, salary
    FROM employees outer
    WHERE salary > (SELECT AVG(salary)
                     FROM employees inner
                     GROUP BY outer.department_id)
    ORDER BY department_id;
BEGIN
  f := UTL_FILE.FOPEN(dir, filename, 'w');
  UTL_FILE.PUT_LINE(f, 'Employees who earn more than average salary: ');
  UTL_FILE.PUT_LINE(f, 'REPORT GENERATED ON ' || SYSDATE);
  UTL_FILE.NEW_LINE(f);
  FOR emp IN avg_csr
  LOOP
    UTL_FILE.PUT_LINE(f,
      RPAD(emp.last_name, 30) || ' ' ||
      LPAD(NVL(TO_CHAR(emp.department_id, '9999'), '-'), 5) || ' ' ||
      LPAD(TO_CHAR(emp.salary, '$99,999.00'), 12));
  END LOOP;
  UTL_FILE.NEW_LINE(f);
  UTL_FILE.PUT_LINE(f, '*** END OF REPORT ***');
  UTL_FILE.FCLOSE(f);
END employee_report;
/
```

Procedure created.

## Práctica 5: Soluciones (continuación)

- b. Llame al programa, utilizando el segundo parámetro con un nombre como `sal_rptxx.txt` en el que `xx` representa el número de usuario (por ejemplo, 01, 15, etc.).

```
EXECUTE employee_report('UTL_FILE','sal_rpt01.txt')
```

```
PL/SQL Procedure sucessfully completed.
```

**Nota:** Los datos muestran el apellido, el identificador de departamento y el salario del empleado.

Pídale al instructor que le dé instrucciones sobre cómo obtener el archivo de informe del servidor utilizando la utilidad Putty PSFTP.

**Cuando utilice PSFTP para recuperar el archivo generado, éste debe contener algo parecido al siguiente ejemplo:**

```
Employees who earn more than average salary:
```

```
REPORT GENERATED ON 16-FEB-04
```

Hartstein	20	\$13,000.00
Raphaely	30	\$11,000.00
Mavris	40	\$6,500.00
Weiss	50	\$8,000.00
Kaufling	50	\$7,900.00
Fripp	50	\$8,200.00
Vollman	50	\$6,500.00
Hunold	60	\$9,000.00
Baer	70	\$10,000.00
Russell	80	\$14,000.00
Bernstein	80	\$9,500.00
Olsen	80	\$8,000.00
:		
:		
Errazuriz	80	\$12,000.00
Zlotkey	80	\$10,500.00
Cambrault	80	\$11,000.00
King	90	\$24,000.00
Kochhar	90	\$17,000.00
De Haan	90	\$17,000.00
Greenberg	100	\$12,000.00
Faviet	100	\$9,000.00
Chen	100	\$8,200.00
Sciarra	100	\$7,700.00
Urman	100	\$7,800.00
Popp	100	\$6,900.00
Higgins	110	\$12,000.00
Gietz	110	\$8,300.00
Grant	-	\$7,000.00

```
*** END OF REPORT ***
```

## Práctica 5: Soluciones (continuación)

2. Cree un procedimiento nuevo denominado WEB\_EMPLOYEE\_REPORT que genere los mismos datos que EMPLOYEE\_REPORT.
  - a. En primer lugar, ejecute SET SERVEROUTPUT ON y a continuación  
http.print('hello') por último, ejecute OWA\_UTIL.SHOWPAGE. Los mensajes de excepción generados se pueden ignorar.

```
SET SERVEROUTPUT ON
EXECUTE HTTP.PRINT('hello')
EXECUTE OWA_UTIL.SHOWPAGE

BEGIN http.print('hello'); END;

*

ERROR at line 1:
ORA-06502: PL/SQL: numeric or value error
ORA-06512: at "SYS.OWA_UTIL", line 325
ORA-06512: at "SYS.HTTP", line 1322
ORA-06512: at "SYS.HTTP", line 1397
ORA-06512: at "SYS.HTTP", line 1684
ORA-06512: at line 1
PL/SQL procedure successfully completed.
```

**Estos pasos se realizan para garantizar que los mensajes no se vuelven a generar. Sin embargo, recuerde que el paquete HTTP está destinado a utilizarse en el contexto de Oracle HTTP Server y no con iSQL\*Plus.**

- b. Escriba el procedimiento WEB\_EMPLOYEE\_REPORT con el paquete HTTP para generar un informe HTML de empleados con un salario mayor que la media de su departamento. Si conoce HTML, cree una tabla HTML, de lo contrario, límitese a crear líneas de datos.  
**Indicación:** Copie la definición del cursor y el bucle FOR del procedimiento EMPLOYEE\_REPORT para la estructura básica del informe Web.

```
CREATE OR REPLACE PROCEDURE web_employee_report IS
CURSOR avg_csr IS
  SELECT last_name, department_id, salary
  FROM employees outer
  WHERE salary > (SELECT AVG(salary)
                  FROM employees inner
                  GROUP BY outer.department_id)
  ORDER BY department_id;
```

## Práctica 5: Soluciones (continuación)

```
BEGIN
  http.htmlopen;
  http.headopen;
  http.title('Employee Salary Report');
  http.headclose;
  http.bodyopen;
  http.header(1, 'Employees who earn more than average salary');
  http.print('REPORT GENERATED ON' || to_char(SYSDATE, 'DD-MON-YY'));
  http.br;
  http.hr;
  http.tableOpen;
  http.tablerowOpen;
  http.tableHeader('Last Name');
  http.tableHeader('Department');
  http.tableHeader('Salary');
  http.tablerowclose;

  FOR emp IN avg_csr
  LOOP
    http.tablerowOpen;
    http.tabledata(emp.last_name);
    http.tabledata(NVL(TO_CHAR(emp.department_id, '9999'), '-'));
    http.tabledata(TO_CHAR(emp.salary, '$99,999.00'));
    http.tablerowclose;
  END LOOP;

  http.tableclose;
  http.hr;
  http.print('*** END OF REPORT ***');
  http.bodyclose;
  http.htmlclose;
END web_employee_report;
/
show errors

Procedure created.

No errors.
```

- c. Ejecute el procedimiento con *iSQL\*Plus* para generar los datos HTML en un buffer del servidor y ejecute el procedimiento `OWA_UTIL.SHOWPAGE` para mostrar el contenido del buffer. Recuerde que `SERVEROUTPUT` debe estar en `ON` antes de ejecutar el código.

```
EXECUTE web_employee_report
EXECUTE owa_util.showpage

PL/SQL procedure successfully completed.

:
```

## Práctica 5: Soluciones (continuación)

```
<HTML>
<HEAD>
<TITLE>Employee Salary Report</TITLE>
</HEAD>
<BODY>
<H1>Employees who earn more than average salary</H1>
REPORT GENERATED ON16-FEB-04
<BR>
<HR>
<TABLE >
<TR>
<TH>Last Name</TH>
<TH>Department</TH>
<TH>Salary</TH>
</TR>
<TR>
<TD>Hartstein</TD>
<TD> 20</TD>
<TD> $13,000.00</TD>
</TR>
<TR>
<TD>Raphaely</TD>
<TD> 30</TD>
<TD> $11,000.00</TD>
</TR>
<TR>
<TD>Mavris</TD>
<TD> 40</TD>
<TD> $6,500.00</TD>
</TR>
<TR>
<TD>Weiss</TD>
<TD> 50</TD>
<TD> $8,000.00</TD>
</TR>
<TR>
<TD>Kaufling</TD>
<TD> 50</TD>
<TD> $7,900.00</TD>
</TR>
<TR>
<TD>Fripp</TD>
<TD> 50</TD>
<TD> $8,200.00</TD>
</TR>
<TR>
<TD>Vollman</TD>
<TD> 50</TD>
<TD> $6,500.00</TD>
</TR>
```

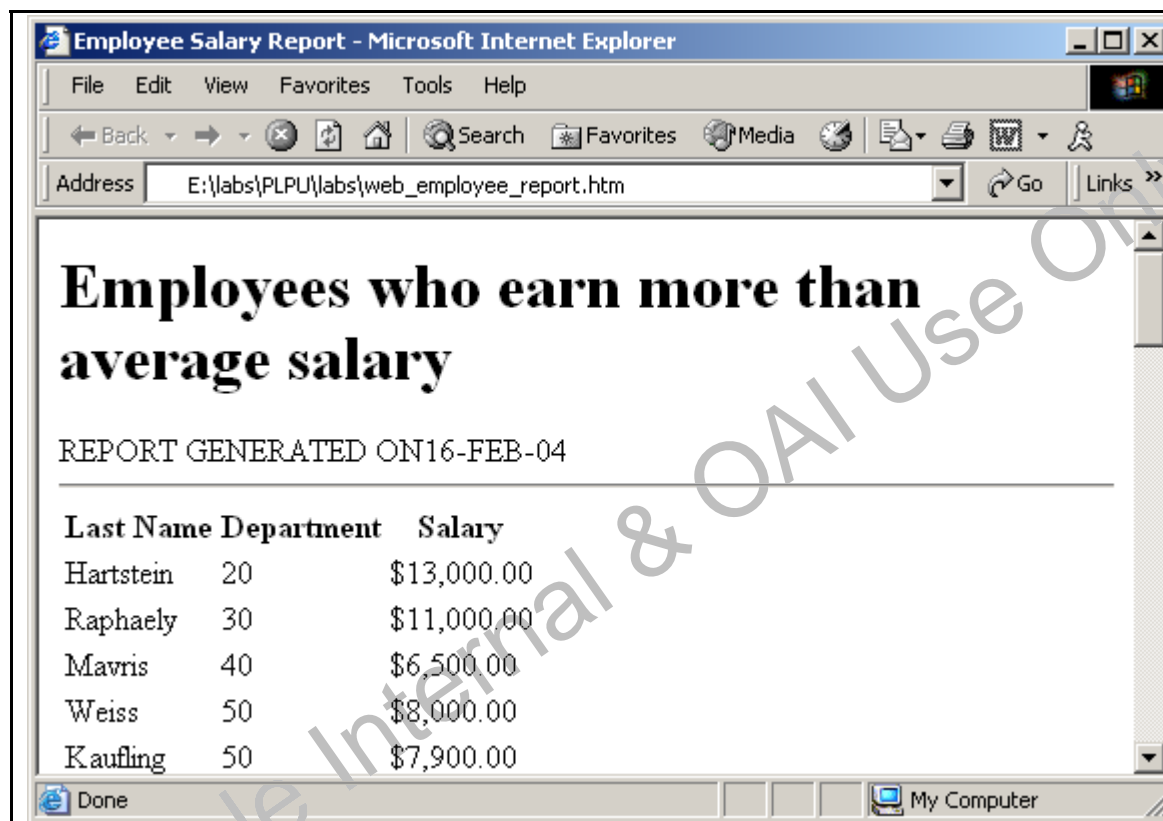
## Práctica 5: Soluciones (continuación)

```
<TR>
<TD>Hunold</TD>
<TD> 60</TD>
<TD> $9,000.00</TD>
</TR>
<TR>
<TD>Baer</TD>
<TD> 70</TD>
<TD> $10,000.00</TD>
</TR>
<TR> <TD>Russell</TD> <TD> 80</TD> <TD> $14,000.00</TD> </TR> <TR>
<TD>Bernstein</TD> <TD> 80</TD> <TD>
$9,500.00</TD> </TR> <TR> <TD>Olsen</TD> <TD> 80</TD> <TD> $8,000.00</TD>
</TR> <TR> <TD>Vishney</TD> <TD> 80</TD> <TD> $10,500.00</TD>
</TR> <TR> <TD>Sewall</TD> <TD> 80</TD> <TD> $7,000.00</TD> </TR> <TR>
<TD>Doran</TD> <TD> 80</TD> <TD>
$7,500.00</TD> </TR> <TR> <TD>Smith</TD> <TD> 80</TD> <TD> $8,000.00</TD>
</TR> <TR> <TD>McEwen</TD> <TD> 80</TD> <TD> $9,000.00</TD>
</TR> <TR> <TD>Sully</TD> <TD> 80</TD> <TD> $9,500.00</TD> </TR> <TR>
<TD>King</TD> <TD> 80</TD> <TD>
$10,000.00</TD> </TR> <TR> <TD>Tuvault</TD> <TD> 80</TD> <TD>
$7,000.00</TD> </TR> <TR> <TD>Cambrault</TD> <TD> 80</TD> <TD>
$7,500.00</TD>
</TR> <TR> <TD>Bates</TD> <TD> 80</TD> <TD> $7,300.00</TD> </TR> <TR>
<TD>Smith</TD> <TD> 80</TD> <TD>
$7,400.00</TD> </TR> <TR> <TD>Fox</TD> <TD> 80</TD> <TD> $9,600.00</TD>
</TR> <TR> <TD>Bloom</TD> <TD> 80</TD> <TD> $10,000.00</TD> </TR>
<TR> <TD>Ozer</TD> <TD> 80</TD> <TD> $11,500.00</TD> </TR> <TR>
<TD>Ande</TD> <TD> 80</TD> <TD> $6,400.00</TD> </TR> <TR> <TD>Lee</TD>
<TD>
80</TD> <TD> $6,800.00</TD> </TR> <TR> <TD>Marvins</TD> <TD> 80</TD> <TD>
$7,200.00</TD> </TR> <TR>
<TD>Greene</TD> <TD> 80</TD> <TD> $9,500.00</TD> </TR> <TR>
<TD>Livingston</TD> <TD> 80</TD> <TD> $8,400.00</TD> </TR> <TR>
<TD>Taylor</TD> <TD>
80</TD> <TD> $8,600.00</TD> </TR> <TR> <TD>Hutton</TD> <TD> 80</TD> <TD>
$8,800.00</TD> </TR>
<TR> <TD>Abel</TD> <TD> 80</TD> <TD> $11,000.00</TD> </TR> <TR>
<TD>Hall</TD> <TD> 80</TD> <TD> $9,000.00</TD> </TR> <TR> <TD>Tucker</TD>
<TD>
80</TD> <TD> $10,000.00</TD> </TR> <TR> <TD>Partners</TD> <TD> 80</TD>
<TD> $13,500.00</TD> </TR> <TR>
<TD>Errazuriz</TD> <TD> 80</TD> <TD> $12,000.00</TD> </TR> <TR>
<TD>Zlotkey</TD> <TD> 80</TD> <TD>
$10,500.00</TD> </TR> <TR> <TD>Cambrault</TD> <TD> 80</TD> <TD>
$11,000.00</TD> </TR> <TR> <TD>King</TD> <TD> 90</TD> <TD>
$24,000.00</TD> </TR>
<TR> <TD>Kochhar</TD> <TD> 90</TD> <TD> $17,000.00</TD> </TR> <TR> <TD>De
Haan</TD> <TD> 90</TD> <TD>
$17,000.00</TD> </TR> <TR> <TD>Greenberg</TD> <TD> 100</TD> <TD>
$12,000.00</TD> </TR>
```

## Práctica 5: Soluciones (continuación)

```
<TR> <TD>Faviet</TD> <TD> 100</TD> <TD> $9,000.00</TD>
</TR> <TR> <TD>Chen</TD> <TD> 100</TD> <TD> $8,200.00</TD> </TR> <TR>
<TD>Sciarra</TD> <TD> 100</TD> <TD>
$7,700.00</TD> </TR> <TR> <TD>Urman</TD> <TD> 100</TD> <TD>
$7,800.00</TD> </TR> <TR> <TD>Popp</TD> <TD> 100</TD> <TD> $6,900.00</TD>
</TR>
<TR> <TD>Higgins</TD> <TD> 110</TD> <TD> $12,000.00</TD> </TR> <TR>
<TD>Gietz</TD> <TD> 110</TD> <TD>
$8,300.00</TD> </TR> <TR> <TD>Grant</TD> <TD>-</TD> <TD> $7,000.00</TD>
</TR> </TABLE> <HR> *** END OF REPORT *** </BODY> </HTML>
PL/SQL procedure successfully completed.
```

- d. Cree un archivo HTML denominado `web_employee_report.htm` que contenga el texto del resultado de salida que seleccione y copie desde la etiqueta de apertura `<HTML>` a la etiqueta de cierre `</HTML>`. Pegue el texto copiado en el archivo y guárdelo en el disco. Haga clic dos veces en el archivo para mostrar los resultados en el explorador por defecto.





## Práctica 5: Soluciones (continuación)

3. Su jefe desea ejecutar el informe de empleados con frecuencia. Cree un procedimiento que utilice el paquete DBMS\_SCHEDULER para planificar el procedimiento EMPLOYEE\_REPORT para su ejecución. Debe utilizar parámetros para especificar una frecuencia y un argumento opcional para especificar los minutos tras los que un trabajo planificado debe terminar.
  - a. Cree un procedimiento denominado SCHEDULE\_REPORT que proporcione los dos parámetros siguientes:
    - interval para especificar una cadena que indique la frecuencia del trabajo planificado.
    - minutes para especificar el tiempo total en minutos (el valor defecto es 10) para el trabajo planificado, tras los que estará terminado. El código dividirá la duración por la cantidad ( $24 \times 60$ ) cuando se agrega a la fecha y hora actuales para especificar la hora de terminación.

Cuando el procedimiento crea un trabajo con el nombre de EMPSAL\_REPORT llamándolo DBMS\_SCHEDULER.CREATE\_JOB, el trabajo debe estar activado y planificado para que el bloque PL/SQL se inicie inmediatamente. Debe planificar un bloque anónimo para llamar al procedimiento EMPLOYEE\_REPORT para que el nombre del archivo se pueda actualizar con una nueva hora cada vez que se ejecute el informe. A EMPLOYEE\_REPORT se le da el nombre de directorio proporcionado por el instructor para la tarea 1 y el parámetro del nombre de archivo se especifica con el siguiente formato:

sal\_rptxx\_hh24-mi-ss.txt, en el que xx es el número de usuario asignado y hh24-mi-ss representa las horas, minutos y segundos.

Utilice la siguiente variable PL/SQL local para construir un bloque PL/SQL:

```
plsql_block VARCHAR2(200) :=
  'BEGIN' ||
  '  EMPLOYEE_REPORT(''UTL_FILE'', ''||
  '    ''sal_rptxx_'' || to_char(sysdate, ''HH24-MI-SS'') || '''.txt''); ' ||
  'END;';
```

Este código se proporciona para ayudarle ya que se trata de una cadena PL/SQL no trivial para construir. En el bloque PL/SQL, **xx** es el número de estudiante.

```
CREATE OR REPLACE PROCEDURE schedule_report(
  interval VARCHAR2, minutes NUMBER := 10) IS
  plsql_block VARCHAR2(200) :=
    'BEGIN' ||
    '  EMPLOYEE_REPORT(''UTL_FILE'', ''||
    '    ''sal_rpt01_'' || to_char(sysdate, ''HH24-MI-SS'') || '''.txt''); ' ||
    'END;';
BEGIN
```

## Práctica 5: Soluciones (continuación)

```
DBMS_SCHEDULER.CREATE_JOB(  
  job_name => 'EMPSAL_REPORT',  
  job_type => 'PLSQL_BLOCK',  
  job_action => plsql_block,  
  start_date => SYSDATE,  
  repeat_interval => interval,  
  end_date => SYSDATE + minutes/(24*60),  
  enabled => TRUE);  
END;  
/  
SHOW ERRORS  
  
Procedure created.  
  
No errors.
```

- b. Pruebe el procedimiento `SCHEDULE_REPORT` ejecutándolo con un parámetro que especifique una frecuencia de 2 minutos y una hora de terminación de 10 minutos desde el inicio.

**Nota:** Deberá conectarse al servidor de base de datos utilizando PSFTP para comprobar si los archivos se han creado.

```
EXECUTE schedule_report('FREQUENCY=MINUTELY;INTERVAL=2', 10)  
  
PL/SQL procedure successfully completed.
```

- c. Durante el proceso y después del mismo, puede consultar `job_name` y las columnas activadas de la tabla `USER_SCHEDULER_JOBS` para comprobar si el trabajo aún existe.

```
SELECT job_name, enabled  
FROM user_scheduler_jobs;
```

**Nota:** Esta consulta no debería devolver ninguna fila cuando hayan pasado 10 minutos.

## Práctica 6: Soluciones

1. Cree un paquete denominado TABLE\_PKG que utilice SQL dinámico nativo para crear o borrar una tabla y para rellenar, modificar y suprimir filas de la tabla.

- a. Cree una especificación del paquete siguiendo estos procedimientos:

```
PROCEDURE make(table_name VARCHAR2, col_specs VARCHAR2)
PROCEDURE add_row(table_name VARCHAR2, col_values VARCHAR2,
  cols VARCHAR2 := NULL)
PROCEDURE upd_row(table_name VARCHAR2, set_values VARCHAR2,
  conditions VARCHAR2 := NULL)
PROCEDURE del_row(table_name VARCHAR2,
  conditions VARCHAR2 := NULL)
PROCEDURE remove(table_name VARCHAR2)
```

Asegúrese de que los subprogramas gestionan parámetros por defecto opcionales con valores NULL.

```
CREATE OR REPLACE PACKAGE table_pkg IS
  PROCEDURE make(table_name VARCHAR2, col_specs VARCHAR2);
  PROCEDURE add_row(table_name VARCHAR2, col_values VARCHAR2,
    cols VARCHAR2 := NULL);
  PROCEDURE upd_row(table_name VARCHAR2, set_values VARCHAR2,
    conditions VARCHAR2 := NULL);
  PROCEDURE del_row(table_name VARCHAR2, conditions VARCHAR2 := NULL);
  PROCEDURE remove(table_name VARCHAR2);
END table_pkg;
/
SHOW ERRORS

Package created.

No errors.
```

- b. Cree el cuerpo del paquete que acepte los parámetros y construya dinámicamente las sentencias SQL adecuadas que se ejecutan utilizando SQL dinámico nativo, excepto por el procedimiento remove que se debería escribir utilizando el paquete DBMS\_SQL.

```
CREATE OR REPLACE PACKAGE BODY table_pkg IS
  PROCEDURE execute(stmt VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(stmt);
    EXECUTE IMMEDIATE stmt;
  END;

  PROCEDURE make(table_name VARCHAR2, col_specs VARCHAR2) IS
    stmt VARCHAR2(200) := 'CREATE TABLE ' || table_name ||
      ' (' || col_specs || ')';
  BEGIN
    execute(stmt);
  END;
```

## Práctica 6: Soluciones (continuación)

```
PROCEDURE add_row(table_name VARCHAR2, col_values VARCHAR2,
  cols VARCHAR2 := NULL) IS
  stmt VARCHAR2(200) := 'INSERT INTO ' || table_name;
BEGIN
  IF cols IS NOT NULL THEN
    stmt := stmt || ' (' || cols || ')';
  END IF;
  stmt := stmt || ' VALUES (' || col_values || ')';
  execute(stmt);
END;

PROCEDURE upd_row(table_name VARCHAR2, set_values VARCHAR2,
  conditions VARCHAR2 := NULL) IS
  stmt VARCHAR2(200) := 'UPDATE ' || table_name || ' SET ' ||
set_values;
BEGIN
  IF conditions IS NOT NULL THEN
    stmt := stmt || ' WHERE ' || conditions;
  END IF;
  execute(stmt);
END;

PROCEDURE del_row(table_name VARCHAR2, conditions VARCHAR2 := NULL) IS
  stmt VARCHAR2(200) := 'DELETE FROM ' || table_name;
BEGIN
  IF conditions IS NOT NULL THEN
    stmt := stmt || ' WHERE ' || conditions;
  END IF;
  execute(stmt);
END;

PROCEDURE remove(table_name VARCHAR2) IS
  csr_id INTEGER;
  stmt VARCHAR2(100) := 'DROP TABLE ' || table_name;
BEGIN
  csr_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_OUTPUT.PUT_LINE(stmt);
  DBMS_SQL.PARSE(csr_id, stmt, DBMS_SQL.NATIVE);
  -- Parse executes DDL statements, no EXECUTE is required.
  DBMS_SQL.CLOSE_CURSOR(csr_id);
END;

END table_pkg;
/
SHOW ERRORS

Package body created.

No errors.
```

## Práctica 6: Soluciones (continuación)

- c. Ejecute el procedimiento MAKE del paquete para crear la siguiente tabla:

```
make('my_contacts', 'id number(4), name varchar2(40)');  
  
EXECUTE table_pkg.make('my_contacts', 'id number(4), name varchar2(40)')  
  
PL/SQL procedure successfully completed.
```

- d. Describa la estructura de la tabla MY\_CONTACTS.

```
DESCRIBE my_contacts
```

Name	Null?	Type
ID		NUMBER(4)
NAME		VARCHAR2(40)

- e. Ejecute el procedimiento empaquetado ADD\_ROW para agregar las siguientes filas:

```
add_row('my_contacts', '1', 'Geoff Gallus', 'id, name');  
add_row('my_contacts', '2', 'Nancy', 'id, name');  
add_row('my_contacts', '3', 'Sunitha Patel', 'id, name');  
add_row('my_contacts', '4', 'Valli Pataballa', 'id, name');  
  
BEGIN  
  table_pkg.add_row('my_contacts', '1', 'Geoff Gallus', 'id, name');  
  table_pkg.add_row('my_contacts', '2', 'Nancy', 'id, name');  
  table_pkg.add_row('my_contacts', '3', 'Sunitha Patel', 'id, name');  
  table_pkg.add_row('my_contacts', '4', 'Valli Pataballa', 'id, name');  
END;  
/  
  
PL/SQL procedure successfully completed.
```

- f. Consulte el contenido de la tabla MY\_CONTACTS.

```
SELECT *  
FROM my_contacts;
```

ID	NAME
1	Geoff Gallus
2	Nancy
3	Sunitha Patel
4	Valli Pataballa

- g. Ejecute el procedimiento empaquetado DEL\_ROW para suprimir un contacto con ID valor 1

```
EXECUTE table_pkg.del_row('my_contacts', 'id=1')  
  
PL/SQL procedure successfully completed.
```

## Práctica 6: Soluciones (continuación)

- h. Ejecute el procedimiento UPD\_ROW con los siguientes datos de fila:

```
upd_row('my_contacts', 'name='Nancy Greenberg'', 'id=2');
```

```
EXEC table_pkg.upd_row('my_contacts', 'name='Nancy Greenberg'', 'id=2')  
PL/SQL procedure successfully completed.
```

- i. Seleccione los datos de la tabla MY\_CONTACTS de nuevo para ver los cambios.

```
SELECT *  
FROM my_contacts;
```

ID	NAME
2	Nancy Greenberg
3	Sunitha Patel
4	Valli Pataballa

- j. Borre la tabla utilizando el procedimiento remove y describa la tabla MY\_CONTACTS.

```
EXECUTE table_pkg.remove('my_contacts')  
DESCRIBE my_contacts  
  
PL/SQL procedure successfully completed.  
  
ERROR:  
ORA-04043: object my_contacts does not exist
```

2. Cree un paquete COMPILE\_PKG que compile el código PL/SQL en el esquema.

- a. En la especificación, cree un procedimiento empaquetado denominado MAKE que acepte el nombre de una unidad de programa PL/SQL para compilarla.

```
CREATE OR REPLACE PACKAGE compile_pkg IS  
    PROCEDURE make(name VARCHAR2);  
END compile_pkg;  
/  
SHOW ERRORS  
  
Package created.  
  
No errors.
```

## Práctica 6: Soluciones (continuación)

- b. En el cuerpo, el procedimiento MAKE debería llamar a una función privada denominada GET\_TYPE para determinar el tipo de objeto PL/SQL del diccionario de datos y devolver el nombre del tipo (utilice PACKAGE para un paquete con cuerpo) si existe el objeto; de lo contrario, devolverá NULL. Si el objeto existe, MAKE lo compila dinámicamente con la sentencia ALTER.

```
CREATE OR REPLACE PACKAGE BODY compile_pkg IS
  PROCEDURE execute(stmt VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(stmt);
    EXECUTE IMMEDIATE stmt;
  END;

  FUNCTION get_type(name VARCHAR2) RETURN VARCHAR2 IS
    proc_type VARCHAR2(30) := NULL;
  BEGIN
    /*
     * The ROWNUM = 1 is added to the condition
     * to ensure only one row is returned if the
     * name represents a PACKAGE, which may also
     * have a PACKAGE BODY. In this case, we can
     * only compile the complete package, but not
     * the specification or body as separate
     * components.
     */
    SELECT object_type INTO proc_type
    FROM user_objects
    WHERE object_name = UPPER(name)
    AND ROWNUM = 1;
    RETURN proc_type;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN NULL;
  END;

  PROCEDURE make(name VARCHAR2) IS
    stmt          VARCHAR2(100);
    proc_type     VARCHAR2(30) := get_type(name);
  BEGIN
    IF proc_type IS NOT NULL THEN
      stmt := 'ALTER ' || proc_type || ' ' || name || ' COMPILE';
      execute(stmt);
    ELSE
      RAISE_APPLICATION_ERROR(-20001,
        'Subprogram ''' || name || ''' does not exist');
    END IF;
  END make;
END compile_pkg;
/
SHOW ERRORS
```

## Práctica 6: Soluciones (continuación)

Package body created.

No errors.

- c. Utilice el procedimiento `COMPILE_PKG.MAKE` para compilar el procedimiento `EMPLOYEE_REPORT` el paquete `EMP_PKG` y un objeto no existente denominado `EMP_DATA`.

```
EXECUTE compile_pkg.make('employee_report')
EXECUTE compile_pkg.make('emp_pkg')
EXECUTE compile_pkg.make('emp_data')

ALTER PROCEDURE employee_report COMPILE
PL/SQL procedure successfully completed.

ALTER PACKAGE emp_pkg COMPILE
PL/SQL procedure successfully completed

BEGIN compile_pkg.make('emp_data'); END;

*

ERROR at line 1:
ORA-20001: Subprogram 'emp_data' does not exist
ORA-06512: at "ORA1.COMPILE_PKG", line 39
ORA-06512: at line 1
```

3. Agregue un procedimiento a `COMPILE_PKG` que utilice `DBMS_METADATA` para obtener una sentencia DDL que pueda regenerar un subprograma PL/SQL con nombre y escriba la sentencia DDL en un archivo utilizando el paquete `UTL_FILE`.
  - a. En la especificación del paquete, cree un procedimiento denominado `REGENERATE` que acepte el nombre de un componente PL/SQL para regenerarse. Declare una variable pública `VARCHAR2` denominada `dir` inicializada con valor de alias de directorio `'UTL_FILE'`. Compile la especificación.

```
CREATE OR REPLACE PACKAGE compile_pkg IS
  dir VARCHAR2(100) := 'UTL_FILE';
  PROCEDURE make(name VARCHAR2);
  PROCEDURE regenerate(name VARCHAR2);
END compile_pkg;
/
SHOW ERRORS

Package created.

No errors.
```

**Nota:** Inicialice el nombre correcto de la ruta de acceso en el valor de la variable `dir` para su curso.



## Práctica 6: Soluciones (continuación)

- b. En el cuerpo del paquete, implemente el procedimiento `REGENERATE` para que utilice la función `GET_TYPE` para determinar el tipo de objeto PL/SQL del nombre proporcionado. Si el objeto existe, obtenga el DDL utilizado para crear el componente con el procedimiento `DBMS_METADATA.GET_DDL`, que se debe proporcionar con el nombre del objeto en mayúscula. Guarde la sentencia DDL en un archivo utilizando el procedimiento `UTL_FILE.PUT`. Escriba el archivo en la ruta de acceso del directorio almacenada en la variable pública denominada `dir` (de la especificación). Construya un nombre de archivo (en minúsculas) concatenando la función `USER`, un subrayado y el nombre de objeto con una extensión `.sql`. Por ejemplo: `ora1_myobject.sql`. Compile el cuerpo.

```
CREATE OR REPLACE PACKAGE BODY compile_pkg IS

  PROCEDURE execute(stmt VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(stmt);
    EXECUTE IMMEDIATE stmt;
  END;

  FUNCTION get_type(name VARCHAR2) RETURN VARCHAR2 IS
    proc_type VARCHAR2(30) := NULL;
  BEGIN
    /*
     * The ROWNUM = 1 is added to the condition
     * to ensure only one row is returned if the
     * name represents a PACKAGE, which may also
     * have a PACKAGE BODY. In this case, we can
     * only compile the complete package, but not
     * the specification or body as separate
     * components.
     */
    SELECT object_type INTO proc_type
    FROM user_objects
    WHERE object_name = UPPER(name)
    AND ROWNUM = 1;
    RETURN proc_type;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN NULL;
  END;
```

## Práctica 6: Soluciones (continuación)

```
PROCEDURE make(name VARCHAR2) IS
    stmt          VARCHAR2(100);
    proc_type     VARCHAR2(30) := get_type(name);
BEGIN
    IF proc_type IS NOT NULL THEN
        stmt := 'ALTER ' || proc_type || ' ' || name || ' COMPILE';
        execute(stmt);
    ELSE
        RAISE_APPLICATION_ERROR(-20001,
            'Subprogram ''' || name || ''' does not exist');
    END IF;
END make;

PROCEDURE regenerate (name VARCHAR2) IS
    file UTL_FILE.FILE_TYPE;
    filename VARCHAR2(100) := LOWER(USER || '_' || name || '.sql');
    proc_type VARCHAR2(30) := get_type(name);
BEGIN
    IF proc_type IS NOT NULL THEN
        file := UTL_FILE.FOPEN(dir, filename, 'w');
        UTL_FILE.PUT(file,
            DBMS_METADATA.GET_DDL(proc_type, UPPER(name)));
        UTL_FILE.FCLOSE(file);
    ELSE
        RAISE_APPLICATION_ERROR(-20001,
            'Object with ''' || name || ''' does not exist');
    END IF;
END regenerate;

END compile_pkg;
/
SHOW ERRORS

Package body created.

No errors.
```

- c. Ejecute el procedimiento `COMPILE_PKG.REGENERATE` utilizando el nombre de `TABLE_PKG` creado en la primera tarea de esta práctica.

```
EXECUTE compile_pkg.regenerate('TABLE_PKG')
```

**Nota:** Si fuera necesario, puede ejecutar la siguiente sentencia para definir el directorio para el archivo:

```
EXECUTE compile_pkg.dir := '<utl_file_dir>';
```

## Práctica 6: Soluciones (continuación)

- d. Utilice Putty FTP para obtener el archivo generado del servidor en el directorio local. Edite el archivo para colocar un carácter de terminación / al final de una sentencia CREATE (si es necesario). Corte y pegue los resultados en el buffer de iSQL\*Plus y ejecute la sentencia.

A continuación se presenta un ejemplo de sesión Putty FTP:

```
psftp> open esslin05
login as: teach7
Using username "teach7".
Password: *****
Remote working directory is /home1/teach7
psftp> cd UTL_FILE
Remote directory is now /home1/teach7/UTL_FILE
psftp> lcd E:\labs\PLPU\labs
New local directory is E:\labs\PLPU\labs
psftp> get oral_emp_pkg.sql
remote:/home1/teach7/UTL_FILE/oral_emp_pkg.sql => local:oral_emp_pkg.sql
psftp> exit
```

Oracle Internal & OAI Use Only

## Práctica 7: Soluciones

1. Actualice EMP\_PKG con un nuevo procedimiento para consultar los empleados de un departamento especificado.
  - a. En la especificación, declare un procedimiento `get_employees`, con el parámetro denominado `dept_id` basado en el tipo de columna `employees.department_id`. Defina un tipo PL/SQL index-by como `TABLE OF EMPLOYEES%ROWTYPE`.

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  TYPE emp_tabtype IS TABLE OF employees%ROWTYPE;
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30);
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    deptid     employees.department_id%TYPE);
  PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE);
  FUNCTION get_employee(emp_id employees.employee_id%type)
    return employees%rowtype;
  FUNCTION get_employee(family_name employees.last_name%type)
    return employees%rowtype;
  PROCEDURE get_employees(dept_id employees.department_id%type);
  PROCEDURE init_departments;
  PROCEDURE print_employee(emprec employees%rowtype);
END emp_pkg;
/
SHOW ERRORS

Package created.

No errors.
```

- b. En el cuerpo del paquete, defina una variable privada denominada `emp_table` basada en el tipo definido en la especificación para almacenar los registros de los empleados. Implemente el procedimiento `get_employees` para recuperar en bloque los datos en la tabla.

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  TYPE boolean_tabtype IS TABLE OF BOOLEAN
    INDEX BY BINARY_INTEGER;
  valid_departments boolean_tabtype;
  emp_table          emp_tabtype;
```

## Práctica 7: Soluciones (continuación)

```
FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
RETURN BOOLEAN;

PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(deptid) THEN
        INSERT INTO employees(employee_id, first_name, last_name, email,
            job_id, manager_id, hire_date, salary, commission_pct, department_id)
        VALUES (employees_seq.NEXTVAL, first_name, last_name, email,
            job, mgr, TRUNC(SYSDATE), sal, comm, deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204,
            'Invalid department ID. Try again.');
```

END IF;

```
END add_employee;

PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    deptid     employees.department_id%TYPE) IS
    email      employees.email%TYPE;
BEGIN
    email := UPPER(SUBSTR(first_name, 1, 1) || SUBSTR(last_name, 1, 7));
    add_employee(first_name, last_name, email, deptid => deptid);
END;

PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO   sal, job
    FROM   employees
    WHERE  employee_id = empid;
END get_employee;
```

## Práctica 7: Soluciones (continuación)

```
FUNCTION get_employee(emp_id employees.employee_id%type)
  return employees%rowtype IS
  emprec employees%rowtype;
BEGIN
  SELECT * INTO emprec
  FROM employees
  WHERE employee_id = emp_id;
  RETURN emprec;
END;

FUNCTION get_employee(family_name employees.last_name%type)
  return employees%rowtype IS
  emprec employees%rowtype;
BEGIN
  SELECT * INTO emprec
  FROM employees
  WHERE last_name = family_name;
  RETURN emprec;
END;

PROCEDURE get_employees(dept_id employees.department_id%type) IS
BEGIN
  SELECT * BULK COLLECT INTO emp_table
  FROM EMPLOYEES
  WHERE department_id = dept_id;
END;

PROCEDURE init_departments IS
BEGIN
  FOR rec IN (SELECT department_id FROM departments)
  LOOP
    valid_departments(rec.department_id) := TRUE;
  END LOOP;
END;

PROCEDURE print_employee(emprec employees%rowtype) IS
BEGIN
  DBMS_OUTPUT.PUT_LINE(emprec.department_id || ' ' ||
    emprec.employee_id || ' ' ||
    emprec.first_name || ' ' ||
    emprec.last_name || ' ' ||
    emprec.job_id || ' ' ||
    emprec.salary);
END;
```

## Práctica 7: Soluciones (continuación)

```
FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
RETURN BOOLEAN IS
    dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

BEGIN
    init_departments;
END emp_pkg;
/
SHOW ERRORS

Package body created.

No errors.
```

- c. Cree un nuevo procedimiento en la especificación y el cuerpo, denominado `show_employees`, que no toma argumentos y muestra el contenido de la variable de tabla PL/SQL privada (si existen datos).

**Indicación:** Utilice el procedimiento `print_employee`.

```
CREATE OR REPLACE PACKAGE emp_pkg IS
TYPE emp_tabtype IS TABLE OF employees%ROWTYPE;
PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name   employees.last_name%TYPE,
    email       employees.email%TYPE,
    job         employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr         employees.manager_id%TYPE  DEFAULT 145,
    sal         employees.salary%TYPE      DEFAULT 1000,
    comm        employees.commission_pct%TYPE DEFAULT 0,
    deptid      employees.department_id%TYPE DEFAULT 30);
PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name   employees.last_name%TYPE,
    deptid      employees.department_id%TYPE);
PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE);
FUNCTION get_employee(emp_id employees.employee_id%type)
    return employees%rowtype;
FUNCTION get_employee(family_name employees.last_name%type)
    return employees%rowtype;
PROCEDURE get_employees(dept_id employees.department_id%type);
PROCEDURE init_departments;
```

## Práctica 7: Soluciones (continuación)

```
PROCEDURE print_employee(emprec employees%rowtype);
PROCEDURE show_employees;
END emp_pkg;
/
SHOW ERRORS

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  TYPE boolean_tabtype IS TABLE OF BOOLEAN
    INDEX BY BINARY_INTEGER;
  valid_departments boolean_tabtype;
  emp_table          emp_tabtype;

  FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
    RETURN BOOLEAN;

  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30) IS
  BEGIN
    IF valid_deptid(deptid) THEN
      INSERT INTO employees(employee_id, first_name, last_name, email,
        job_id, manager_id, hire_date, salary, commission_pct, department_id)
      VALUES (employees_seq.NEXTVAL, first_name, last_name, email,
        job, mgr, TRUNC(SYSDATE), sal, comm, deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204,
        'Invalid department ID. Try again.');
```

END IF;

END add\_employee;

```
PROCEDURE add_employee(
  first_name employees.first_name%TYPE,
  last_name  employees.last_name%TYPE,
  deptid     employees.department_id%TYPE) IS
  email      employees.email%type;
BEGIN
  email := UPPER(SUBSTR(first_name, 1, 1)||SUBSTR(last_name, 1, 7));
  add_employee(first_name, last_name, email, deptid => deptid);
END;
```



## Práctica 7: Soluciones (continuación)

```
PROCEDURE get_employee(  
    empid IN employees.employee_id%TYPE,  
    sal    OUT employees.salary%TYPE,  
    job    OUT employees.job_id%TYPE) IS  
BEGIN  
    SELECT  salary, job_id  
    INTO    sal, job  
    FROM    employees  
    WHERE   employee_id = empid;  
END get_employee;  
  
FUNCTION get_employee(emp_id employees.employee_id%type)  
    return employees%rowtype IS  
    emprec employees%rowtype;  
BEGIN  
    SELECT * INTO emprec  
    FROM employees  
    WHERE employee_id = emp_id;  
    RETURN emprec;  
END;  
  
FUNCTION get_employee(family_name employees.last_name%type)  
    return employees%rowtype IS  
    emprec employees%rowtype;  
BEGIN  
    SELECT * INTO emprec  
    FROM employees  
    WHERE last_name = family_name;  
    RETURN emprec;  
END;  
  
PROCEDURE get_employees(dept_id employees.department_id%type) IS  
BEGIN  
    SELECT * BULK COLLECT INTO emp_table  
    FROM EMPLOYEES  
    WHERE department_id = dept_id;  
END;  
  
PROCEDURE init_departments IS  
BEGIN  
    FOR rec IN (SELECT department_id FROM departments)  
    LOOP  
        valid_departments(rec.department_id) := TRUE;  
    END LOOP;  
END;
```

## Práctica 7: Soluciones (continuación)

```
PROCEDURE print_employee(emprec employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(emprec.department_id || ' ' ||
                          emprec.employee_id || ' ' ||
                          emprec.first_name || ' ' ||
                          emprec.last_name || ' ' ||
                          emprec.job_id || ' ' ||
                          emprec.salary);
END;

PROCEDURE show_employees IS
BEGIN
    IF emp_table IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('Employees in Package table');
        FOR i IN 1 .. emp_table.COUNT
        LOOP
            print_employee(emp_table(i));
        END LOOP;
    END IF;
END show_employees;

FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
RETURN BOOLEAN IS
    dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

BEGIN
    init_departments;
END emp_pkg;
/
SHOW ERRORS

Package created.

No errors.

Package body created.

No errors.
```

## Práctica 7: Soluciones (continuación)

- d. Llame al procedimiento `emp_pkg.get_employees` para el departamento 30 y, a continuación, llame a `emp_pkg.show_employees`. Repítalo para el departamento 60.

```
EXECUTE emp_pkg.get_employees(30)
EXECUTE emp_pkg.show_employees

PL/SQL procedure successfully completed.
```

```
Employees in Package table
30 114 Den Raphaely PU_MAN 11000
30 115 Alexander Khoo PU_CLERK 3100
30 116 Shelli Baida PU_CLERK 2900
30 117 Sigal Tobias PU_CLERK 2800
30 118 Guy Himuro PU_CLERK 2600
30 119 Karen Colmenares PU_CLERK 2500
30 209 Samuel Joplin SA_REP 1000
PL/SQL procedure successfully completed.
```

```
EXECUTE emp_pkg.get_employees(60)
EXECUTE emp_pkg.show_employees

PL/SQL procedure successfully completed.
```

```
Employees in Package table
60 103 Alexander Hunold IT_PROG 9000
60 104 Bruce Ernst IT_PROG 6000
60 105 David Austin IT_PROG 4800
60 106 Valli Pataballa IT_PROG 4800
60 107 Diana Lorentz IT_PROG 4200
PL/SQL procedure successfully completed.
```

2. El director desea mantener un log cada vez que se llama al procedimiento `add_employee` en el paquete para insertar un nuevo empleado en la tabla `EMPLOYEES`.

- a. En primer lugar, cargue y ejecute el archivo de comandos `E:\labs\PLPU\labs\lab_07_02_a.sql` para crear una tabla de log denominada `LOG_NEWEMP` y una secuencia denominada `log_newemp_seq`.

```
CREATE TABLE log_newemp (
  entry_id  NUMBER(6) CONSTRAINT log_newemp_pk PRIMARY KEY,
  user_id   VARCHAR2(30),
  log_time  DATE,
  name      VARCHAR2(60)
);

CREATE SEQUENCE log_newemp_seq;

Table created.

Sequence created.
```

## Práctica 7: Soluciones (continuación)

- b. En el cuerpo del paquete, modifique el procedimiento `add_employee` que realiza la operación real `INSERT` para tener un procedimiento local denominado `audit_newemp`. El procedimiento `audit_newemp` debe utilizar una transacción autónoma para insertar un registro log en la tabla `LOG_NEWEMP`. Almacene el `USER`, la hora actual y el nombre del nuevo empleado en la fila de la tabla de log. Utilice `log_newemp_seq` para definir la columna `entry_id`.

**Nota:** Recuerde realizar una operación `COMMIT` en un procedimiento con una transacción autónoma.

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tabtype IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;
    valid_departments boolean_tabtype;
    emp_table          emp_tabtype;

    FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
        RETURN BOOLEAN;

    PROCEDURE add_employee(
        first_name employees.first_name%TYPE,
        last_name  employees.last_name%TYPE,
        email      employees.email%TYPE,
        job        employees.job_id%TYPE          DEFAULT 'SA_REP',
        mgr        employees.manager_id%TYPE      DEFAULT 145,
        sal        employees.salary%TYPE          DEFAULT 1000,
        comm       employees.commission_pct%TYPE  DEFAULT 0,
        deptid     employees.department_id%TYPE   DEFAULT 30) IS

        PROCEDURE audit_newemp IS
            PRAGMA AUTONOMOUS_TRANSACTION;
            user_id VARCHAR2(30) := USER;
        BEGIN
            INSERT INTO log_newemp (entry_id, user_id, log_time, name)
            VALUES (log_newemp_seq.NEXTVAL, user_id, sysdate,
                    first_name||' '||last_name);
            COMMIT;
        END audit_newemp;

    BEGIN
        IF valid_deptid(deptid) THEN
            INSERT INTO employees(employee_id, first_name, last_name, email,
                                job_id, manager_id, hire_date, salary, commission_pct, department_id)
            VALUES (employees_seq.NEXTVAL, first_name, last_name, email,
                    job, mgr, TRUNC(SYSDATE), sal, comm, deptid);
        ELSE
            RAISE_APPLICATION_ERROR (-20204,
                                     'Invalid department ID. Try again.');
```

## Práctica 7: Soluciones (continuación)

```
PROCEDURE add_employee(  
    first_name employees.first_name%TYPE,  
    last_name  employees.last_name%TYPE,  
    deptid employees.department_id%TYPE) IS  
    email employees.email%type;  
BEGIN  
    email := UPPER(SUBSTR(first_name, 1, 1)||SUBSTR(last_name, 1, 7));  
    add_employee(first_name, last_name, email, deptid => deptid);  
END;  
  
...  
  
FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)  
    RETURN BOOLEAN IS  
    dummy PLS_INTEGER;  
BEGIN  
    RETURN valid_departments.exists(deptid);  
EXCEPTION  
    WHEN NO_DATA_FOUND THEN  
        RETURN FALSE;  
END valid_deptid;  
  
BEGIN  
    init_departments;  
END emp_pkg;  
/  
SHOW ERRORS  
  
Package body created.  
  
No errors.
```

- c. Modifique el procedimiento `add_employee` para llamar a `audit_emp` antes de que realice la operación de inserción.

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS  
    TYPE boolean_tabtype IS TABLE OF BOOLEAN  
        INDEX BY BINARY_INTEGER;  
    valid_departments boolean_tabtype;  
    emp_table          emp_tabtype;  
  
    FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)  
        RETURN BOOLEAN;
```

## Práctica 7: Soluciones (continuación)

```
PROCEDURE add_employee(  
    first_name employees.first_name%TYPE,  
    last_name  employees.last_name%TYPE,  
    email      employees.email%TYPE,  
    job        employees.job_id%TYPE          DEFAULT 'SA_REP',  
    mgr        employees.manager_id%TYPE      DEFAULT 145,  
    sal        employees.salary%TYPE          DEFAULT 1000,  
    comm       employees.commission_pct%TYPE  DEFAULT 0,  
    deptid     employees.department_id%TYPE   DEFAULT 30) IS  
  
    PROCEDURE audit_newemp IS  
        PRAGMA AUTONOMOUS_TRANSACTION;  
        user_id VARCHAR2(30) := USER;  
    BEGIN  
        INSERT INTO log_newemp (entry_id, user_id, log_time, name)  
        VALUES (log_newemp_seq.NEXTVAL, user_id, sysdate,  
                first_name||' '||last_name);  
        COMMIT;  
    END audit_newemp;  
    BEGIN  
        IF valid_deptid(deptid) THEN  
            audit_newemp;  
            INSERT INTO employees(employee_id, first_name, last_name, email,  
                                job_id, manager_id, hire_date, salary, commission_pct, department_id)  
            VALUES (employees_seq.NEXTVAL, first_name, last_name, email,  
                    job, mgr, TRUNC(SYSDATE), sal, comm, deptid);  
        ELSE  
            RAISE_APPLICATION_ERROR (-20204,  
                                    'Invalid department ID. Try again.');        END IF;  
    END add_employee;  
    ...  
    FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)  
    RETURN BOOLEAN IS  
        dummy PLS_INTEGER;  
    BEGIN  
        RETURN valid_departments.exists(deptid);  
    EXCEPTION  
        WHEN NO_DATA_FOUND THEN  
            RETURN FALSE;  
    END valid_deptid;  
  
    BEGIN  
        init_departments;  
    END emp_pkg;  
    /  
    SHOW ERRORS  
  
    Package body created.  
  
    No errors.
```

## Práctica 7: Soluciones (continuación)

- d. Llame al procedimiento `add_employee` para los siguientes nuevos empleados: Max Smart del departamento 20 y Clark Kent del departamento 10. ¿Qué sucede?

```
EXECUTE emp_pkg.add_employee('Max', 'Smart', 20)
EXECUTE emp_pkg.add_employee('Clark', 'Kent', 10)

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.
```

**Ambas operaciones de inserción se han terminado correctamente y la tabla de log tiene dos registros como se muestra en el paso siguiente.**

- e. Consulte los dos registros `EMPLOYEES` que se han agregado y los registros de la tabla `LOG_NEWEMP`.  
¿Cuántos registros log están presentes?

```
SELECT department_id, first_name, last_name
FROM employees
WHERE last_name IN ('Smart', 'Kent');
```

DEPARTMENT_ID	EMPLOYEE_ID	LAST_NAME	FIRST_NAME
10	222	Kent	Clark
20	221	Smart	Max

```
SELECT *
FROM log_newemp;
```

ENTRY_ID	USER_ID	LOG_TIME	NAME
1	ORA1	18-FEB-04	Max Smart
2	ORA1	18-FEB-04	Clark Kent

**Hay dos registros log, uno para ~~Smart~~ y el otro para ~~Kent~~.**

- f. Ejecute una sentencia `ROLLBACK` para deshacer las operaciones de inserción que no se han confirmado. Utilice las mismas consultas del Ejercicio 2e: la primera para comprobar si las filas para los empleados Smart y Kent se han eliminado y la segunda para comprobar los registros log de la tabla `LOG_NEWEMP`. ¿Cuántos registros log están presentes? ¿Por qué?

```
ROLLBACK;

Rollback complete.
```

## Práctica 7: Soluciones (continuación)

```
SELECT department_id, first_name, last_name
FROM employees
WHERE last_name IN ('Smart','Kent');
```

no rows selected

```
SELECT *
FROM log_newemp;
```

ENTRY_ID	USER_ID	LOG_TIME	NAME
1	ORA1	18-FEB-04	Max Smart
2	ORA1	18-FEB-04	Clark Kent

**Los registros de los dos empleados se han eliminado (se ha realizado un rollback). Ambos registros log permanecen en la tabla de log porque se insertaron utilizando una transacción autónoma que no se ve afectada por el rollback que se ha llevado a cabo en la transacción principal.**

**Si tiene tiempo, realice el siguiente ejercicio:**

3. Modifique el paquete EMP\_PKG para utilizar AUTHID de CURRENT\_USER y pruebe el comportamiento con otro estudiante.

**Nota:** Verifique si existe en esta práctica la tabla LOG\_NEWEMP del Ejercicio 2.

- a. Primero, otorgue el privilegio EXECUTE a otro estudiante en el paquete EMP\_PKG.

**Suponga que es ORA1 y el otro estudiante es ORA2. Introduzca:**

```
GRANT EXECUTE ON EMP_PKG TO ORA2;
```

Grant succeeded.

- b. Pídale a otro estudiante que llame al procedimiento add\_employee para insertar al empleado Jaco Pastorius en el departamento 10. Recuerde anteponer el nombre del propietario del paquete al nombre del paquete. La llamada funcionará con los derechos del responsable de la definición.

**El usuario ORA2 introduce:**

```
EXECUTE ora1.emp_pkg.add_employee('Jaco', 'Pastorius', 10)
```

PL/SQL procedure successfully completed.



## Práctica 7: Soluciones (continuación)

- c. Ahora, ejecute una consulta de los empleados del departamento 10. ¿En la tabla de empleados de qué usuario se ha insertado el nuevo registro?

**El usuario ORA1 ejecuta:**

```
SELECT department_id, first_name, last_name
FROM employees
WHERE department_id = 10;
```

DEPARTMENT_ID	FIRST_NAME	LAST_NAME
10	Jennifer	Whalen
10	Jaco	Pastorius

**El usuario ORA2 ejecuta:**

```
SELECT department_id, first_name, last_name
FROM departments
WHERE department_id = 10;
```

DEPARTMENT_ID	FIRST_NAME	LAST_NAME
10	Jennifer	Whalen

**El nuevo empleado se ha agregado a la tabla en el esquema ORA1, es decir, en la tabla del propietario del paquete EMP\_PKG.**

- d. Ahora, modifique la especificación del paquete EMP\_PKG para utilizar AUTHID CURRENT\_USER. Compile el cuerpo de EMP\_PKG.

```
CREATE OR REPLACE PACKAGE emp_pkg AUTHID CURRENT_USER IS
  TYPE emp_tabtype IS TABLE OF employees%ROWTYPE;
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30);
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    deptid     employees.department_id%TYPE);
  PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE);
  FUNCTION get_employee(emp_id employees.employee_id%type)
    return employees%rowtype;
```

## Práctica 7: Soluciones (continuación)

```
FUNCTION get_employee(family_name employees.last_name%type)
    return employees%rowtype;
PROCEDURE get_employees(dept_id employees.department_id%type);
PROCEDURE init_departments;
PROCEDURE print_employee(emprec employees%rowtype);
PROCEDURE show_employees;
END emp_pkg;
/
SHOW ERRORS

ALTER PACKAGE emp_pkg COMPILE BODY;

Package created.

No errors.

Package body altered.
```

- e. Pídale al mismo estudiante que vuelva a ejecutar el procedimiento `add_employee`, para agregar al empleado Joe Zawinal en el departamento 10.

**El usuario ORA2 ejecuta:**

```
EXECUTE ora1.emp_pkg.add_employee('Joe', 'Zawinal', 10)
```

PL/SQL procedure successfully completed.

- f. Consulte a los empleados del departamento 10. ¿En qué tabla se agregó el nuevo empleado?

**El usuario ORA1 ejecuta:**

```
SELECT department_id, first_name, last_name
FROM employees
WHERE department_id = 10;
```

DEPARTMENT_ID	FIRST_NAME	LAST_NAME
10	Jennifer	Whalen
10	Jaco	Pastorius

**El usuario ORA2 ejecuta:**

```
SELECT department_id, first_name, last_name
FROM employees
WHERE department_id = 10;
```

DEPARTMENT_ID	FIRST_NAME	LAST_NAME
10	Joe	Zawinal
10	Jennifer	Whalen

## Práctica 7: Soluciones (continuación)

**El nuevo empleado se ha agregado a la tabla de empleados del usuario ORA2. Es decir, el nuevo empleado se ha agregado a la tabla propiedad del emisor del procedimiento empaquetado (derechos del invocador).**

- g. Escriba una consulta para mostrar los registros que se han agregado en las tablas LOG\_NEWEMP. Pídale al otro estudiante que consulte su propia copia de la tabla.

**El usuario ORA1 ejecuta:**

```
SELECT *  
FROM log_newemp;
```

ENTRY_ID	USER_ID	LOG_TIME	NAME
1	ORA1	18-FEB-04	Max Smart
2	ORA1	18-FEB-04	Clark Kent
3	ORA2	18-FEB-04	Jaco Pastorius

**El usuario ORA2 ejecuta:**

```
SELECT *  
FROM log_newemp;
```

ENTRY_ID	USER_ID	LOG_TIME	NAME
3	ORA2	18-FEB-04	Joe Zawinal
1	ORA2	18-FEB-04	Max Smart
2	ORA2	18-FEB-04	Clark Kent

**Los registros log que ha creado el procedimiento `audit_emp` (que ejecuta la transacción autónoma) se almacenan en la tabla de log del propietario del paquete cuando el procedimiento empaquetado se ejecuta con los derechos del responsable de la definición (propietario). Los registros log se almacenan en la tabla de log del emisor cuando se ejecuta el procedimiento empaquetado con los derechos del invocador (emisor).**

## Práctica 8: Soluciones

1. Responda a estas preguntas:

- a. ¿Se puede invalidar una tabla o un sinónimo?

**No se puede invalidar nunca una tabla o un sinónimo; sin embargo, los objetos dependientes, sí.**

- b. Considere el siguiente ejemplo de dependencia:

El procedimiento autónomo MY\_PROC depende del procedimiento del paquete MY\_PROC\_PACK. Se cambia la definición del procedimiento MY\_PROC\_PACK mediante la recompilación del cuerpo del paquete. La declaración del procedimiento MY\_PROC\_PACK no se modifica en la especificación del paquete.

Dada esta situación, ¿se ha invalidado el procedimiento autónomo MY\_PROC?

**No, no se ha invalidado porque el procedimiento autónomo MY\_PROC depende del procedimiento empaquetado MY\_PROC\_PACK, que no se ha modificado. Aunque el cuerpo del paquete se ha vuelto a compilar, la especificación del paquete no se ha invalidado y no es necesario que se vuelva a compilar.**

2. Cree una estructura de árbol que muestre todas las dependencias relacionadas con el procedimiento add\_employee y con la función valid\_deptid.

**Nota:** add\_employee y valid\_deptid se crearon en la lección 2 (“Creación de Funciones Almacenadas”). Puede ejecutar los archivos de comandos de la solución de la práctica 2 si tiene que crear el procedimiento y la función.

- a. Cargue y ejecute el archivo de comandos utldtree.sql ubicado en la carpeta E:\lab\PLPU\labs.

Cuando ejecute el archivo de comandos, se mostrarán los siguientes resultados (puede ignorar los mensajes de error):

```
drop sequence deptree_seq
*

ERROR at line 1:
ORA-02289: sequence does not exist
Sequence created.

drop table deptree temptab
*

ERROR at line 1:
ORA-00942: table or view does not exist
Table created.

Procedure created.
```

## Práctica 8: Soluciones (continuación)

```
drop view deptree
*
```

```
ERROR at line 1:
ORA-00942: table or view does not exist
```

```
REM This view will succeed if current user is sys. This view shows
REM which shared cursors depend on the given object. If the current
REM user is not sys, then this view get an error either about lack
REM of privileges or about the non-existence of table x$kgls.
```

```
set echo off
```

```
from deptree_temptab d, dba_objects o
*
```

```
ERROR at line 5:
ORA-00942: table or view does not exist
```

```
REM This view will succeed if current user is not sys. This view
REM does *not* show which shared cursors depend on the given object.
REM If the current user is sys then this view will get an error
REM indicating that the view already exists (since prior view create
REM will have succeeded).
```

```
set echo off
View created.
```

```
drop view ideptree
*
```

```
ERROR at line 1:
ORA-00942: table or view does not exist
View created.
```

b. Ejecute el procedimiento `deptree_fill` para el procedimiento `add_employee`.

```
EXECUTE deptree_fill('PROCEDURE', USER, 'add_employee')
```

```
PL/SQL procedure successfully completed.
```

c. Consulte la tabla `IDEPTREE` para ver los resultados.

```
SELECT * FROM IDEPTREE;
```

### DEPENDENCIES

```
PROCEDURE ORA1.ADD_EMPLOYEE
```

## Práctica 8: Soluciones (continuación)

- d. Ejecute el procedimiento `deptree_fill` para la función `valid_deptid`.

```
EXECUTE deptree_fill('FUNCTION', USER, 'valid_deptid')

PL/SQL procedure successfully completed.
```

- e. Consulte la tabla `IDEPTREE` para ver los resultados.

```
SELECT * FROM IDEPTREE;
```

DEPENDENCIES
FUNCTION ORA1.VALID_DEPTID
PROCEDURE ORA1.ADD_EMPLOYEE

### Si tiene tiempo, realice el siguiente ejercicio:

3. Valide de forma dinámica los objetos no válidos.

- a. Realice una copia de la tabla `EMPLOYEES` denominada `EMPS`.

```
CREATE TABLE emps AS
  SELECT * FROM employees;

Table created.
```

- b. Cambie la tabla `EMPLOYEES` y agregue la columna `TOTSAL` con tipo de dato `NUMBER(9,2)`.

```
ALTER TABLE employees
  ADD (totsal NUMBER(9,2));

Table altered.
```

- c. Cree y guarde una consulta para mostrar el nombre, el tipo y el estado de todos los objetos no válidos.

```
SELECT object_name, object_type, status
FROM USER_OBJECTS
WHERE status = 'INVALID';
```

## Práctica 8: Soluciones (continuación)

OBJECT_NAME	OBJECT_TYPE	STATUS
EMP_DETAILS_VIEW	VIEW	INVALID
SECURE_EMPLOYEES	TRIGGER	INVALID
UPDATE_JOB_HISTORY	TRIGGER	INVALID
TOTAL_SALARY	FUNCTION	INVALID
GET_EMPLOYEE	PROCEDURE	INVALID
GET_ANNUAL_COMP	FUNCTION	INVALID
ADD_EMPLOYEE	PROCEDURE	INVALID
EMP_PKG	PACKAGE	INVALID
EMP_PKG	PACKAGE BODY	INVALID
EMPLOYEE_REPORT	PROCEDURE	INVALID
WEB_EMPLOYEE_REPORT	PROCEDURE	INVALID

11 rows selected.

- d. En `compile_pkg` (creado en la práctica 6 de la lección titulada “SQL Dinámico y Metadatos”), agregue un procedimiento denominado `recompile` que recompile todos los procedimientos, las funciones y los paquetes no válidos del esquema. Utilice SQL dinámico nativo para realizar operaciones `ALTER` en el tipo de objeto no válido así como operaciones `COMPILE`.

```
CREATE OR REPLACE PACKAGE compile_pkg IS
    PROCEDURE make(name VARCHAR2);
    PROCEDURE recompile;
END compile_pkg;
/
SHOW ERRORS

CREATE OR REPLACE PACKAGE BODY compile_pkg IS

    PROCEDURE execute(stmt VARCHAR2) IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE(stmt);
        EXECUTE IMMEDIATE stmt;
    END;
```

## Práctica 8: Soluciones (continuación)

```
FUNCTION get_type(name VARCHAR2) RETURN VARCHAR2 IS
    proc_type VARCHAR2(30) := NULL;
BEGIN
    /*
     * The ROWNUM = 1 is added to the condition
     * to ensure only one row is returned if the
     * name represents a PACKAGE, which may also
     * have a PACKAGE BODY. In this case, we can
     * only compile the complete package, but not
     * the specification or body as separate
     * components.
     */
    SELECT object_type INTO proc_type
    FROM user_objects
    WHERE object_name = UPPER(name)
    AND ROWNUM = 1;
    RETURN proc_type;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN NULL;
END;

PROCEDURE make(name VARCHAR2) IS
    stmt          VARCHAR2(100);
    proc_type     VARCHAR2(30) := get_type(name);
BEGIN
    IF proc_type IS NOT NULL THEN
        stmt := 'ALTER ' || proc_type || ' ' || name || ' COMPILE';
        execute(stmt);
    ELSE
        RAISE_APPLICATION_ERROR(-20001,
            'Subprogram ''' || name || ''' does not exist');
    END IF;
END make;

PROCEDURE recompile IS
    stmt VARCHAR2(200);
    obj_name user_objects.object_name%type;
    obj_type user_objects.object_type%type;
BEGIN
    FOR objrec IN (SELECT object_name, object_type
                   FROM user_objects
                   WHERE status = 'INVALID'
                   AND object_type <> 'PACKAGE BODY')
    LOOP
        stmt := 'ALTER ' || objrec.object_type || ' ' ||
            objrec.object_name || ' COMPILE';
        execute(stmt);
    END LOOP;
END recompile;
END compile_pkg;
/
```



## Práctica 8: Soluciones (continuación)

```
SHOW ERRORS
```

```
Package created.
```

```
No errors.
```

```
Package body created.
```

```
No errors.
```

- e. Ejecute el procedimiento `compile_pkg.recompile`.

```
EXECUTE compile_pkg.recompile
```

```
PL/SQL procedure successfully completed.
```

- f. Ejecute el archivo de comandos que ha creado en el paso 3c para comprobar el valor de la columna de estado. ¿Aún tiene objetos con estado `INVALID`?

```
SELECT object_name, object_type, status  
FROM USER_OBJECTS  
WHERE status = 'INVALID';
```

```
no rows selected
```

No se devuelve ninguna fila. No hay objetos con estado `INVALID`.

Oracle Internal & OAI Use Only

## Práctica 9: Soluciones

1. Cree una tabla denominada PERSONNEL ejecutando el archivo de comandos  
E:\labs\PLPU\labs\ lab\_09\_01.sql. La tabla contiene los siguientes atributos y tipos de dato:

Nombre de la Columna	Tipo de Dato	Longitud
ID	NUMBER	6
last_name	VARCHAR2	35
review	CLOB	N/A
picture	BLOB	N/A

```
CREATE TABLE personnel (  
  id          NUMBER(6) constraint personnel_id_pk PRIMARY KEY,  
  last_name   VARCHAR2(35),  
  review      CLOB,  
  picture     BLOB);
```

Table created.

2. Inserte dos filas en la tabla PERSONNEL, una por cada empleado 2034 (cuyo apellido es Allen) y para el empleado 2035 (cuyo apellido es Bond). Utilice la función vacía para CLOB y proporcione el valor NULL para BLOB.

```
INSERT INTO  personnel  
VALUES (2034, 'Allen', empty_clob(), NULL);
```

```
INSERT INTO  personnel  
VALUES (2035, 'Bond', empty_clob(), NULL);
```

1 row created.

1 row created.

## Práctica 9: Soluciones (continuación)

3. Examine y ejecute el archivo de comandos E:\labs\PLPU\labs\lab\_09\_03.sql. El archivo de comandos crea una tabla denominada REVIEW\_TABLE. Esta tabla contiene información de la revisión anual de cada empleado. El archivo de comandos también tiene dos sentencias para insertar datos de la revisión de dos empleados.

```
CREATE TABLE review_table (  
  employee_id number,  
  ann_review VARCHAR2(2000));  
  
INSERT INTO review_table  
VALUES (2034,  
  'Very good performance this year. '||  
  'Recommended to increase salary by $500');  
INSERT INTO review_table  
VALUES (2035,  
  'Excellent performance this year. '||  
  'Recommended to increase salary by $1000');  
  
COMMIT;  
  
Table created.  
  
1 row created.  
  
1 row created.  
  
Commit complete.
```

4. Actualice la tabla PERSONNEL.

- a. Rellene la primera fila del CLOB utilizando la siguiente subconsulta en una sentencia UPDATE:

```
SELECT ann_review  
FROM review_table  
WHERE employee_id = 2034;
```

```
UPDATE personnel  
SET review = (SELECT ann_review  
  FROM review_table  
  WHERE employee_id = 2034)  
WHERE last_name = 'Allen';  
  
1 row updated.
```

## Práctica 9: Soluciones (continuación)

- b. Rellene la segunda fila de CLOB , utilizando PL/SQL y el paquete DBMS\_LOB.  
Utilice la siguiente sentencia SELECT para proporcionar un valor para el localizador LOB.

```
SELECT ann_review
FROM   review_table
WHERE  employee_id = 2035;
```

```
DECLARE
  lobloc CLOB;
  text VARCHAR2(2000);
  amount NUMBER ;
  offset INTEGER;
BEGIN
  SELECT ann_review INTO text
  FROM   review_table
  WHERE  employee_id = 2035;
  SELECT review INTO lobloc
  FROM   personnel
  WHERE  last_name = 'Bond' FOR UPDATE;
  offset := 1;
  amount := length(text);
  DBMS_LOB.WRITE (lobloc, amount, offset, text);
  COMMIT;
END;
/

PL/SQL procedure successfully completed.
```

**Si tiene tiempo, realice el siguiente ejercicio:**

5. Cree un procedimiento que agregue un localizador a un archivo binario en la columna PICTURE de la tabla COUNTRIES. El archivo binario es una imagen de una bandera. Los archivos de imagen se nombran según los identificadores de los países. Tiene que cargar un localizador de archivo de imagen en todas las filas de la región de Europa (REGION\_ID = 1) en la tabla COUNTRIES. Debe crear un objeto de DIRECTORY denominado COUNTRY\_PIC que hace referencia a la ubicación de los archivos binarios.

- a. Agregue la columna de imagen a la tabla COUNTRIES utilizando:

```
ALTER TABLE countries ADD (picture BFILE);
```

```
ALTER TABLE countries ADD (picture BFILE);
```

```
Table altered.
```

También puede utilizar el archivo E:\labs\PLPU\labs\ Lab\_09\_05\_a.sql.

## Práctica 9: Soluciones (continuación)

- b. Cree un procedimiento PL/SQL denominado `load_country_image` que utilice `DBMS_LOB.FILEEXISTS` para comprobar si el archivo de la imagen del país existe. Si es así, defina el localizador `BFILE` para el archivo en la columna `PICTURE`; de lo contrario, aparece el mensaje de que el archivo no existe. Utilice el paquete `DBMS_OUTPUT` para proporcionar la información del tamaño del archivo para cada imagen asociada a la columna `PICTURE`.

```
CREATE OR REPLACE PROCEDURE load_country_image (dir IN VARCHAR2) IS
    file          BFILE;
    filename       VARCHAR2(40);
    rec_number     NUMBER;
    file_exists    BOOLEAN;
    CURSOR country_csr IS
        SELECT country_id
        FROM countries
        WHERE region_id = 1
        FOR UPDATE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('LOADING LOCATORS TO IMAGES...');
    FOR rec IN country_csr
    LOOP
        filename := rec.country_id || '.gif';
        file := BFILENAME(dir, filename);
        file_exists := (DBMS_LOB.FILEEXISTS(file) = 1);
        IF file_exists THEN
            DBMS_LOB.FILEOPEN(file);
            UPDATE countries
            SET picture = file
            WHERE CURRENT OF country_csr;
            DBMS_OUTPUT.PUT_LINE('Set Locator to file: ' || filename ||
                ' Size: ' || DBMS_LOB.GETLENGTH(file));
            DBMS_LOB.FILECLOSE(file);
            rec_number := country_csr%ROWCOUNT;
        ELSE
            DBMS_OUTPUT.PUT_LINE('File ' || filename || ' does not exist');
        END IF;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('TOTAL FILES UPDATED: ' || rec_number);
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_LOB.FILECLOSE(file);
            DBMS_OUTPUT.PUT_LINE('Error: ' || to_char(SQLCODE) || SQLERRM);
END load_country_image;
/
SHOW ERRORS

Procedure created.

No errors.
```

## Práctica 9: Soluciones (continuación)

- c. Llame al procedimiento transfiriendo el nombre del objeto de directorio COUNTRY\_PIC como valor de parámetro de literal de cadena.

```
SET SERVEROUTPUT ON
EXECUTE load_country_image('COUNTRY_PIC')

LOADING LOCATORS TO IMAGES...
Set Locator to file: BE.gif Size: 1397
Set Locator to file: CH.gif Size: 1202
Set Locator to file: DE.gif Size: 1271
Set Locator to file: DK.gif Size: 1327
Set Locator to file: FR.gif Size: 1337
Set Locator to file: IT.gif Size: 1322
Set Locator to file: NL.gif Size: 1205
Set Locator to file: UK.gif Size: 2489
TOTAL FILES UPDATED: 8
PL/SQL procedure successfully completed.
```

Oracle Internal & OAI Use Only

## Práctica 10: Soluciones

1. Las filas de la tabla JOBS almacenan los salarios mínimo y máximo permitidos para los distintos valores de JOB\_ID. Le piden que escriba un código para garantizar que el salario de los empleados está dentro del rango permitido por su tipo de trabajo, para operaciones de inserción y actualización.
  - a. Escriba un procedimiento denominado CHECK\_SALARY que acepte dos parámetros, uno para la cadena del identificador de trabajo del empleado y el otro para el salario. El procedimiento utiliza el identificador de trabajo para determinar el salario mínimo y máximo para el trabajo especificado. Si el parámetro del salario, mínimo y máximo incluidos, no está dentro del rango de salarios aparecerá una excepción de aplicación con el mensaje “Invalid salary <sal>. Salaries for job <jobid> must be between <min> and <max>”. Sustituya los distintos elementos del mensaje por los valores que proporcionan los parámetros y las variables rellenos con consultas.

```
CREATE OR REPLACE PROCEDURE check_salary (the_job VARCHAR2, the_salary
NUMBER) IS
    minsal jobs.min_salary%type;
    maxsal jobs.max_salary%type;
BEGIN
    SELECT min_salary, max_salary INTO minsal, maxsal
    FROM jobs
    WHERE job_id = UPPER(the_job);
    IF the_salary NOT BETWEEN minsal AND maxsal THEN
        RAISE_APPLICATION_ERROR(-20100,
            'Invalid salary $'||the_salary||'. '||
            'Salaries for job '|| the_job ||
            ' must be between $'|| minsal ||' and $' || maxsal);
    END IF;
END;
/
SHOW ERRORS

Procedure created.

No errors.
```

- b. Cree un disparador denominado CHECK\_SALARY\_TRG en la tabla EMPLOYEES que arranque ante una operación INSERT o UPDATE en cada fila. El disparador debe llamar al procedimiento CHECK\_SALARY para ejecutar la lógica de negocio. El disparador transferirá el nuevo identificador de trabajo y salario a los parámetros de procedimiento.

```
CREATE OR REPLACE TRIGGER check_salary_trg
BEFORE INSERT OR UPDATE OF job_id, salary
ON employees
FOR EACH ROW
BEGIN
    check_salary(:new.job_id, :new.salary);
END;
/
SHOW ERRORS
```

## Práctica 10: Soluciones (continuación)

```
Trigger created.
```

```
No errors.
```

2. Pruebe CHECK\_SAL\_TRG utilizando los siguientes casos:

- a. Utilice el procedimiento EMP\_PKG.ADD\_EMPLOYEE para agregar a la empleada Eleanor Beh al departamento 30. ¿Qué sucede? ¿Por qué?

```
EXECUTE emp_pkg.add_employee('Eleanor', 'Beh', 30)

BEGIN emp_pkg.add_employee('Eleanor', 'Beh', 30); END;

*

ERROR at line 1:
ORA-20100: Invalid salary $1000. Salaries for job SA_REP must be between
$6000 and $12000
ORA-06512: at "ORA1.CHECK_SALARY", line 9
ORA-06512: at "ORA1.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA1.CHECK_SALARY_TRG'
ORA-06512: at "ORA1.EMP_PKG", line 33
ORA-06512: at "ORA1.EMP_PKG", line 50
ORA-06512: at line 1
```

**El disparador produce una excepción porque el procedimiento EMP\_PKG.ADD\_EMPLOYEE llama a una versión sobrecargada de sí mismo que utiliza un salario por defecto de 1.000 dólares y un identificador de trabajo por defecto de SA\_REP. Sin embargo, la tabla JOBS almacena un salario mínimo de 6.000 dólares para el tipo de trabajo SA\_REP.**

- b. Actualice el salario del empleado 115 a 2.000 dólares. En otra operación de actualización cambie el identificador de trabajo del empleado a HR\_REP. ¿Qué sucede en cada caso?

```
UPDATE employees
  SET salary = 2000
 WHERE employee_id = 115;

UPDATE employees
  *

ERROR at line 1:
ORA-20100: Invalid salary $2000. Salaries for job PU_CLERK must be
between $2500 and $5500
ORA-06512: at "ORA1.CHECK_SALARY", line 9
ORA-06512: at "ORA1.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA1.CHECK_SALARY_TRG'
```



## Práctica 10: Soluciones (continuación)

```
UPDATE employees
  SET job_id = 'HR_REP'
WHERE  employee_id = 115;

UPDATE employees
  *
```

ERROR at line 1:  
ORA-20100: Invalid salary \$3100. Salaries for job HR\_REP must be between \$4000 and \$9000  
ORA-06512: at "ORA1.CHECK\_SALARY", line 9  
ORA-06512: at "ORA1.CHECK\_SALARY\_TRG", line 2  
ORA-04088: error during execution of trigger 'ORA1.CHECK\_SALARY\_TRG'

**La primera sentencia de actualización no consigue definir el salario en 2.000 dólares. La regla del disparador de comprobación del salario no consigue actualizar la operación porque el nuevo salario para el empleado 115 es menor que el mínimo que permite el trabajo PU\_CLERK.**

**La segunda actualización no consigue cambiar el trabajo del empleado porque el salario actual de éste es de 3.100 dólares, menor que el mínimo para un nuevo trabajo de HR\_REP.**

- c. Actualice el salario del empleado 115 a 2.800 dólares. ¿Qué sucede?

```
UPDATE employees
  SET salary = 2800
WHERE  employee_id = 115;

1 row updated.
```

**La operación de actualización se ha realizado correctamente porque el nuevo salario está dentro del rango aceptado para el identificador de trabajo actual.**

3. Actualice el disparador CHECK\_SALARY\_TRG para que arranque sólo cuando los valores del identificador de trabajo o el salario hayan cambiado en realidad.
- a. Implemente la regla de negocio utilizando una cláusula WHEN para comprobar si los valores JOB\_ID o SALARY han cambiado.
- Nota:** Asegúrese de que la condición maneja NULL en los valores de OLD.column\_name si se realiza una operación INSERT; si no es así, la operación de inserción fallará.

```
CREATE OR REPLACE TRIGGER check_salary_trg
BEFORE INSERT OR UPDATE OF job_id, salary
ON employees FOR EACH ROW
WHEN (new.job_id <> NVL(old.job_id, '?') OR
      new.salary <> NVL(old.salary, 0))
BEGIN
  check_salary(:new.job_id, :new.salary);
END;
/
```

## Práctica 10: Soluciones (continuación)

```
SHOW ERRORS
```

```
Trigger created.
```

```
No errors.
```

- b. Compruebe el disparador ejecutando el procedimiento EMP\_PKG.ADD\_EMPLOYEE con los siguientes valores de parámetros: first\_name='Eleanor', last name='Beh', email='EBEH', job='IT\_PROG', sal=5000.

```
BEGIN
```

```
emp_pkg.add_employee('Eleanor', 'Beh', 'EBEH',  
                     job => 'IT_PROG', sal => 5000);
```

```
END;
```

```
/
```

```
PL/SQL procedure successfully completed.
```

- c. Actualice a los empleados con un trabajo IT\_PROG incrementando su salario en 2.000 dólares. ¿Qué sucede?

```
UPDATE employees  
SET salary = salary + 2000  
WHERE job_id = 'IT_PROG';
```

```
UPDATE employees  
*
```

```
ERROR at line 1:
```

```
ORA-20100: Invalid salary $11000. Salaries for job IT_PROG must be  
between $4000 and $10000
```

```
ORA-06512: at "ORA1.CHECK_SALARY", line 9
```

```
ORA-06512: at "ORA1.CHECK_SALARY_TRG", line 2
```

```
ORA-04088: error during execution of trigger 'ORA1.CHECK_SALARY_TRG'
```

**El salario de un empleado en el tipo de trabajo especificado excede el salario máximo para ese tipo de trabajo. No se ha actualizado ningún salario de los empleados con tipo de trabajo IT\_PROG.**

## Práctica 10: Soluciones (continuación)

- d. Actualice a 9.000 dólares el salario de Eleanor Beh.

**Indicación:** Utilice una sentencia UPDATE con una subconsulta en la cláusula WHERE.

¿Qué sucede?

```
UPDATE employees
  SET salary = 9000
WHERE employee_id = (SELECT employee_id
                     FROM employees
                     WHERE last_name = 'Beh');

1 row updated
```

**La operación de actualización se realizó correctamente porque el salario es válido para el tipo de trabajo del empleado.**

- e. Cambie el trabajo de Eleanor Beh a ST\_MAN utilizando otra sentencia UPDATE con una subconsulta. ¿Qué sucede?

```
UPDATE employees
  set job_id = 'ST_MAN'
WHERE employee_id = (SELECT employee_id
                     FROM employees
                     WHERE last_name = 'Beh');

UPDATE employees
  *
```

ERROR at line 1:  
ORA-20100: Invalid salary \$9000. Salaries for job ST\_MAN must be between \$5500 and \$8500  
ORA-06512: at "ORA1.CHECK\_SALARY", line 9  
ORA-06512: at "ORA1.CHECK\_SALARY\_TRG", line 2  
ORA-04088: error during execution of trigger 'ORA1.CHECK\_SALARY\_TRG'

**El salario máximo del nuevo tipo de trabajo es menor que el salario actual del empleado. Por lo tanto, la operación de actualización falla.**

## Práctica 10: Soluciones (continuación)

4. Se le pide que evite que se suprima a los empleados durante las horas laborables.
  - a. Escriba un disparador de sentencia denominado DELETE\_EMP\_TRG en la tabla EMPLOYEES para evitar que las filas se supriman durante horas laborables entre semana, es decir, de las 9:00 a.m. a las 6:00 p.m.

```
CREATE OR REPLACE TRIGGER delete_emp_trg
BEFORE DELETE ON employees
DECLARE
    the_day VARCHAR2(3) := TO_CHAR(SYSDATE, 'DY');
    the_hour PLS_INTEGER := TO_NUMBER(TO_CHAR(SYSDATE, 'HH24'));
BEGIN
    IF (the_hour BETWEEN 9 AND 18) AND (the_day NOT IN ('SAT','SUN')) THEN
        RAISE_APPLICATION_ERROR(-20150,
            'Employee records cannot be deleted during the week 9am and 6pm');
    END IF;
END;
/
SHOW ERRORS

Trigger created.

No errors.
```

- b. Intente suprimir los empleados con JOB\_ID de SA\_REP que no estén asignados a un departamento.

**Nota:** Empleado Grant con identificador 178.

```
DELETE FROM employees
WHERE job_id = 'SA_REP'
AND    department_id IS NULL;

DELETE FROM employees
      *
```

ERROR at line 1:  
ORA-20150: Employee records cannot be deleted during the week 9am and 6pm  
ORA-06512: at "ORA1.DELETE\_EMP\_TRG", line 6  
ORA-04088: error during execution of trigger 'ORA1.DELETE\_EMP\_TRG'

## Práctica 11: Soluciones

1. Los empleados reciben un aumento de sueldo automáticamente si el salario mínimo de un trabajo se aumenta a un valor superior que su salario actual. Implemente este requisito con un procedimiento empaquetado al que llame el disparador de la tabla JOBS. Cuando intenta actualizar el salario mínimo en la tabla JOBS e intenta actualizar el salario de los empleados, el disparador CHECK\_SALARY intenta leer la tabla JOBS, que está sujeta a cambios y obtendrá una excepción de tabla mutante que se resuelve creando un nuevo paquete y disparadores adicionales.
  - a. Actualice el paquete EMP\_PKG (de la práctica 7) agregando un procedimiento denominado SET\_SALARY que actualiza los salarios de los empleados. Este procedimiento acepta dos parámetros: el identificador de trabajo para estos salarios que es posible que tenga que actualizarse y el nuevo salario mínimo para el identificador de trabajo. El procedimiento define todos los salarios de los empleados en el mínimo para su trabajo si el salario actual es menor que el nuevo valor mínimo.

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  TYPE emp_tabtype IS TABLE OF employees%ROWTYPE;
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30);
  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    deptid     employees.department_id%TYPE);
  PROCEDURE get_employee(
    empid IN employees.employee_id%TYPE,
    sal   OUT employees.salary%TYPE,
    job   OUT employees.job_id%TYPE);
  FUNCTION get_employee(emp_id employees.employee_id%type)
    return employees%rowtype;
  FUNCTION get_employee(family_name employees.last_name%type)
    return employees%rowtype;
  PROCEDURE get_employees(dept_id employees.department_id%type);
  PROCEDURE init_departments;
  PROCEDURE print_employee(emprec employees%rowtype);
  PROCEDURE set_salary(jobid VARCHAR2, min_salary NUMBER);
END emp_pkg;
/
SHOW ERRORS
```

## Práctica 11: Soluciones (continuación)

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  TYPE boolean_tabtype IS TABLE OF BOOLEAN
    INDEX BY BINARY_INTEGER;
  valid_departments boolean_tabtype;
  emp_table          emp_tabtype;

  FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
    RETURN BOOLEAN;

  PROCEDURE add_employee(
    first_name employees.first_name%TYPE,
    last_name  employees.last_name%TYPE,
    email      employees.email%TYPE,
    job        employees.job_id%TYPE      DEFAULT 'SA_REP',
    mgr        employees.manager_id%TYPE  DEFAULT 145,
    sal        employees.salary%TYPE      DEFAULT 1000,
    comm       employees.commission_pct%TYPE DEFAULT 0,
    deptid     employees.department_id%TYPE DEFAULT 30) IS

    PROCEDURE audit_newemp IS
      PRAGMA AUTONOMOUS_TRANSACTION;
      user_id VARCHAR2(30) := USER;
    BEGIN
      INSERT INTO log_newemp (entry_id, user_id, log_time, name)
      VALUES (log_newemp_seq.NEXTVAL, user_id, sysdate,
              first_name||' '||last_name);
      COMMIT;
    END audit_newemp;

  BEGIN
    IF valid_deptid(deptid) THEN
      audit_newemp;
      INSERT INTO employees(employee_id, first_name, last_name, email,
                           job_id, manager_id, hire_date, salary, commission_pct, department_id)
      VALUES (employees_seq.NEXTVAL, first_name, last_name, email,
              job, mgr, TRUNC(SYSDATE), sal, comm, deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204,
                              'Invalid department ID. Try again.');
```

END IF;

```
END add_employee;

PROCEDURE add_employee(
  first_name employees.first_name%TYPE,
  last_name  employees.last_name%TYPE,
  deptid     employees.department_id%TYPE) IS
  email      employees.email%type;
BEGIN
  email := UPPER(SUBSTR(first_name, 1, 1)||SUBSTR(last_name, 1, 7));
  add_employee(first_name, last_name, email, deptid => deptid);
END;
```

## Práctica 11: Soluciones (continuación)

```
PROCEDURE get_employee(  
    empid IN employees.employee_id%TYPE,  
    sal    OUT employees.salary%TYPE,  
    job    OUT employees.job_id%TYPE) IS  
BEGIN  
    SELECT  salary, job_id  
    INTO    sal, job  
    FROM    employees  
    WHERE   employee_id = empid;  
END get_employee;  
  
FUNCTION get_employee(emp_id employees.employee_id%type)  
    return employees%rowtype IS  
    emprec employees%rowtype;  
BEGIN  
    SELECT * INTO emprec  
    FROM employees  
    WHERE employee_id = emp_id;  
    RETURN emprec;  
END;  
  
FUNCTION get_employee(family_name employees.last_name%type)  
    return employees%rowtype IS  
    emprec employees%rowtype;  
BEGIN  
    SELECT * INTO emprec  
    FROM employees  
    WHERE last_name = family_name;  
    RETURN emprec;  
END;  
  
PROCEDURE get_employees(dept_id employees.department_id%type) IS  
BEGIN  
    SELECT * BULK COLLECT INTO emp_table  
    FROM EMPLOYEES  
    WHERE department_id = dept_id;  
END;  
  
PROCEDURE init_departments IS  
BEGIN  
    FOR rec IN (SELECT department_id FROM departments)  
    LOOP  
        valid_departments(rec.department_id) := TRUE;  
    END LOOP;  
END;
```

## Práctica 11: Soluciones (continuación)

```
PROCEDURE print_employee(emprec employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(emprec.department_id || ' ' ||
                          emprec.employee_id || ' ' ||
                          emprec.first_name || ' ' ||
                          emprec.last_name || ' ' ||
                          emprec.job_id || ' ' ||
                          emprec.salary);
END;

PROCEDURE show_employees IS
BEGIN
    IF emp_table IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('Employees in Package table');
        FOR i IN 1 .. emp_table.COUNT
        LOOP
            print_employee(emp_table(i));
        END LOOP;
    END IF;
END show_employees;

FUNCTION valid_deptid(deptid IN departments.department_id%TYPE)
RETURN BOOLEAN IS
    dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

PROCEDURE set_salary(jobid VARCHAR2, min_salary NUMBER) IS
    CURSOR empcsr IS
        SELECT employee_id
        FROM employees
        WHERE job_id = jobid AND salary < min_salary;
BEGIN
    FOR emprec IN empcsr
    LOOP
        UPDATE employees
        SET salary = min_salary
        WHERE employee_id = emprec.employee_id;
    END LOOP;
END set_salary;

BEGIN
    init_departments;
END emp_pkg;
/
SHOW ERRORS
```



## Práctica 11: Soluciones (continuación)

Package created.

No errors.

Package body created.

No errors.

- b. Cree un disparador de fila denominado UPD\_MINSALARY\_TRG en la tabla JOBS que llame al procedimiento EMP\_PKG.SET\_SALARY, cuando el salario mínimo de la tabla JOBS se actualice para un identificador de trabajo especificado.

```
CREATE OR REPLACE TRIGGER upd_minsalary_trg
AFTER UPDATE OF min_salary ON JOBS
FOR EACH ROW
BEGIN
    emp_pkg.set_salary(:new.job_id, :new.min_salary);
END;
/
SHOW ERRORS
```

Trigger created.

No errors.

- c. Escriba una consulta para mostrar el identificador de empleado, el apellido, el identificador de trabajo, el salario actual y el salario mínimo para los empleados que sean programadores, es decir, su JOB\_ID es 'IT\_PROG'. A continuación, actualice el salario mínimo en la tabla JOBS para aumentarlo en 1.000 dólares. ¿Qué sucede?

```
SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'IT_PROG';

UPDATE jobs
SET min_salary = min_salary + 1000
WHERE job_id = 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	SALARY
103	Hunold	9000
104	Ernst	6000
105	Austin	4800
106	Pataballa	4800
107	Lorentz	4200
226	Beh	9000

6 rows selected.

## Práctica 11: Soluciones (continuación)

```
UPDATE jobs
*

ERROR at line 1:
ORA-04091: table ORA1.JOBS is mutating, trigger/function may not see it
ORA-06512: at "ORA1.CHECK_SALARY", line 5
ORA-06512: at "ORA1.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA1.CHECK_SALARY_TRG'
ORA-06512: at "ORA1.EMP_PKG", line 140
ORA-06512: at "ORA1.UPD_MINSALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA1.UPD_MINSALARY_TRG'
```

La actualización de la columna **MIN\_SALARY** para el trabajo '**IT\_PROG**' falla porque el disparador **UPD\_MINSALARY\_TRG** de la tabla **JOBS** intenta actualizar los salarios de los empleados llamando al procedimiento **EMP\_PKG.SET\_SALARY**. El procedimiento **SET\_SALARY** hace que el disparador **CHECK\_SALARY\_TRG** arranque (efecto en cascada). El disparador **CHECK\_SALARY\_TRG** llama al procedimiento **CHECK\_SALARY**, que intenta leer los datos de la tabla **JOBS**, encontrando una excepción de tabla mutante en la tabla **JOBS**, que es la tabla que está sujeta a la operación **UPDATE** original.

2. Para resolver el problema de la tabla mutante, cree **JOBS\_PKG** para mantener en memoria una copia de las filas de la tabla **JOBS**. A continuación, se modifica el procedimiento **CHECK\_SALARY** para utilizar los datos del paquete en vez de emitir una consulta en una tabla mutante para evitar la excepción. Sin embargo, se debe crear un disparador de sentencia **BEFORE INSERT OR UPDATE** en la tabla **EMPLOYEES** para inicializar el estado del paquete **JOBS\_PKG** antes de que el disparador de fila **CHECK\_SALARY** se arranque.

- a. Cree un nuevo paquete denominado **JOBS\_PKG** con la siguiente especificación.

```
PROCEDURE initialize;
FUNCTION get_minsalary(jobid VARCHAR2) RETURN NUMBER;
FUNCTION get_maxsalary(jobid VARCHAR2) RETURN NUMBER;
PROCEDURE set_minsalary(jobid VARCHAR2,min_salary NUMBER);
PROCEDURE set_maxsalary(jobid VARCHAR2,max_salary NUMBER);
```

```
CREATE OR REPLACE PACKAGE jobs_pkg IS
  PROCEDURE initialize;
  FUNCTION get_minsalary(jobid VARCHAR2) RETURN NUMBER;
  FUNCTION get_maxsalary(jobid VARCHAR2) RETURN NUMBER;
  PROCEDURE set_minsalary(jobid VARCHAR2, min_salary NUMBER);
  PROCEDURE set_maxsalary(jobid VARCHAR2, max_salary NUMBER);
END jobs_pkg;
/
SHOW ERRORS

Package created.

No errors.
```

## Práctica 11: Soluciones (continuación)

- b. Implemente el cuerpo de JOBS\_PKG donde:

Se declara una tabla PL/SQL index-by privada denominada jobs\_tabtype indexada por un tipo de cadena basada en JOBS.JOB\_ID%TYPE.

Se declara una variable privada denominada jobstab basada en jobs\_tabtype.

El procedimiento INITIALIZE lee las filas en la tabla JOBS con un bucle de cursor y utiliza el valor JOB\_ID para el índice jobstab que se le asigne a la fila correspondiente.

La función GET\_MINSALARY utiliza un parámetro jobid como índice para jobstab y devuelve min\_salary para ese elemento.

La función GET\_MAXSALARY utiliza un parámetro jobid como índice para jobstab y devuelve max\_salary para ese elemento.

El procedimiento SET\_MINSALARY utiliza jobid como índice para jobstab para definir el campo de su elemento min\_salary en el valor del parámetro min\_salary.

El procedimiento SET\_MAXSALARY utiliza jobid como índice para jobstab para definir el campo de su elemento max\_salary en el valor del parámetro max\_salary.

```
CREATE OR REPLACE PACKAGE BODY jobs_pkg IS
  TYPE jobs_tabtype IS TABLE OF jobs%rowtype
    INDEX BY jobs.job_id%type;
  jobstab jobs_tabtype;

  PROCEDURE initialize IS
  BEGIN
    FOR jobrec IN (SELECT * FROM jobs)
    LOOP
      jobstab(jobrec.job_id) := jobrec;
    END LOOP;
  END initialize;

  FUNCTION get_minsalary(jobid VARCHAR2) RETURN NUMBER IS
  BEGIN
    RETURN jobstab(jobid).min_salary;
  END get_minsalary;

  FUNCTION get_maxsalary(jobid VARCHAR2) RETURN NUMBER IS
  BEGIN
    RETURN jobstab(jobid).max_salary;
  END get_maxsalary;

  PROCEDURE set_minsalary(jobid VARCHAR2, min_salary NUMBER) IS
  BEGIN
    jobstab(jobid).max_salary := min_salary;
  END set_minsalary;
```

## Práctica 11: Soluciones (continuación)

```
PROCEDURE set_maxsalary(jobid VARCHAR2, max_salary NUMBER) IS
BEGIN
    jobstab(jobid).max_salary := max_salary;
END set_maxsalary;

END jobs_pkg;
/
SHOW ERRORS

Package body created.

No errors.
```

- c. Copie el procedimiento CHECK\_SALARY de la práctica 10, ejercicio 1ª, y modifique el código sustituyendo la consulta de la tabla JOBS con sentencias para definir las variables locales minsal y maxsal con valores de los datos JOBS\_PKG llamando a las funciones GET\_\*SALARY apropiadas. Este paso eliminará la excepción de disparador mutante.

```
CREATE OR REPLACE PROCEDURE check_salary (the_job VARCHAR2, the_salary
NUMBER) IS
    minsal jobs.min_salary%type;
    maxsal jobs.max_salary%type;
BEGIN
    /*
    ** Commented out to avoid mutating trigger exception on the JOBS table
    SELECT min_salary, max_salary INTO minsal, maxsal
    FROM jobs
    WHERE job_id = UPPER(the_job);
    */
    minsal := jobs_pkg.get_minsalary(UPPER(the_job));
    maxsal := jobs_pkg.get_maxsalary(UPPER(the_job));
    IF the_salary NOT BETWEEN minsal AND maxsal THEN
        RAISE_APPLICATION_ERROR(-20100,
            'Invalid salary $'||the_salary||'. '||
            'Salaries for job '|| the_job ||
            ' must be between $'|| minsal ||' and $' || maxsal);
    END IF;
END;
/
SHOW ERRORS

Procedure created.

No errors.
```

## Práctica 11: Soluciones (continuación)

- d. Implemente un disparador de sentencia BEFORE INSERT OR UPDATE denominado INIT\_JOBPKG\_TRG que utilice la sintaxis CALL para llamar al procedimiento JOBS\_PKG.INITIALIZE con el fin de garantizar que el estado del paquete sea actual antes de que se realicen las operaciones DML.

```
CREATE OR REPLACE TRIGGER init_jobpkg_trg
BEFORE INSERT OR UPDATE ON jobs
CALL jobs_pkg.initialize
/
SHOW ERRORS

Trigger created.

No errors.
```

- e. Pruebe los cambios de código ejecutando la consulta para mostrar los empleados que son programadores, a continuación, emita una sentencia de actualización para aumentar el salario mínimo del tipo de trabajo IT\_PROG en 1000 en la tabla JOBS, seguido de una consulta sobre los empleados con tipo de trabajo IT\_PROG para comprobar los cambios resultantes. ¿Los salarios de qué empleados se han definido en el mínimo para su trabajo?

```
SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'IT_PROG';

UPDATE jobs
SET min_salary = min_salary + 1000
WHERE job_id = 'IT_PROG';

SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	SALARY
103	Hunold	9000
104	Ernst	6000
105	Austin	4800
106	Pataballa	4800
107	Lorentz	4200
226	Beh	9000

6 rows selected.

1 row updated.

## Práctica 11: Soluciones (continuación)

EMPLOYEE_ID	LAST_NAME	SALARY
103	Hunold	9000
104	Ernst	6000
105	Austin	5000
106	Pataballa	5000
107	Lorentz	5000
226	Beh	9000

6 rows selected.

A los empleados cuyos apellidos son **Austin**, **Pataballa** y **Lorentz** se les ha actualizado el salario. No se ha producido ninguna excepción durante este proceso y se ha implementado una solución para la excepción de disparador de tabla mutante.

3. Debido a que `CHECK_SALARY_TRG` arranca el procedimiento `CHECK_SALARY`, antes de insertar o actualizar un empleado, debe comprobar si aún funciona como se esperaba.
  - a. Pruébalo agregando un nuevo empleado mediante `EMP_PKG.ADD_EMPLOYEE` con los siguientes parámetros: ('Steve', 'Morse', 'SMORSE', sal => 6500). ¿Qué sucede?

```
EXECUTE emp_pkg.add_employee('Steve', 'Morse', 'SMORSE', sal => 6500)

BEGIN emp_pkg.add_employee('Steve', 'Morse', 'SMORSE', sal => 6500); END;

*

ERROR at line 1:
ORA-01403: no data found
ORA-01403: no data found
ORA-06512: at "ORA1.JOBS_PKG", line 16
ORA-06512: at "ORA1.CHECK_SALARY", line 11
ORA-06512: at "ORA1.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA1.CHECK_SALARY_TRG'
ORA-06512: at "ORA1.EMP_PKG", line 33
ORA-06512: at line 1
```

El problema aquí es que el procedimiento `CHECK_SALARY` intenta leer el valor de las variables de estado del paquete que todavía no se han inicializado. Esto se debe a que se ha modificado para leer los salarios mínimo y máximo desde `JOBS_PK`, que almacenará los datos en una tabla PL/SQL. Cuando `CHECK_SALARY` intenta llamar a `JOBS_PKG.GET_MINSALARY` y a `JOBS_PKG.GET_MAXSALARY`, éstos devuelven excepciones `NO_DATA_FOUND` que provocan que el disparador y la operación de inserción fallen. Esto se puede solucionar con un disparador de sentencia **BEFORE** que llame a `JOBS_PKG.INITIALIZE` para garantizar que el estado de `JOBS_PKG` se ha definido antes de leerlo. Esto se realizará en el siguiente ejercicio (3b).

## Práctica 11: Soluciones (continuación)

- b. Para corregir el problema encontrado al agregar o actualizar un empleado, cree un disparador de sentencia BEFORE INSERT OR UPDATE denominado EMPLOYEE\_INITJOBS\_TRG en la tabla EMPLOYEES que llame al procedimiento JOBS\_PKG.INITIALIZE. Utilice la sintaxis CALL en el cuerpo del disparador.

```
CREATE TRIGGER employee_initjobs_trg
BEFORE INSERT OR UPDATE OF job_id, salary ON employees
CALL jobs_pkg.initialize
/
```

Trigger created.

- c. Pruebe el disparador agregando el empleado Steve Morse de nuevo. Confirme el registro insertado en la tabla employees mostrando el identificador de empleado, nombre y apellido, salario, identificador de trabajo e identificador de departamento.

```
EXECUTE emp_pkg.add_employee('Steve', 'Morse', 'SMORSE', sal => 6500)
```

PL/SQL procedure successfully completed.

```
SELECT employee_id, first_name, last_name, salary, job_id, department_id
FROM employees
WHERE last_name = 'Morse';
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	JOB_ID	DEPARTMENT_ID
241	Steve	Morse	6500	SA_REP	30

## Práctica 12: Soluciones

1. Modifique el parámetro `PLSQL_COMPILER_FLAGS` para activar la compilación nativa para la sesión y compile cualquier subprograma que haya escrito.

- a. Ejecute el comando `ALTER SESSION` para activar la compilación nativa.

```
ALTER SESSION SET PLSQL_COMPILER_FLAGS = 'NATIVE';  
  
Session altered.
```

- b. Compile el procedimiento `EMPLOYEE_REPORT`. ¿Qué sucede durante la compilación?

```
ALTER PROCEDURE employee_report COMPILE;  
  
Procedure altered.
```

**Se genera una biblioteca compartida en un directorio especificado por el parámetro de base de datos `plsql_native_library_dir`. El nombre de la biblioteca lleva como prefijos el nombre del objeto y del usuario que lo ha compilado, como se muestra en el siguiente ejemplo: `EMPLOYEE_REPORT_ORA1_P_50344.so`.**

- c. Ejecute `EMPLOYEE_REPORT` con el valor `'UTL_FILE'` como primer parámetro y `'native_salrepXX.txt'` donde XX es el número de estudiante.

```
EXECUTE employee_report('UTL_FILE', 'native_salrep01.txt')  
  
PL/SQL procedure successfully completed.
```

- d. Cambie la compilación para utilizar compilación interpretada.

```
ALTER SESSION SET PLSQL_COMPILER_FLAGS = 'INTERPRETED';  
  
Session altered.
```

2. En `COMPILE_PKG` (de la práctica 6), agregue una versión sobrecargada del procedimiento `MAKE`, que compilará un procedimiento, función o paquete con nombre.

- a. En la especificación, declare un procedimiento `MAKE` que acepte dos argumentos de cadena, uno para el nombre de la construcción PL/SQL y el otro para el tipo de programa PL/SQL, como `PROCEDURE`, `FUNCTION`, `PACKAGE`, o `PACKAGE BODY`.

```
CREATE OR REPLACE PACKAGE compile_pkg IS  
    dir VARCHAR2(100) := 'UTL_FILE';  
    PROCEDURE make(name VARCHAR2);  
    PROCEDURE make(name VARCHAR2, objtype VARCHAR2);  
    PROCEDURE regenerate(name VARCHAR2);  
END compile_pkg;  
/  
SHOW ERRORS  
  
Package created.  
  
No errors.
```



## Práctica 12: Soluciones (continuación)

- b. En el cuerpo, escriba el procedimiento MAKE para llamar al paquete DBMS\_WARNINGS para suprimir la categoría PERFORMANCE. Sin embargo, guarde los valores actuales de las advertencias del compilador antes de modificarlos. A continuación, escriba una sentencia EXECUTE IMMEDIATE para compilar el objeto PL/SQL utilizando una sentencia ALTER...COMPILE adecuada con los valores de parámetros proporcionados. Por último, restaure los valores de las advertencias del compilador que existían para el entorno de llamada antes de que se llame al procedimiento.

```
CREATE OR REPLACE PACKAGE BODY compile_pkg IS

    PROCEDURE execute(stmt VARCHAR2) IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE(stmt);
        EXECUTE IMMEDIATE stmt;
    END;

    FUNCTION get_type(name VARCHAR2) RETURN VARCHAR2 IS
        proc_type VARCHAR2(30) := NULL;
    BEGIN
        /*
         * The ROWNUM = 1 is added to the condition
         * to ensure only one row is returned if the
         * name represents a PACKAGE, which may also
         * have a PACKAGE BODY. In this case, we can
         * only compile the complete package, but not
         * the specification or body as separate
         * components.
         */
        SELECT object_type INTO proc_type
        FROM user_objects
        WHERE object_name = UPPER(name)
        AND ROWNUM = 1;
        RETURN proc_type;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN NULL;
    END;

    PROCEDURE make(name VARCHAR2) IS
        stmt          VARCHAR2(100);
        proc_type     VARCHAR2(30) := get_type(name);
    BEGIN
        IF proc_type IS NOT NULL THEN
            stmt := 'ALTER ' || proc_type || ' ' || name || ' COMPILE';
            execute(stmt);
        ELSE
            RAISE_APPLICATION_ERROR(-20001,
                'Subprogram ''' || name || ''' does not exist');
        END IF;
    END make;
```

## Práctica 12: Soluciones (continuación)

```
PROCEDURE make(name VARCHAR2, objtype VARCHAR2) IS
  stmt          VARCHAR2(100);
  warn_value varchar2(200);
BEGIN
  stmt := 'ALTER ' || objtype || ' ' || name || ' COMPILE';
  warn_value := dbms_warning.get_warning_setting_string;
  dbms_warning.add_warning_setting_cat(
    'PERFORMANCE', 'DISABLE', 'SESSION');
  execute(stmt);
  dbms_warning.set_warning_setting_string(
    warn_value, 'SESSION');
END make;

PROCEDURE regenerate (name VARCHAR2) IS
  file UTL_FILE.FILE_TYPE;
  filename VARCHAR2(100) := LOWER(USER || '_' || name || '.sql');
  proc_type VARCHAR2(30) := get_type(name);
BEGIN
  IF proc_type IS NOT NULL THEN
    file := UTL_FILE.FOPEN(dir, filename, 'w');
    UTL_FILE.PUT(file,
      DBMS_METADATA.GET_DDL(proc_type, UPPER(name)));
    UTL_FILE.FCLOSE(file);
  ELSE
    RAISE_APPLICATION_ERROR(-20001,
      'Object with ''' || name || ''' does not exist');
  END IF;

  END regenerate;

END compile_pkg;
/
SHOW ERRORS

Package body created.

No errors.
```

## Práctica 12: Soluciones (continuación)

3. Escriba un nuevo paquete PL/SQL denominado TEST\_PKG que contenga un procedimiento denominado GET\_EMPLOYEES que utilice un argumento IN OUT.
- a. En la especificación, declare el procedimiento GET\_EMPLOYEES con dos parámetros, un parámetro de entrada que especifique un identificador de departamento y un parámetro IN OUT que especifique una tabla PL/SQL de filas de empleados.
- Indicación:** Tendrá que declarar un TYPE en la especificación del paquete para el tipo de dato del parámetro de la tabla PL/SQL.

```
CREATE OR REPLACE PACKAGE test_pkg IS
  TYPE emp_tabtype IS TABLE OF employees%ROWTYPE;
  PROCEDURE get_employees(dept_id NUMBER, emps IN OUT emp_tabtype);
END test_pkg;
/
SHOW ERRORS

Package created.

No errors.
```

- b. En el cuerpo del paquete, implemente el procedimiento GET\_EMPLOYEES para recuperar todas las filas de empleados de un departamento especificado en el parámetro IN OUT en la tabla PL/SQL.
- Indicación:** Utilice la sintaxis SELECT ... BULK COLLECT INTO para simplificar el código.

```
CREATE OR REPLACE PACKAGE BODY test_pkg IS
  PROCEDURE get_employees(dept_id NUMBER, emps IN OUT emp_tabtype) IS
  BEGIN
    SELECT * BULK COLLECT INTO emps
    FROM employees
    WHERE department_id = dept_id;
  END get_employees;
END test_pkg;
/
SHOW ERRORS

Package body created.

No errors.
```

4. Utilice la sentencia ALTER SESSION para definir PLSQL\_WARNINGS para que todas las categorías de las advertencias del compilador estén activadas.

```
ALTER SESSION SET PLSQL_WARNINGS = 'ENABLE:ALL';

Session altered.
```

## Práctica 12: Soluciones (continuación)

5. Recompile el TEST\_PKG creado en una tarea anterior. ¿Qué advertencias del compilador (si hay alguna) se muestran?

```
ALTER PACKAGE test_pkg COMPILE;  
SHOW ERRORS
```

```
SP2-0809: Package altered with compilation warnings  
Errors for PACKAGE TEST_PKG:
```

LINE/COL	ERROR
3/43	PLW-07203: parameter 'EMPS' may benefit from use of the NOCOPY compiler hint

6. Escriba un bloque anónimo PL/SQL para compilar el paquete TEST\_PKG utilizando el procedimiento sobrecargado COMPILE\_PKG.MAKE con dos parámetros. El bloque anónimo mostrará el valor de la cadena de advertencia de la sesión actual antes y después de que llame al procedimiento COMPILE\_PKG.MAKE. ¿Ve algún mensaje de advertencia? Confirme las observaciones ejecutando el comando SHOW ERRORS PACKAGE para TEST\_PKG.

```
BEGIN  
  dbms_output.put_line('Warning level before compilation: ' ||  
    dbms_warning.get_warning_setting_string);  
  compile_pkg.make('TEST_PKG', 'PACKAGE');  
  dbms_output.put_line('Warning level after compilation: ' ||  
    dbms_warning.get_warning_setting_string);  
END;  
/  
SHOW ERRORS PACKAGE test_pkg;
```

```
Warning level before compilation: ENABLE:ALL  
ALTER PACKAGE TEST_PKG COMPILE  
Warning level after compilation: ENABLE:ALL  
PL/SQL procedure successfully completed.
```

```
No errors.
```

**Nota:** La definición actual del nivel de advertencia deberá ser la misma antes y después de la llamada al procedimiento **COMPILE\_PKG.MAKE**, que modifica los valores para suprimir advertencias y restaura el valor original antes de devolverlo al emisor de la llamada.

# B

## Descripciones de las Tablas y Datos

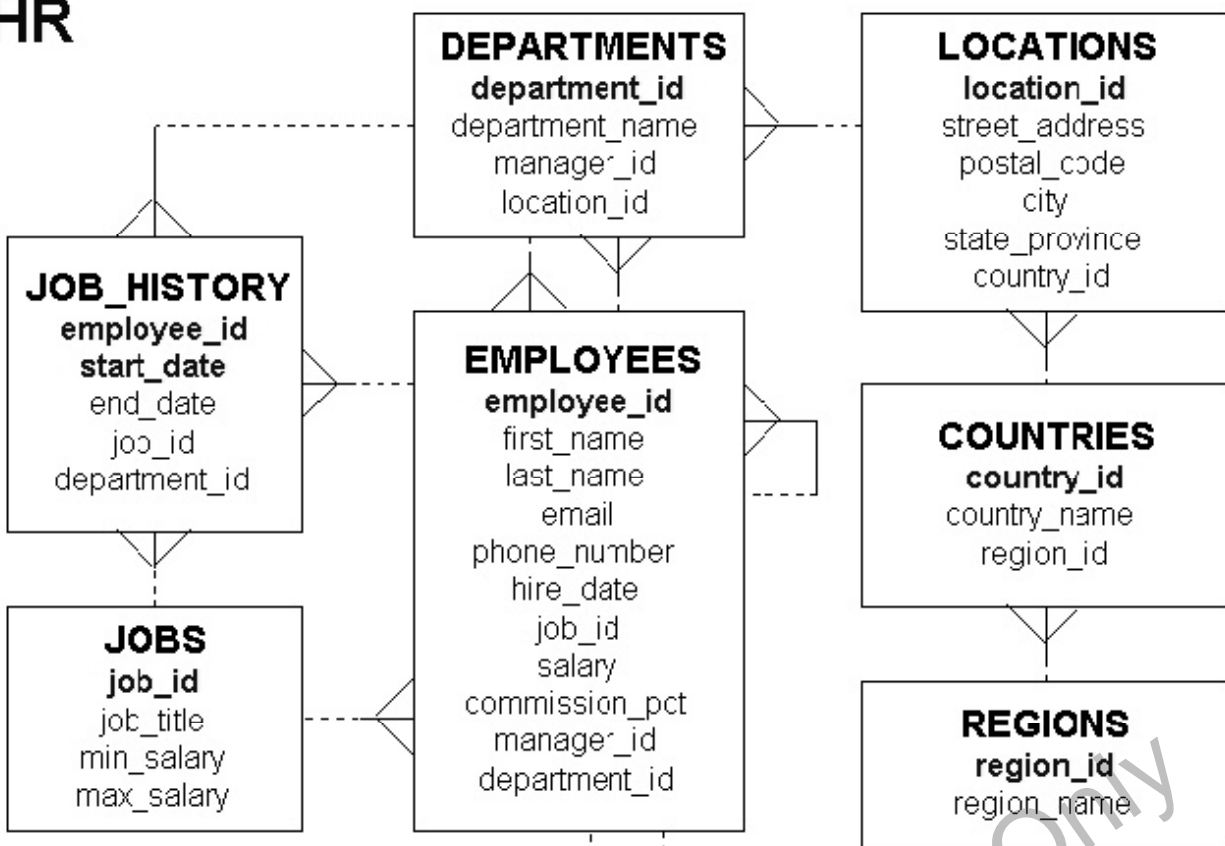
ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

## Diagrama de Entidad/Relación

# HR



## Tablas del Esquema

```
SELECT * FROM tab;
```

TNAME	TABTYPE	CLUSTERID
COUNTRIES	TABLE	
DEPARTMENTS	TABLE	
EMPLOYEES	TABLE	
EMP_DETAILS_VIEW	VIEW	
JOBS	TABLE	
JOB_HISTORY	TABLE	
LOCATIONS	TABLE	
REGIONS	TABLE	

8 rows selected.

Oracle Internal & OAI Use Only

## Tabla REGIONS

DESCRIBE regions

Name	Null?	Type
REGION_ID	NOT NULL	NUMBER
REGION_NAME		VARCHAR2(25)

SELECT \* FROM regions;

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa

Oracle Internal & OAI Use Only



## Tabla COUNTRIES

DESCRIBE countries

Name	Null?	Type
COUNTRY_ID	NOT NULL	CHAR(2)
COUNTRY_NAME		VARCHAR2(40)
REGION_ID		NUMBER

SELECT \* FROM countries;

CO	COUNTRY_NAME	REGION_ID
AR	Argentina	2
AU	Australia	3
BE	Belgium	1
BR	Brazil	2
CA	Canada	2
CH	Switzerland	1
CN	China	3
DE	Germany	1
DK	Denmark	1
EG	Egypt	4
FR	France	1
HK	HongKong	3
IL	Israel	4
IN	India	3
CO	COUNTRY_NAME	REGION_ID
IT	Italy	1
JP	Japan	3
KW	Kuwait	4
MX	Mexico	2
NG	Nigeria	4
NL	Netherlands	1
SG	Singapore	3
UK	United Kingdom	1
US	United States of America	2
ZM	Zambia	4
ZW	Zimbabwe	4

25 rows selected.

**Tabla LOCATIONS**

```
DESCRIBE locations;
```

Name	Null?	Type
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY	NOT NULL	VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

```
SELECT * FROM locations;
```

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	CO
1000	1297 Via Cola di Rie	00989	Roma		IT
1100	93091 Calle della Testa	10934	Venice		IT
1200	2017 Shinjuku-ku	1689	Tokyo	Tokyo Prefecture	JP
1300	9450 Kamiya-cho	6823	Hiroshima		JP
1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
1500	2011 Interiors Blvd	99236	South San Francisco	California	US
1600	2007 Zagora St	50090	South Brunswick	New Jersey	US
1700	2004 Charade Rd	98199	Seattle	Washington	US
1800	147 Spadina Ave	M5V 2L7	Toronto	Ontario	CA
1900	6092 Boxwood St	YSW 9T2	Whitehorse	Yukon	CA
2000	40-5-12 Laogianggen	190518	Beijing		CN
2100	1298 Vileparle (E)	490231	Bombay	Maharashtra	IN
LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	CO
2400	8204 Arthur St		London		UK
2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK
2600	9702 Chester Road	09629850293	Stretford	Manchester	UK
2700	Schwanthalerstr 7031	80925	Munich	Bavaria	DE
2800	Rua Frei Caneca 1360	01307-002	Sao Paulo	Sao Paulo	BR
2900	20 Rue des Corps-Saints	1730	Geneva	Geneve	CH
3000	Murtenstrasse 921	3095	Bern	BE	CH
3100	Pieter Breughelstraat 837	3029SK	Utrecht	Utrecht	NL
3200	Mariano Escobedo 9991	11932	Mexico City	Distrito Federal,	MX

23 rows selected.

## Tabla DEPARTMENTS

DESCRIBE departments

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

SELECT \* FROM departments;

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
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
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
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

27 rows selected.

## Tabla JOBS

DESCRIBE jobs

Name	Null?	Type
JOB_ID	NOT NULL	VARCHAR2(10)
JOB_TITLE	NOT NULL	VARCHAR2(35)
MIN_SALARY		NUMBER(6)
MAX_SALARY		NUMBER(6)

SELECT \* FROM jobs;

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
AD_PRES	President	20000	40000
AD_VP	Administration Vice President	15000	30000
AD_ASST	Administration Assistant	3000	6000
FI_MGR	Finance Manager	8200	16000
FI_ACCOUNT	Accountant	4200	9000
AC_MGR	Accounting Manager	8200	16000
AC_ACCOUNT	Public Accountant	4200	9000
SA_MAN	Sales Manager	10000	20000
SA_REP	Sales Representative	6000	12000
PU_MAN	Purchasing Manager	8000	15000
PU_CLERK	Purchasing Clerk	2500	5500
ST_MAN	Stock Manager	5500	8500
ST_CLERK	Stock Clerk	2000	5000
SH_CLERK	Shipping Clerk	2500	5500
JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_PROG	Programmer	4000	10000
MK_MAN	Marketing Manager	9000	15000
MK_REP	Marketing Representative	4000	9000
HR_REP	Human Resources Representative	4000	9000
PR_REP	Public Relations Representative	4500	10500

19 rows selected.

## Tabla **EMPLOYEES**

DESCRIBE employees

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

Oracle Internal & OAI Use Only

## Tabla EMPLOYEES

Las cabeceras de las columnas COMMISSION\_PCT, MANAGER\_ID y DEPARTMENT\_ID están definidas en COMM, MGRID y DEPTID en la siguiente captura de pantalla para ajustar los valores de la tabla en la página.

```
SELECT * FROM employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000			90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000		100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000		100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000		102	60
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000		103	60
105	David	Austin	DAUSTIN	590.423.4569	25-JUN-97	IT_PROG	4800		103	60
106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-98	IT_PROG	4800		103	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200		103	60
108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-94	FI_MGR	12000		101	100
109	Daniel	Faviet	DFAVET	515.124.4169	16-AUG-94	FI_ACCOUNT	9000		108	100
110	John	Chen	JCHEN	515.124.4269	28-SEP-97	FI_ACCOUNT	8200		108	100
111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-97	FI_ACCOUNT	7700		108	100
112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-98	FI_ACCOUNT	7800		108	100
113	Luis	Popp	LPOPP	515.124.4567	07-DEC-99	FI_ACCOUNT	6900		108	100
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
114	Den	Raphaely	DRAPHEAL	515.127.4561	07-DEC-94	PU_MAN	11000		100	30
115	Alexander	Khoo	AKHOO	515.127.4562	18-MAY-95	PU_CLERK	3100		114	30
116	Shelli	Baida	SBAIDA	515.127.4563	24-DEC-97	PU_CLERK	2900		114	30
117	Sigal	Tobias	STOBIAS	515.127.4564	24-JUL-97	PU_CLERK	2800		114	30
118	Guy	Himuro	GHIMURO	515.127.4565	15-NOV-98	PU_CLERK	2600		114	30
119	Karen	Colmenares	KCOLMENA	515.127.4566	10-AUG-99	PU_CLERK	2500		114	30
120	Matthew	Weiss	MWEISS	650.123.1234	18-JUL-96	ST_MAN	8000		100	50
121	Adam	Fripp	AFRIPP	650.123.2234	10-APR-97	ST_MAN	8200		100	50
122	Payam	Kaufling	PKAUFLIN	650.123.3234	01-MAY-95	ST_MAN	7900		100	50
123	Shanta	Vollman	SVOLLMAN	650.123.4234	10-OCT-97	ST_MAN	6500		100	50
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800		100	50
125	Julia	Nayer	JNAYER	650.124.1214	16-JUL-97	ST_CLERK	3200		120	50
126	Irene	Mikkilineni	IMIKKILI	650.124.1224	28-SEP-98	ST_CLERK	2700		120	50
127	James	Landry	JLANDRY	650.124.1334	14-JAN-99	ST_CLERK	2400		120	50

**Tabla EMPLOYEES (continuación)**

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
128	Steven	Markle	SMARKLE	650.124.1434	08-MAR-00	ST_CLERK	2200		120	50
129	Laura	Bissot	LBISSOT	650.124.5234	20-AUG-97	ST_CLERK	3300		121	50
130	Mozhe	Atkinson	MATKINSO	650.124.6234	30-OCT-97	ST_CLERK	2800		121	50
131	James	Marlow	JAMRLOW	650.124.7234	16-FEB-97	ST_CLERK	2500		121	50
132	TJ	Olson	TJOLSON	650.124.8234	10-APR-99	ST_CLERK	2100		121	50
133	Jason	Mallin	JMALLIN	650.127.1934	14-JUN-96	ST_CLERK	3300		122	50
134	Michael	Rogers	MROGERS	650.127.1834	26-AUG-98	ST_CLERK	2900		122	50
135	Ki	Gee	KGEE	650.127.1734	12-DEC-99	ST_CLERK	2400		122	50
136	Hazel	Philtanker	HPHILTAN	650.127.1634	06-FEB-00	ST_CLERK	2200		122	50
137	Renske	Ladwig	RLADWIG	650.121.1234	14-JUL-95	ST_CLERK	3600		123	50
138	Stephen	Stiles	SSTILES	650.121.2034	26-OCT-97	ST_CLERK	3200		123	50
139	John	Seo	JSEO	650.121.2019	12-FEB-98	ST_CLERK	2700		123	50
140	Joshua	Patel	JPATEL	650.121.1834	06-APR-98	ST_CLERK	2500		123	50
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3500		124	50
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
142	Curtis	Davies	CDAMES	650.121.2994	29-JAN-97	ST_CLERK	3100		124	50
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600		124	50
144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500		124	50
145	John	Russell	JRUSSEL	011.44.1344.429268	01-OCT-96	SA_MAN	14000	.4	100	80
146	Karen	Partners	KPARTNER	011.44.1344.467268	05-JAN-97	SA_MAN	13500	.3	100	80
147	Alberto	Erazuriz	AERRAZUR	011.44.1344.429278	10-MAR-97	SA_MAN	12000	.3	100	80
148	Gerald	Cambrault	GCAMBRAU	011.44.1344.619268	15-OCT-99	SA_MAN	11000	.3	100	80
149	Beni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	10500	.2	100	80
150	Peter	Tucker	PTUCKER	011.44.1344.129268	30-JAN-97	SA_REP	10000	.3	145	80
151	David	Bernstein	DBERNSTE	011.44.1344.345268	24-MAR-97	SA_REP	9500	.25	145	80
152	Peter	Hall	PHALL	011.44.1344.478968	20-AUG-97	SA_REP	9000	.25	145	80
153	Christopher	Olsen	COLSEN	011.44.1344.498718	30-MAR-98	SA_REP	8000	.2	145	80
154	Nanette	Cambrault	NCAMBRAU	011.44.1344.987668	09-DEC-98	SA_REP	7500	.2	145	80
155	Oliver	Tuvault	OTUVAULT	011.44.1344.486508	23-NOV-99	SA_REP	7000	.15	145	80
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
156	Janette	King	JKING	011.44.1345.429268	30-JAN-96	SA_REP	10000	.35	146	80
157	Patrick	Sully	PSULLY	011.44.1345.929268	04-MAR-96	SA_REP	9500	.35	146	80
158	Allan	McEwen	AMCEWEN	011.44.1345.829268	01-AUG-96	SA_REP	9000	.35	146	80
159	Lindsey	Smith	LSMITH	011.44.1345.729268	10-MAR-97	SA_REP	8000	.3	146	80
160	Louise	Doran	LDORAN	011.44.1345.629268	15-DEC-97	SA_REP	7500	.3	146	80
161	Sarath	Sewall	SSEWALL	011.44.1345.529268	03-NOV-98	SA_REP	7000	.25	146	80
162	Clara	Vishney	CVISHNEY	011.44.1346.129268	11-NOV-97	SA_REP	10500	.25	147	80
163	Danielle	Greene	DGREENE	011.44.1346.229268	19-MAR-99	SA_REP	9500	.15	147	80
164	Mattea	Marvins	MMARVINS	011.44.1346.329268	24-JAN-00	SA_REP	7200	.1	147	80
165	David	Lee	DLEE	011.44.1346.529268	23-FEB-00	SA_REP	6800	.1	147	80
166	Sundar	Ande	SANDE	011.44.1346.629268	24-MAR-00	SA_REP	6400	.1	147	80
167	Amit	Banda	ABANDA	011.44.1346.729268	21-APR-00	SA_REP	6200	.1	147	80
168	Lisa	Ozer	LOZER	011.44.1343.929268	11-MAR-97	SA_REP	11500	.25	148	80
169	Harrison	Bloom	HBLOOM	011.44.1343.829268	23-MAR-98	SA_REP	10000	.2	148	80

**Tabla EMPLOYEES (continuación)**

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
170	Taylor	Fox	TFOX	011.44.1343.729268	24-JAN-98	SA_REP	9600	.2	148	80
171	William	Smith	WSMITH	011.44.1343.629268	23-FEB-99	SA_REP	7400	.15	148	80
172	Elizabeth	Bates	EBATES	011.44.1343.529268	24-MAR-99	SA_REP	7300	.15	148	80
173	Sundita	Kumar	SKUMAR	011.44.1343.329268	21-APR-00	SA_REP	6100	.1	148	80
174	Elen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	11000	.3	149	80
175	Alyssa	Hutton	AHUTTON	011.44.1644.429266	19-MAR-97	SA_REP	8800	.25	149	80
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	8600	.2	149	80
177	Jack	Livingston	JLIVINGS	011.44.1644.429264	23-APR-98	SA_REP	8400	.2	149	80
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	.15	149	
179	Charles	Johnson	CJOHNSON	011.44.1644.429262	04-JAN-00	SA_REP	6200	.1	149	80
180	Winston	Taylor	WTAYLOR	650.507.9876	24-JAN-98	SH_CLERK	3200		120	50
181	Jean	Fleaur	JFLEAUR	650.507.9877	23-FEB-98	SH_CLERK	3100		120	50
182	Martha	Sullivan	MSULLIVA	650.507.9878	21-JUN-99	SH_CLERK	2500		120	50
183	Girard	Geoni	GGEONI	650.507.9879	03-FEB-00	SH_CLERK	2800		120	50
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
184	Nandita	Sarchand	NSARCHAN	650.509.1876	27-JAN-96	SH_CLERK	4200		121	50
185	Alexis	Bull	ABULL	650.509.2876	20-FEB-97	SH_CLERK	4100		121	50
186	Julia	Dellinger	JDELLING	650.509.3876	24-JUN-98	SH_CLERK	3400		121	50
187	Anthony	Cabrio	ACABRIO	650.509.4876	07-FEB-99	SH_CLERK	3000		121	50
188	Kelly	Chung	KCHUNG	650.505.1876	14-JUN-97	SH_CLERK	3800		122	50
189	Jennifer	Dilly	JDILLY	650.505.2876	13-AUG-97	SH_CLERK	3600		122	50
190	Timothy	Gates	TGATES	650.505.3876	11-JUL-98	SH_CLERK	2900		122	50
191	Randall	Perkins	RPERKINS	650.505.4876	19-DEC-99	SH_CLERK	2500		122	50
192	Sarah	Bell	SBELL	650.501.1876	04-FEB-96	SH_CLERK	4000		123	50
193	Britney	Everett	BEVERETT	650.501.2876	03-MAR-97	SH_CLERK	3900		123	50
194	Samuel	McCain	SMCCAIN	650.501.3876	01-JUL-98	SH_CLERK	3200		123	50
195	Vance	Jones	VJONES	650.501.4876	17-MAR-99	SH_CLERK	2800		123	50
196	Alana	Walsh	AWALSH	650.507.9811	24-APR-98	SH_CLERK	3100		124	50
197	Kevin	Feeney	KFEENEY	650.507.9822	23-MAY-98	SH_CLERK	3000		124	50
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
198	Donald	OConnell	DOCONNEL	650.507.9833	21-JUN-99	SH_CLERK	2600		124	50
199	Douglas	Grant	DGRANT	650.507.9844	13-JAN-00	SH_CLERK	2600		124	50
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400		101	10
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000		100	20
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000		201	20
203	Susan	Mavris	SMAVRIS	515.123.7777	07-JUN-94	HR_REP	6500		101	40
204	Hermann	Baer	HBAER	515.123.8888	07-JUN-94	PR_REP	10000		101	70
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000		101	110
206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACCOUNT	8300		205	110

107 rows selected.



**Tabla JOB\_HISTORY**

```
DESCRIBE job_history
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
DEPARTMENT_ID		NUMBER(4)

```
SELECT * FROM job_history;
```

EMPLOYEE_ID	START_DAT	END_DATE	JOB_ID	deptid
102	13-JAN-93	24-JUL-98	IT_PROG	60
101	21-SEP-89	27-OCT-93	AC_ACCOUNT	110
101	28-OCT-93	15-MAR-97	AC_MGR	110
201	17-FEB-96	19-DEC-99	MK_REP	20
114	24-MAR-98	31-DEC-99	ST_CLERK	50
122	01-JAN-99	31-DEC-99	ST_CLERK	50
200	17-SEP-87	17-JUN-93	AD_ASST	90
176	24-MAR-98	31-DEC-98	SA_REP	80
176	01-JAN-99	31-DEC-99	SA_MAN	80
200	01-JUL-94	31-DEC-98	AC_ACCOUNT	90

10 rows selected.

Oracle Internal & OAI Use Only

# **Estudios para Implementación de Disparadores**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

# Objetivos

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

- **Mejorar la seguridad de la base de datos con disparadores**
- **Auditar los cambios de datos con disparadores DML**
- **Forzar la integridad de los datos con disparadores DML**
- **Mantener la integridad referencial con disparadores**
- **Utilizar disparadores para replicar datos entre tablas**
- **Utilizar disparadores para automatizar el cálculo de datos derivados**
- **Proporcionar funciones de registro de eventos con disparadores**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Objetivos

En esta lección aprenderá a desarrollar disparadores de base de datos para mejorar las funciones que, de lo contrario, no podría implementar el servidor de Oracle. En algunos casos, puede que baste con evitar el uso de disparadores y aceptar la funcionalidad proporcionada por el servidor de Oracle.

En esta lección se abordan los siguientes supuestos de aplicación de negocio:

- Seguridad
- Auditoría
- Integridad de los datos
- Integridad referencial
- Replicación de tablas
- Cálculo automático de datos derivados
- Registro de eventos

# Control de la Seguridad en el Servidor

## Uso de la Seguridad de la Base de Datos con la Sentencia GRANT

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON    employees
TO    clerk;                -- database role
GRANT clerk TO scott;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Control de la Seguridad en el Servidor

Desarrolle esquemas y roles en el servidor de Oracle para controlar la seguridad de las operaciones de datos en tablas de acuerdo con la identidad del usuario.

- Basar los privilegios en el nombre de usuario proporcionado cuando el usuario se conecta a la base de datos.
- Determinar el acceso a tablas, vistas, sinónimos y secuencias.
- Determinar los privilegios de consulta, manipulación de datos y definición de datos.

## Control de la Seguridad con un Disparador de Base de Datos

```
CREATE OR REPLACE TRIGGER secure_emp
  BEFORE INSERT OR UPDATE OR DELETE ON employees
DECLARE
  dummy PLS_INTEGER;
BEGIN
  IF (TO_CHAR (SYSDATE, 'DY') IN ('SAT','SUN')) THEN
    RAISE_APPLICATION_ERROR(-20506,'You may only
      change data during normal business hours.');
```

END IF;

```
  SELECT COUNT(*) INTO dummy FROM holiday
  WHERE holiday_date = TRUNC (SYSDATE);
  IF dummy > 0 THEN
    RAISE_APPLICATION_ERROR(-20507,
      'You may not change data on a holiday.');
```

END IF;

```
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Control de la Seguridad con un Disparador de Base de Datos

Desarrolle disparadores para manejar requisitos de seguridad más complejos.

- Basar los privilegios en cualquier valor de base de datos, como la hora del día, el día de la semana, etc.
- Determinar sólo el acceso a las tablas.
- Determinar sólo los privilegios de manipulación de datos.

# Uso de la Utilidad del Servidor para Auditoría de Operaciones de Datos

El servidor de Oracle almacena la información de auditoría en una tabla de diccionario de datos o un archivo del sistema operativo.

```
AUDIT INSERT, UPDATE, DELETE  
ON departments  
BY ACCESS  
WHENEVER SUCCESSFUL;
```

Audit succeeded.

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Auditoría de Operaciones de Datos

Puede auditar operaciones de datos en el servidor de Oracle. Las auditorías de base de datos se utilizan para controlar y recopilar datos de actividades concretas de la base de datos. El DBA puede recopilar estadísticas como qué tablas se están actualizando, cuántas entradas/salidas se han realizado, cuántos usuarios se han conectado a la vez en horas punta, etc.

- Auditar usuarios, sentencias u objetos.
- Auditar sentencias de recuperación, manipulación y definición de datos.
- Escribir la pista de auditoría en una tabla de auditoría centralizada.
- Generar un registro de auditoría por sesión o por intento de acceso.
- Capturar intentos correctos, intentos incorrectos o ambos.
- Activar y desactivar dinámicamente.

Al ejecutar SQL mediante unidades de programa PL/SQL, puede que se generen varios registros de auditoría ya que las unidades de programa pueden hacer referencia a otros objetos de base de datos.

## Auditoría con un Disparador

```
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE
ON employees FOR EACH ROW
BEGIN
  IF (audit_emp_pkg. reason IS NULL) THEN
    RAISE_APPLICATION_ERROR (-20059, 'Specify a
      reason for operation through the procedure
      AUDIT_EMP_PKG.SET_REASON to proceed.');
```

```
  ELSE
    INSERT INTO audit_emp_table (user_name,
      timestamp, id, old_last_name, new_last_name,
      old_salary, new_salary, comments)
    VALUES (USER, SYSDATE, :OLD.employee_id,
      :OLD.last_name, :NEW.last_name, :OLD.salary,
      :NEW.salary, audit_emp_pkg.reason);
  END IF;
END;
```

```
CREATE OR REPLACE TRIGGER cleanup_audit_emp
AFTER INSERT OR UPDATE OR DELETE ON employees
BEGIN audit_emp_package.g_reason := NULL;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Auditoría de Valores de Datos

Audite los valores de datos reales con disparadores.

Puede hacer lo siguiente:

- Auditar sólo las sentencias de manipulación de datos.
- Escribir la pista de auditoría en una tabla de auditoría definida por el usuario.
- Generar un registro de auditoría para la sentencia y otro para cada fila.
- Capturar sólo los intentos correctos.
- Activar y desactivar dinámicamente.

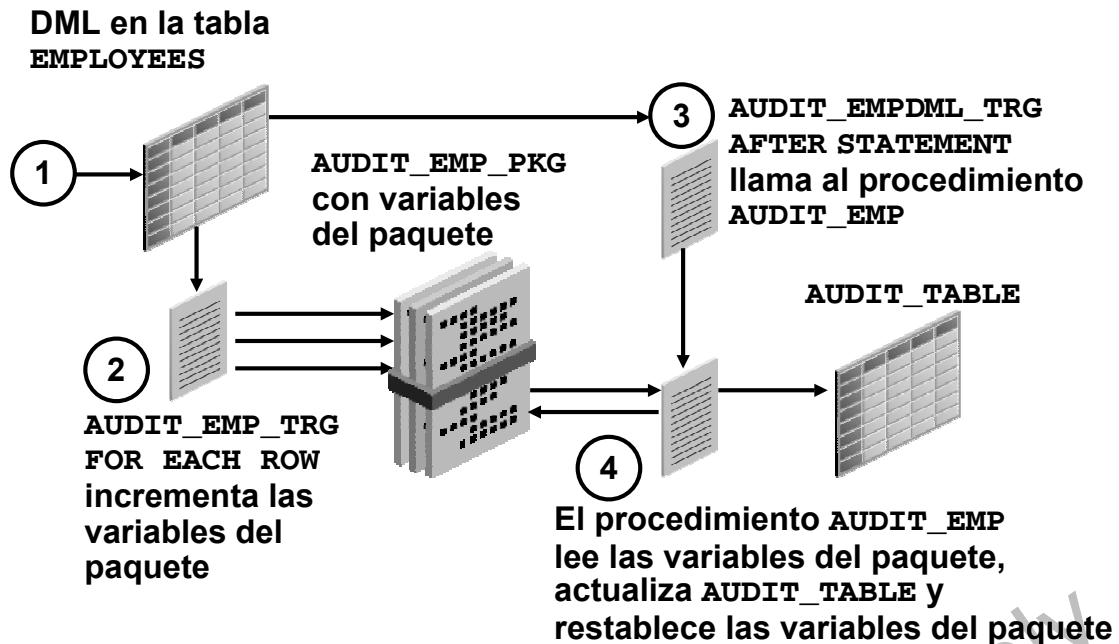
Con el servidor de Oracle, puede realizar una auditoría de la base de datos. La auditoría de la base de datos no puede registrar los cambios realizados en valores de columnas concretas.

Si se necesita realizar un seguimiento de los cambios realizados en las columnas de la tabla y almacenar los valores de las columnas para cada cambio, utilice la auditoría de la aplicación.

La auditoría de la aplicación se puede realizar mediante procedimientos almacenados o disparadores de base de datos, tal como se muestra en el ejemplo de la transparencia.



## Auditoría de Disparadores con Construcciones de Paquetes



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Auditoría de Disparadores con Construcciones de Paquetes

En las siguientes páginas se abordan los subprogramas PL/SQL e incluyen ejemplos de la interacción de disparadores, procedimientos de paquetes, funciones y variables globales.

Secuencia de eventos:

1. Ejecute un comando INSERT, UPDATE o DELETE que pueda manipular una o más filas.
2. AUDIT\_EMP\_TRG (disparador AFTER ROW) llama al procedimiento del paquete para incrementar las variables globales en el paquete VAR\_PACK. Al tratarse de un disparador de fila, éste se arranca una vez por cada fila actualizada.
3. Cuando la sentencia ha terminado, AUDIT\_EMP\_TAB (disparador AFTER STATEMENT) llama al procedimiento AUDIT\_EMP.
4. Este procedimiento asigna los valores de las variables globales a variables locales con las funciones del paquete, actualiza AUDIT\_TABLE y, a continuación, restablece las variables globales.

# Auditoría de Disparadores con Construcciones de Paquetes

## Disparador de sentencia AFTER:

```
CREATE OR REPLACE TRIGGER audit_empdml_trg
AFTER UPDATE OR INSERT OR DELETE on employees
BEGIN
    audit_emp;          -- write the audit data
END audit_emp_tab;
/
```

## Disparador de fila AFTER:

```
CREATE OR REPLACE TRIGGER audit_emp_trg
AFTER UPDATE OR INSERT OR DELETE ON EMPLOYEES
FOR EACH ROW
-- Call Audit package to maintain counts
CALL audit_emp_pkg.set(INSERTING, UPDATING, DELETING);
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Auditoría de Disparadores con Construcciones de Paquetes (continuación)

El disparador AUDIT\_EMP\_TRIG es un disparador de fila que se arranca cada vez que se manipula una fila. Este disparador llama a los procedimientos del paquete en función del tipo de operación DML realizada. Por ejemplo, si la operación DML actualiza el salario de un empleado, el disparador llama al procedimiento SET\_G\_UP\_SAL. Este procedimiento del paquete, a su vez, llama a la función G\_UP\_SAL. Esta función incrementa la variable del paquete GV\_UP\_SAL que realiza un recuento del número de filas que se cambia como consecuencia de la actualización del salario.

El disparador AUDIT\_EMP\_TAB se activa una vez terminada la sentencia. Este disparador llama al procedimiento AUDIT\_EMP, que se explica en las siguientes páginas. El procedimiento AUDIT\_EMP actualiza la tabla AUDIT\_TABLE. Se crea una entrada en la tabla AUDIT\_TABLE con información como el usuario que ha realizado la operación DML, la tabla en la que se ha realizado la operación DML y el número total de manipulaciones de datos realizadas hasta el momento en la tabla (indicado mediante el valor de la columna correspondiente en la tabla AUDIT\_TABLE). Por último, el procedimiento AUDIT\_EMP restablece las variables del paquete en 0.

## Paquete AUDIT\_PKG

```
CREATE OR REPLACE PACKAGE audit_emp_pkg IS
    delcnt PLS_INTEGER := 0;
    inscnt PLS_INTEGER := 0;
    updcnt PLS_INTEGER := 0;
    PROCEDURE init;
    PROCEDURE set(i BOOLEAN,u BOOLEAN,d BOOLEAN);
END audit_emp_pkg;
/
CREATE OR REPLACE PACKAGE BODY audit_emp_pkg IS
    PROCEDURE init IS
    BEGIN
        inscnt := 0; updcnt := 0; delcnt := 0;
    END;
    PROCEDURE set(i BOOLEAN,u BOOLEAN,d BOOLEAN) IS
    BEGIN
        IF i THEN inscnt := inscnt + 1;
        ELIF d THEN delcnt := delcnt + 1;
        ELSE upd := updcnt + 1;
        END IF;
    END;
END audit_emp_pkg;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Paquete AUDIT\_PKG

El paquete AUDIT\_PKG declara las variables del paquete públicas (inscnt, updcnt, delcnt) que se utilizan para realizar un seguimiento del número de operaciones INSERT, UPDATE y DELETE realizadas. En el ejemplo de código, se declaran públicamente para una mayor simplicidad. Sin embargo, puede que prefiera declararlas como variables privadas para evitar que se modifiquen directamente. Si las variables se declaran de forma privada, en el cuerpo del paquete, debe proporcionar subprogramas públicos adicionales que devuelvan sus valores al usuario del paquete.

El procedimiento init se utiliza para inicializar las variables del paquete públicas en cero.

El procedimiento set acepta tres argumentos BOOLEAN i, u y d para una operación INSERT, UPDATE o DELETE, respectivamente. El valor del parámetro adecuado se define en TRUE cuando el disparador que llama al procedimiento set se activa durante una de las operaciones DML. Una variable de paquete se incrementa en 1, en función de qué valor de argumento es TRUE al llamar al procedimiento set.

**Nota:** Un disparador DML se puede arrancar una vez para cada DML de cada fila. Por lo tanto, sólo una de las tres variables transferidas al procedimiento set puede ser TRUE en un momento determinado. Los otros dos argumentos se definirán en el valor FALSE.

## Tabla AUDIT\_TABLE y Procedimiento AUDIT\_EMP

```
CREATE TABLE audit_table (  
  USER_NAME  VARCHAR2(30),  
  TABLE_NAME VARCHAR2(30),  
  INS        NUMBER,  
  UPD        NUMBER,  
  DEL        NUMBER)  
/  
CREATE OR REPLACE PROCEDURE audit_emp IS  
BEGIN  
  IF delcnt + inscnt + updcnt <> 0 THEN  
    UPDATE audit_table  
      SET del = del + audit_emp_pkg.delcnt,  
          ins = ins + audit_emp_pkg.inscnt,  
          upd = upd + audit_emp_pkg.updcnt  
    WHERE user_name = USER  
      AND table_name = 'EMPLOYEES';  
    audit_emp_pkg.init;  
  END IF;  
END audit_emp;  
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Tabla AUDIT\_TABLE y Procedimiento AUDIT\_EMP

El procedimiento AUDIT\_EMP actualiza la tabla AUDIT\_TABLE y llama a las funciones del paquete AUDIT\_EMP\_PKG que restablecen las variables del paquete, preparadas para la siguiente sentencia DML.

## Forzado de Integridad de Datos en el Servidor

```
ALTER TABLE employees ADD  
CONSTRAINT ck_salary CHECK (salary >= 500);
```

Table altered.

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Forzado de Integridad de Datos en el Servidor

Puede forzar la integridad de los datos en el servidor de Oracle y desarrollar disparadores para manejar reglas de integridad de datos más complejas.

Las reglas de integridad de datos estándar son clave ajena, primaria, única y no nula.

Utilice estas reglas para:

- Proporcionar valores por defecto constantes
- Forzar restricciones estáticas
- Activar y desactivar dinámicamente

#### Ejemplo

El ejemplo de código de la transparencia garantiza que el salario sea al menos \$500.

## Protección de la Integridad de los Datos con un Disparador

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE UPDATE OF salary ON employees
  FOR EACH ROW
  WHEN (NEW.salary < OLD.salary)
BEGIN
  RAISE_APPLICATION_ERROR (-20508,
    'Do not decrease salary. ');
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Protección de la Integridad de los Datos con un Disparador

Proteja la integridad de los datos con un disparador y fuerce comprobaciones de integridad de datos no estándar.

- Proporcionar valores por defecto variables.
- Forzar restricciones dinámicas.
- Activar y desactivar dinámicamente.
- Incorporar restricciones declarativas en la definición de una tabla para proteger la integridad de los datos.

#### Ejemplo

El ejemplo de código de la transparencia garantiza que el salario nunca se reduzca.

## Forzado de la Integridad Referencial en el Servidor

```
ALTER TABLE employees
  ADD CONSTRAINT emp_deptno_fk
  FOREIGN KEY (department_id)
    REFERENCES departments(department_id)
  ON DELETE CASCADE;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Forzado de la Integridad Referencial en el Servidor

Incorpore restricciones de integridad referencial a la definición de una tabla para evitar la inconsistencia de datos y forzar la integridad referencial en el servidor.

- Restringir las actualizaciones y supresiones.
- Realizar supresiones en cascada.
- Activar y desactivar dinámicamente.

#### Ejemplo

Al eliminar un departamento de la tabla principal DEPARTMENTS, puede realizar supresiones en cascada hasta las filas correspondientes de la tabla secundaria EMPLOYEES.

## Protección de la Integridad Referencial con un Disparador

```
CREATE OR REPLACE TRIGGER cascade_updates
AFTER UPDATE OF department_id ON departments
FOR EACH ROW
BEGIN
    UPDATE employees
    SET employees.department_id=:NEW.department_id
    WHERE employees.department_id=:OLD.department_id;
    UPDATE job_history
    SET department_id=:NEW.department_id
    WHERE department_id=:OLD.department_id;
END;
/
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Protección de la Integridad Referencial con un Disparador

Las restricciones declarativas no soportan las siguientes reglas de integridad referencial:

- Realizar actualizaciones en cascada.
- Definir en NULL para actualizaciones y supresiones.
- Definir en un valor por defecto al realizar actualizaciones y supresiones.
- Forzar la integridad referencial en un sistema distribuido.
- Activar y desactivar dinámicamente.

Puede desarrollar disparadores para implementar estas reglas de integridad.

#### Ejemplo

Puede forzar la integridad referencial con un disparador. Cuando el valor DEPARTMENT\_ID cambie en la tabla principal DEPARTMENTS, realice actualizaciones en cascada hasta las filas correspondientes de la tabla secundaria EMPLOYEES.

Para una solución de integridad referencial completa con disparadores, no es suficiente un único disparador.



# Replicación de Tablas en el Servidor

```
CREATE MATERIALIZED VIEW emp_copy  
NEXT sysdate + 7  
AS SELECT * FROM employees@ny;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Creación de Vistas Materializadas

Las vistas materializadas permiten mantener copias de datos remotos en el nodo local con fines de replicación. Puede seleccionar datos en una vista materializada de la misma forma que en una vista o tabla de base de datos normal. Una vista materializada es un objeto de base de datos que contiene los resultados de una consulta o una copia de una base de datos relacionada con la consulta. La cláusula FROM de la consulta de una vista materializada puede asignar nombres a tablas, vistas y otras vistas materializadas.

Cuando se utiliza una vista materializada, el servidor de Oracle realiza la replicación implícitamente. Esto funciona mejor que utilizar disparadores PL/SQL definidos por el usuario para la replicación. Las vistas materializadas:

- Copian datos de tablas locales y remotas de forma asíncrona, en intervalos definidos por el usuario.
- Pueden estar basadas en varias tablas maestras.
- Son de sólo lectura por defecto, excepto si se utiliza la función Oracle Advanced Replication.
- Mejoran el rendimiento de la manipulación de datos en la tabla maestra.

Asimismo, puede replicar tablas mediante disparadores.

El ejemplo de la transparencia crea una copia de la tabla EMPLOYEES remota de Nueva York.

La cláusula NEXT especifica una expresión de fecha y hora para el intervalo entre refrescamientos automáticos.

## Replicación de Tablas con un Disparador

```
CREATE OR REPLACE TRIGGER emp_replica
BEFORE INSERT OR UPDATE ON employees FOR EACH ROW
BEGIN /* Proceed if user initiates data operation,
      NOT through the cascading trigger.*/
  IF INSERTING THEN
    IF :NEW.flag IS NULL THEN
      INSERT INTO employees@sf
      VALUES(:new.employee_id,...,'B');
      :NEW.flag := 'A';
    END IF;
  ELSE /* Updating. */
    IF :NEW.flag = :OLD.flag THEN
      UPDATE employees@sf
      SET ename=:NEW.last_name,...,flag=:NEW.flag
      WHERE employee_id = :NEW.employee_id;
    END IF;
    IF :OLD.flag = 'A' THEN :NEW.flag := 'B';
    ELSE :NEW.flag := 'A';
  END IF;
  END IF;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Replicación de Tablas con un Disparador

Puede replicar una tabla con un disparador. Con la replicación de una tabla, puede:

- Copiar tablas de forma síncrona, en tiempo real
- Basar las réplicas en una única tabla maestra
- Leer las réplicas, así como escribir en ellas

**Nota:** El uso excesivo de disparadores puede afectar al rendimiento de la manipulación de datos en la tabla maestra, especialmente si la red falla.

#### Ejemplo

En Nueva York, puede replicar la tabla EMPLOYEES local para San Francisco.

## Cálculo de Datos Derivados en el Servidor

```
UPDATE departments
SET total_sal=(SELECT SUM(salary)
                FROM employees
                WHERE employees.department_id =
                    departments.department_id);
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Cálculo de Datos Derivados en el Servidor

Con el servidor, puede planificar trabajos por lotes o utilizar el planificador de base de datos para los siguientes supuestos:

- Calcular valores de columna derivados de forma asíncrona, en intervalos definidos por el usuario.
- Almacenar valores derivados sólo en las tablas de base de datos.
- Modificar datos en una primera transferencia a la base de datos y calcular datos derivados en una segunda transferencia.

Asimismo, puede utilizar disparadores para seguir realizando cálculos de los datos derivados.

#### Ejemplo

Mantenga el salario total de cada departamento en una columna TOTAL\_SALARY especial de la tabla DEPARTMENTS.

## Cálculo de Valores Derivados con un Disparador

```
CREATE PROCEDURE increment_salary
(id NUMBER, new_sal NUMBER) IS
BEGIN
    UPDATE departments
    SET    total_sal = NVL (total_sal, 0)+ new_sal
    WHERE department_id = id;
END increment_salary;
```

```
CREATE OR REPLACE TRIGGER compute_salary
AFTER INSERT OR UPDATE OF salary OR DELETE
ON employees FOR EACH ROW
BEGIN
    IF DELETING THEN      increment_salary(
        :OLD.department_id, (-1* :OLD.salary));
    ELSIF UPDATING THEN   increment_salary(
        :NEW.department_id, (:NEW.salary-:OLD.salary));
    ELSE                  increment_salary(
        :NEW.department_id, :NEW.salary); --INSERT
    END IF;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Cálculo de Valores de Datos Derivados con un Disparador

Si utiliza un disparador, puede realizar las siguientes tareas:

- Calcular las columnas derivadas de forma síncrona, en tiempo real.
- Almacenar los valores derivados en tablas de base de datos o en variables globales de paquete.
- Modificar datos y calcular datos derivados en una única transferencia a la base de datos.

#### Ejemplo

Mantenga el salario total actual de cada departamento en una columna TOTAL\_SALARY especial de la tabla DEPARTMENTS.

## Registro de Eventos con un Disparador

```
CREATE OR REPLACE TRIGGER notify_reorder_rep
BEFORE UPDATE OF quantity_on_hand, reorder_point
ON inventories FOR EACH ROW
DECLARE
  dsc product_descriptions.product_description%TYPE;
  msg_text VARCHAR2(2000);
BEGIN
  IF :NEW.quantity_on_hand <=
    :NEW.reorder_point THEN
    SELECT product_description INTO dsc
    FROM product_descriptions
    WHERE product_id = :NEW.product_id;
    msg_text := 'ALERT: INVENTORY LOW ORDER:' ||
      'Yours,' || CHR(10) || user || ' .' || CHR(10);
  ELSIF :OLD.quantity_on_hand >=
    :NEW.quantity_on_hand THEN
    msg_text := 'Product #' || ... CHR(10);
  END IF;
  UTL_MAIL.SEND('inv@oracle.com', 'ord@oracle.com',
    message=>msg_text, subject=>'Inventory Notice');
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Registro de Eventos con un Disparador

En el servidor, puede registrar eventos consultando datos o realizando operaciones de forma manual. De esta forma, se envía un mensaje de correo electrónico cuando el inventario de un determinado producto está por debajo del límite aceptable. Este disparador utiliza el paquete UTL\_MAIL proporcionado por Oracle para enviar el mensaje de correo electrónico.

#### Registro de Eventos en el Servidor

1. Consulte datos explícitamente para determinar si es necesaria una operación.
2. Realice la operación, como enviar un mensaje.

#### Uso de Disparadores para Registrar Eventos

1. Realice operaciones implícitamente, como desactivar una nota electrónica automática.
2. Modifique datos y realice la operación dependiente en un único paso.
3. Registre eventos automáticamente al cambiar los datos.

## Registro de Eventos con un Disparador (continuación)

### Registro de Eventos de Forma Transparente

En el código del disparador:

- CHR(10) es un retorno de carro.
- Reorder\_point no tiene el valor NULL.
- Otra transacción puede recibir y leer el mensaje en el canal.

### Ejemplo

```
CREATE OR REPLACE TRIGGER notify_reorder_rep
BEFORE UPDATE OF amount_in_stock, reorder_point
ON inventory FOR EACH ROW
DECLARE
    dsc product.descrip%TYPE;
    msg_text VARCHAR2(2000);
BEGIN
    IF :NEW.amount_in_stock <= :NEW.reorder_point THEN
        SELECT descrip INTO dsc
        FROM PRODUCT WHERE prodid = :NEW.product_id;
        msg_text := 'ALERT: INVENTORY LOW ORDER:' || CHR(10) ||
        'It has come to my personal attention that, due to recent'
        || CHR(10) || 'transactions, our inventory for product # ' ||
        TO_CHAR(:NEW.product_id) || '-- ' || dsc ||
        ' -- has fallen below acceptable levels.' || CHR(10) ||
        'Yours,' || CHR(10) || user || '.' || CHR(10) || CHR(10);
    ELSIF :OLD.amount_in_stock >= :NEW.amount_in_stock THEN
        msg_text := 'Product #' || TO_CHAR(:NEW.product_id)
        || ' ordered. ' || CHR(10) || CHR(10);
    END IF;
    UTL_MAIL.SEND('inv@oracle.com','ord@oracle.com',
        message => msg_text, subject => 'Inventory Notice');
END;
```

## Resumen

**En esta lección, debe haber aprendido lo siguiente:**

- **Utilizar disparadores de base de datos o la funcionalidad del servidor de base de datos para:**
  - **Mejorar la seguridad de la base de datos**
  - **Auditar cambios de datos**
  - **Forzar la integridad de los datos**
  - **Mantener la integridad referencial**
  - **Replicar datos entre tablas**
  - **Automatizar el cálculo de datos derivados**
  - **Proporcionar funciones de registro de eventos**
- **Reconocer cuándo utilizar disparadores para la funcionalidad de la base de datos**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Resumen

Esta lección proporciona una comparación detallada del uso de la funcionalidad del servidor de la base de datos Oracle para implementar las operaciones de seguridad, auditoría, integridad de los datos, replicación y registro. También cubre la forma en que se pueden utilizar los disparadores de base de datos para implementar las mismas operaciones pero mejorando las funciones proporcionadas por el servidor de base de datos. En algunos casos, debe utilizar un disparador para realizar algunas actividades (como el cálculo de datos derivados), ya que el servidor de Oracle no sabe cómo implementar este tipo de regla de negocio sin algún esfuerzo de programación.

Oracle Internal & OAI Use Only



# **Revisión de PL/SQL**

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

## Estructura en Bloque para Bloques PL/SQL Anónimos

- **DECLARE** (opcional)
  - Declarar objetos PL/SQL que se deben utilizar en este bloque
- **BEGIN** (obligatorio)
  - Definir las sentencias ejecutables
- **EXCEPTION** (opcional)
  - Definir las acciones que tienen lugar si se produce un error o una excepción
- **END;** (obligatorio)

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Bloques Anónimos

Los bloques anónimos no tienen nombres. Se declaran en el momento en el que se van a ejecutar en una aplicación y se transfieren al sistema PL/SQL para su ejecución en tiempo de ejecución.

- La sección situada entre las palabras clave **DECLARE** y **BEGIN** se denomina sección de declaraciones. En la sección de declaraciones, se definen los objetos PL/SQL como variables, constantes, cursores y excepciones definidas por el usuario a los que desea hacer referencia dentro del bloque. La palabra clave **DECLARE** es opcional si no declara ningún objeto PL/SQL.
- Las palabras clave **BEGIN** y **END** son obligatorias y delimitan el conjunto de acciones que se debe realizar. Esta sección se conoce como la sección ejecutable del bloque.
- La sección situada entre **EXCEPTION** y **END** se conoce como sección de excepciones. La sección de excepciones comprende las condiciones de error. En ella, puede definir las acciones que se deben realizar si se produce la condición especificada. La sección de excepciones es opcional.

Las palabras clave **DECLARE**, **BEGIN** y **EXCEPTION** no aparecen seguidas de punto y coma, pero **END** y todas las demás sentencias PL/SQL necesitan punto y coma.

# Declaración de Variables PL/SQL

- **Sintaxis:**

```
identifier [CONSTANT] datatype [NOT NULL]
    [:= | DEFAULT expr];
```

- **Ejemplos:**

```
Declare
  v_hiredate      DATE;
  v_deptno        NUMBER(2) NOT NULL := 10;
  v_location      VARCHAR2(13) := 'Atlanta';
  c_ comm         CONSTANT NUMBER := 1400;
  v_count         BINARY_INTEGER := 0;
  v_valid         BOOLEAN NOT NULL := TRUE;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Declaración de Variables PL/SQL

Debe declarar todos los identificadores PL/SQL de la sección de declaraciones antes de hacer referencia a ellos en el bloque PL/SQL. Puede asignar un valor inicial. No es necesario asignar un valor a una variable para declararla. Si hace referencia a otras variables de una declaración, asegúrese de declararlas por separado en una sentencia anterior.

En la sintaxis,

*Identifier*      Es el nombre de la variable.

CONSTANT      Limita la variable para que no se pueda cambiar el valor; las constantes se deben inicializar.

*datatype*      Es un tipo de dato escalar, compuesto, de referencia o LOB. (En este curso se abordan únicamente los tipos de dato escalar y compuesto.)

NOT NULL      Limita la variable para que contenga obligatoriamente un valor; las variables NOT NULL se deben inicializar.

*expr*      Es cualquier expresión PL/SQL que sea un literal, otra variable o una expresión que incluya operadores y funciones.

## Declaración de Variables con el Atributo %TYPE

### Ejemplos:

```
...  
v_ename          employees.last_name%TYPE;  
v_balance        NUMBER(7,2);  
v_min_balance    v_balance%TYPE := 10;  
...
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Declaración de Variables con el Atributo %TYPE

Declare variables para almacenar el nombre de un empleado.

```
...  
v_ename          employees.last_name%TYPE;  
...
```

Declare variables para almacenar el saldo de una cuenta bancaria, así como el saldo mínimo que empieza en 10.

```
...  
v_balance        NUMBER(7,2);  
v_min_balance    v_balance%TYPE := 10;  
...
```

Las restricciones de columna NOT NULL no se aplican a las variables que se declaran con %TYPE. Por lo tanto, si declara una variable con el atributo %TYPE y una columna de base de datos definida como NOT NULL, puede asignar el valor NULL a la variable.

## Creación de un Registro PL/SQL

**Declarar variables para almacenar el nombre, el trabajo y el salario de un nuevo empleado.**

**Ejemplo:**

```
...  
    TYPE emp_record_type IS RECORD  
        (ename      VARCHAR2(25),  
         job        VARCHAR2(10),  
         sal         NUMBER(8,2));  
    emp_record      emp_record_type;  
...
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Creación de un Registro PL/SQL

Las declaraciones de campos son como las declaraciones de variables. Cada campo tiene un nombre único y un tipo de dato concreto. No existen tipos de dato predefinidos para registros PL/SQL, como ocurre con las variables escalares. Por lo tanto, debe crear primero el tipo de dato y, a continuación, declarar un identificador que utilice ese tipo de dato.

En el siguiente ejemplo se muestra cómo utilizar el atributo %TYPE para especificar un tipo de dato de campo:

```
DECLARE  
    TYPE emp_record_type IS RECORD  
        (empid  NUMBER(6) NOT NULL := 100,  
         ename   employees.last_name%TYPE,  
         job     employees.job_id%TYPE);  
    emp_record  emp_record_type;  
...
```

**Nota:** Puede agregar la restricción NOT NULL a cualquier declaración de campo y evitar así la asignación de valores nulos a dicho campo. Recuerde que los campos declarados como NOT NULL se deben inicializar.

## Atributo %ROWTYPE

### Ejemplos:

- **Declarar una variable para almacenar la misma información sobre un departamento que se almacena en la tabla DEPARTMENTS.**

```
dept_record    departments%ROWTYPE;
```

- **Declarar una variable para almacenar la misma información sobre un empleado que se almacena en la tabla EMPLOYEES.**

```
emp_record     employees%ROWTYPE;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Ejemplos

La primera declaración de la transparencia crea un registro con los mismos nombres de campo y tipos de dato de campo que una fila de la tabla DEPARTMENTS. Los campos son DEPARTMENT\_ID, DEPARTMENT\_NAME, MANAGER\_ID y LOCATION\_ID.

La segunda declaración de la transparencia crea un registro con los mismos nombres de campo y tipos de dato de campo que una fila de la tabla EMPLOYEES. Los campos son EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, EMAIL, PHONE\_NUMBER, HIRE\_DATE, JOB\_ID, SALARY, COMMISSION\_PCT, MANAGER\_ID y DEPARTMENT\_ID.

En el siguiente ejemplo, se seleccionan los valores de columna en un registro denominado item\_record.

```
DECLARE
    job_record  jobs%ROWTYPE;
    ...
BEGIN
    SELECT * INTO job_record
    FROM    jobs
    WHERE   ...
```

## Creación de una Tabla PL/SQL

```
DECLARE
  TYPE ename_table_type IS TABLE OF
    employees.last_name%TYPE
    INDEX BY BINARY_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
    INDEX BY BINARY_INTEGER;
  ename_table      ename_table_type;
  hiredate_table   hiredate_table_type;
BEGIN
  ename_table(1) := 'CAMERON';
  hiredate_table(8) := SYSDATE + 7;
  IF ename_table.EXISTS(1) THEN
    INSERT INTO ...
    ...
  END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Creación de una Tabla PL/SQL

No existen tipos de dato predefinidos para registros PL/SQL, como ocurre con las variables escalares. Por lo tanto, debe crear primero el tipo de dato y, a continuación, declarar un identificador que utilice ese tipo de dato.

#### Referencia a una Tabla PL/SQL

##### Sintaxis

`pl/sql_table_name(primary_key_value)`

En esta sintaxis, `primary_key_value` pertenece al tipo `BINARY_INTEGER`.

Haga referencia a la tercera fila de una tabla PL/SQL `ENAME_TABLE`.

`ename_table(3) ...`

El rango de magnitud de `BINARY_INTEGER` es de `-2147483647` a `2147483647`. Por lo tanto, la clave primaria puede ser negativa. La indexación no tiene que empezar por 1.

**Nota:** La sentencia `table.EXISTS(i)` devuelve el valor `TRUE` si al menos se devuelve una fila con el índice `i`. Utilice la sentencia `EXISTS` para evitar que se produzca un error en relación con un elemento de tabla que no existe.

# Sentencias **SELECT** en PL/SQL

La cláusula **INTO** es obligatoria.

Ejemplo:

```
DECLARE
  v_deptid    NUMBER(4);
  v_loc       NUMBER(4);
BEGIN
  SELECT  department_id, location_id
  INTO    v_deptno, v_loc
  FROM    departments
  WHERE   department_name = 'Sales';
  ...
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Cláusula **INTO**

La cláusula **INTO** es obligatoria y se produce entre las cláusulas **SELECT** y **FROM**. Se utiliza para especificar los nombres de las variables para contener los valores que devuelve SQL de la cláusula **SELECT**. Debe proporcionar una variable para cada elemento seleccionado y el orden de las variables debe corresponder a los elementos seleccionados.

La cláusula **INTO** se utiliza para rellenar variables PL/SQL o del host.

### Las Consultas Deben Devolver una Única Fila

Las sentencias **SELECT** de un bloque PL/SQL pertenecen a la clasificación ANSI de SQL embebido, a la que se aplica la siguiente regla:

Las consultas deben devolver una única fila. Si se devuelven más filas o no se devuelve ninguna, se genera un error.

PL/SQL se ocupa de estos errores emitiendo excepciones estándar, comprendida en la sección de excepciones del bloque con las excepciones **NO\_DATA\_FOUND** y **TOO\_MANY\_ROWS**. Debe codificar las sentencias **SELECT** para devolver una única fila.



# Inserción de Datos

**Agregar nueva información de empleado a la tabla EMPLOYEES.**

**Ejemplo:**

```
DECLARE
  v_empid employees.employee_id%TYPE;
BEGIN
  SELECT employees_seq.NEXTVAL
  INTO    v_empno
  FROM    dual;
  INSERT INTO employees(employee_id, last_name,
                        job_id, department_id)
  VALUES(v_empno, 'HARDING', 'PU_CLERK', 30);
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Inserción de Datos

- Utilice funciones SQL, como USER y SYSDATE.
- Genere valores de clave primaria con secuencias de base de datos.
- Derive valores en el bloque PL/SQL.
- Agregue valores de columna por defecto.

**Nota:** No hay posibilidades de que exista ambigüedad entre los identificadores y los nombres de columna en la sentencia INSERT. Cualquier identificador de la cláusula INSERT debe ser un nombre de columna de base de datos.

## Actualización de Datos

**Aumentar el salario de todos los empleados de la tabla `EMPLOYEES` que sean oficinistas en el departamento de compras.**

**Ejemplo:**

```
DECLARE
    v_sal_increase    employees.salary%TYPE := 2000;
BEGIN
    UPDATE employees
    SET      salary = salary + v_sal_increase
    WHERE   job_id = 'PU_CLERK';
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Actualización de Datos

Puede que exista ambigüedad en la cláusula `SET` de la sentencia `UPDATE` ya que, aunque el identificador situado a la izquierda del operador de asignación es siempre una columna de base de datos, el identificador de la derecha puede ser tanto una columna de base de datos como una variable PL/SQL.

Recuerde que la cláusula `WHERE` se utiliza para determinar qué filas se verán afectadas. Si no se modifica ninguna fila, no se produce ningún error (a diferencia que con la sentencia `SELECT` en PL/SQL).

**Nota:** Las asignaciones de variables PL/SQL siempre utilizan `:=` mientras que las asignaciones de columnas SQL siempre utilizan `=`. Recuerde que si los nombres de columna y los nombres de identificador son idénticos en la cláusula `WHERE`, el servidor de Oracle buscará primero el nombre en la base de datos.

## Supresión de Datos

**Suprimir filas que pertenecen al departamento 190 de la tabla `EMPLOYEES`.**

**Ejemplo:**

```
DECLARE
  v_deptid    employees.department_id%TYPE := 190;
BEGIN
  DELETE FROM employees
  WHERE department_id = v_deptid;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Supresión de Datos

Suprime un trabajo concreto:

```
DECLARE
  v_jobid      jobs.job_id%TYPE := 'PR_REP';
BEGIN
  DELETE FROM jobs
  WHERE job_id = v_jobid;
END;
```

## Sentencias COMMIT y ROLLBACK

- **Iniciar una transacción con el primer comando DML después de una sentencia COMMIT o ROLLBACK.**
- **Utilizar las sentencias SQL COMMIT y ROLLBACK para terminar una transacción de forma explícita.**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Control de Transacciones

Las sentencias SQL COMMIT y ROLLBACK ayudan a controlar la lógica de las transacciones, convirtiendo en permanentes algunos grupos de cambios de la base de datos y desechando otros. Como ocurre con el servidor de Oracle, las transacciones DML empiezan en el primer comando después de una sentencia COMMIT o ROLLBACK y terminan en la siguiente sentencia COMMIT o ROLLBACK correcta. Estas acciones se pueden realizar en un bloque PL/SQL o como consecuencia de los eventos del entorno del host. Una sentencia COMMIT termina la transacción actual convirtiendo en permanentes todos los cambios pendientes realizados en la base de datos.

#### Sintaxis

```
COMMIT [WORK] ;  
ROLLBACK [WORK] ;
```

En esta sintaxis, WORK es compatible con los estándares ANSI.

**Nota:** Todos los comandos de control de transacciones son válidos en PL/SQL, aunque puede que el entorno del host emita alguna restricción en relación con su uso.

También puede incluir comandos de bloqueo explícito (como LOCK TABLE y SELECT . . . FOR UPDATE) en un bloque. Estos permanecerán en vigor hasta el final de la transacción. Por otro lado, un bloque PL/SQL no implica necesariamente una transacción.

## Atributos de Cursor SQL

**Con los atributos de cursor SQL, puede probar el resultado de las sentencias SQL.**

<b>SQL%ROWCOUNT</b>	<b>Número de filas afectadas por la sentencia SQL más reciente (valor entero)</b>
<b>SQL%FOUND</b>	<b>Atributo booleano que se evalúa como <b>TRUE</b> si la sentencia SQL más reciente afecta a una o más filas</b>
<b>SQL%NOTFOUND</b>	<b>Atributo booleano que se evalúa como <b>TRUE</b> si la sentencia SQL más reciente no afecta a ninguna fila</b>
<b>SQL%ISOPEN</b>	<b>Siempre se evalúa como <b>FALSE</b> ya que PL/SQL cierra los cursores implícitos inmediatamente después de que se ejecuten</b>

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Atributos de Cursor SQL

Los atributos de cursor SQL le permiten evaluar lo que ocurrió la última vez que se utilizó el cursor implícito. Se utilizan en sentencias PL/SQL como funciones. No se pueden utilizar en sentencias SQL.

Puede utilizar los atributos SQL%ROWCOUNT, SQL%FOUND, SQL%NOTFOUND y SQL%ISOPEN en la sección de excepciones de un bloque para recopilar información sobre la ejecución de una sentencia DML. En PL/SQL, una sentencia DML que no cambia ninguna fila no se considera una condición de error, mientras que la sentencia SELECT devolverá una excepción si no encuentra ninguna fila.

## Sentencias IF, THEN y ELSIF

Para un valor concreto introducido, se puede devolver un valor calculado.

**Ejemplo:**

```
. . .  
IF v_start > 100 THEN  
    v_start := 2 * v_start;  
ELSIF v_start >= 50 THEN  
    v_start := 0,5 * v_start;  
ELSE  
    v_start := 0,1 * v_start;  
END IF;  
. . .
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Sentencias IF, THEN y ELSIF

Cuando sea posible, utilice la cláusula ELSIF en lugar de anidar sentencias IF. El código es más fácil de leer y entender, y la lógica se identifica con claridad. Si la acción de la cláusula ELSE consiste básicamente en otra sentencia IF, resulta más práctico utilizar la cláusula ELSIF. Esto hace que el código sea más claro, ya que elimina la necesidad de introducir sentencias END IF anidadas al final de cada juego de condiciones y acciones.

**Ejemplo**

```
IF condition1 THEN  
    statement1;  
ELSIF condition2 THEN  
    statement2;  
ELSIF condition3 THEN  
    statement3;  
END IF;
```

Las sentencias IF, THEN y ELSIF de la transparencia se definen en profundidad de la siguiente forma:

Para un valor concreto introducido, se puede devolver un valor calculado. Si el valor introducido es superior a 100, el valor calculado será el doble del valor introducido. Si el valor introducido está entre 50 y 100, el valor calculado será el 50% del valor inicial. Si el valor introducido es inferior a 50, el valor calculado será el 10% del valor inicial.

**Nota:** Cualquier expresión aritmética que contenga valores nulos se evalúa como nula.

## Bucle Básico

### Ejemplo:

```
DECLARE
  v_ordid    order_items.order_id%TYPE := 101;
  v_counter  NUMBER(2) := 1;
BEGIN
  LOOP
    INSERT INTO order_items(order_id,line_item_id)
    VALUES(v_ordid, v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
  END LOOP;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Bucle Básico

El ejemplo de bucle básico que se muestra en la transparencia se define de la siguiente forma:  
Inserte los primeros 10 elementos de nueva línea para el pedido número 101.

**Nota:** Un bucle básico permite la ejecución de sus sentencias al menos una vez, incluso si la condición se ha cumplido al introducir el bucle.

## Bucle FOR

**Inserte los primeros 10 elementos de nueva línea para el pedido número 101.**

**Ejemplo:**

```
DECLARE
  v_ordid      order_items.order_id%TYPE := 101;
BEGIN
  FOR i IN 1..10 LOOP
    INSERT INTO order_items(order_id,line_item_id)
      VALUES(v_ordid, i);
  END LOOP;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Bucle FOR

La transparencia muestra un bucle FOR que inserta 10 filas en la tabla `order_items`.



## Bucle WHILE

### Ejemplo:

```
ACCEPT p_price PROMPT 'Enter the price of the item: '
ACCEPT p_itemtot -
  PROMPT 'Enter the maximum total for purchase of item: '
DECLARE
...
v_qty                NUMBER(8) := 1;
v_running_total      NUMBER(7,2) := 0;
BEGIN
...
  WHILE v_running_total < &p_itemtot LOOP
    ...
    v_qty := v_qty + 1;
    v_running_total := v_qty * &p_price;
  END LOOP;
...

```

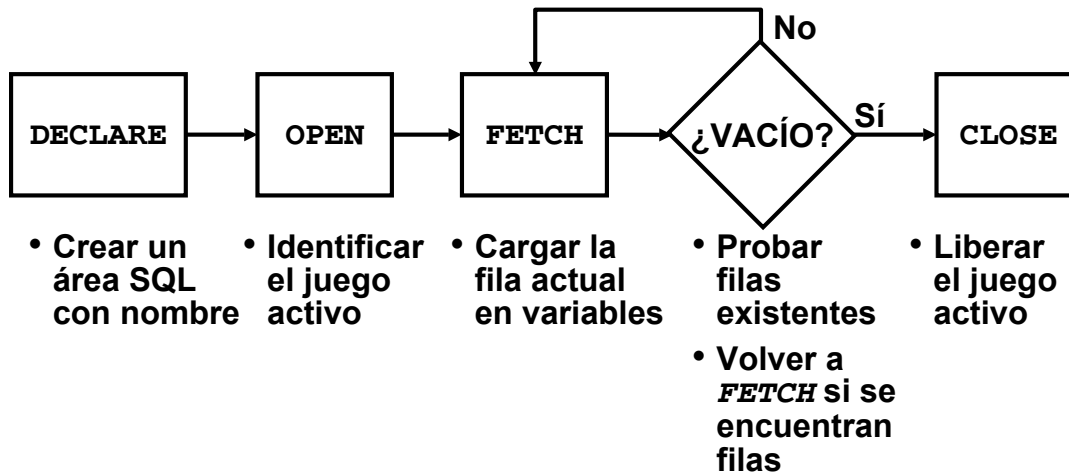
ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Bucle WHILE

En el ejemplo de la transparencia, la cantidad aumenta con cada iteración del bucle hasta que la cantidad deje de ser inferior al precio máximo permitido para el elemento.

## Control de Cursores Explícitos



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Cursores Explícitos

#### Control de Cursores Explícitos con Cuatro Comandos

1. Declare el cursor asignándole un nombre y definiendo la estructura de la consulta que desea realizar en él.
2. Abra el cursor. La sentencia **OPEN** ejecuta la consulta y enlaza cualquier variable a la que se haga referencia. Las filas identificadas por la consulta se denominan *juego activo* y pasan a estar disponibles para su recuperación.
3. Recupere los datos del cursor. La sentencia **FETCH** carga la fila actual del cursor en variables. Con cada recuperación, el puntero del cursor se mueve a la siguiente fila del juego activo. Por lo tanto, cada recuperación accede a una fila diferente devuelta por la consulta. En el diagrama de flujo de la transparencia, cada recuperación realiza pruebas en el cursor en busca de filas existentes. Si encuentra filas, carga la fila actual en variables; de lo contrario, cierra el cursor.
4. Cierre el cursor. La sentencia **CLOSE** libera el juego de filas activo. Ya puede volver a abrir el cursor para establecer un nuevo juego activo.

# Declaración del Cursor

## Ejemplo:

```
DECLARE
  CURSOR c1 IS
    SELECT employee_id, last_name
    FROM   employees;

  CURSOR c2 IS
    SELECT *
    FROM   departments
    WHERE  department_id = 10;
BEGIN
  ...
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Declaración de Cursores Explícitos

Recupere los empleados de uno en uno.

```
DECLARE
  v_empid employees.employee_id%TYPE;
  v_ename employees.last_name%TYPE;
  CURSOR c1 IS
    SELECT employee_id, last_name
    FROM   employees;
BEGIN
  ...
```

**Nota:** Puede hacer referencia a las variables en la consulta, pero debe declararlas antes de la sentencia CURSOR.

# Apertura del Cursor

## Sintaxis:

```
OPEN cursor_name;
```

- **Abrir el cursor para ejecutar la consulta e identificar el juego activo.**
- **Si la consulta no devuelve ninguna fila, no se emite ninguna excepción.**
- **Utilizar los atributos de cursor para probar el resultado después de una recuperación.**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Sentencia OPEN

Abra el cursor para ejecutar la consulta e identificar el juego de resultados, que comprende todas las filas que cumplen los criterios de búsqueda de la consulta. Ahora el puntero apunta a la primera fila del juego de resultados.

En la sintaxis, `cursor_name` es el nombre del cursor declarado anteriormente.

OPEN es una sentencia ejecutable que realiza las siguientes operaciones:

1. Asigna memoria dinámicamente para un área de contexto que contiene de forma eventual información crucial de procesamiento.
2. Analiza la sentencia SELECT.
3. Enlaza las variables de entrada, es decir, establece el valor de las variables de entrada al obtener las direcciones de su memoria.
4. Identifica el juego de resultados, es decir, el juego de filas que cumplen los criterios de búsqueda. Las filas del juego de resultados no se recuperan en variables cuando se ejecuta la sentencia OPEN. Más bien, la sentencia FETCH recupera las filas.
5. Sitúa el puntero justo antes de la primera fila del juego activo.

**Nota:** Si la consulta no devuelve ninguna fila cuando se abre el cursor, PL/SQL no emite una excepción. Sin embargo, puede probar el estado del cursor después de una recuperación.

Para los cursores declarados mediante la cláusula FOR UPDATE, la sentencia OPEN también bloquea esas filas.

# Recuperación de Datos del Cursor

## Ejemplos:

```
FETCH c1 INTO v_empid, v_ename;
```

```
...  
OPEN defined_cursor;  
LOOP  
    FETCH defined_cursor INTO defined_variables  
    EXIT WHEN ...;  
    ...  
    -- Process the retrieved data  
    ...  
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Sentencia **FETCH**

La sentencia **FETCH** se utiliza para recuperar los valores de la fila actual en variables de salida. Después de la recuperación, puede manipular las variables con otras sentencias. Para cada valor de columna devuelto por la consulta asociada al cursor, debe haber un variable correspondiente en la lista **INTO**. Asimismo, los tipos de dato deben ser compatibles. Puede recuperar los primeros 10 empleados de uno en uno:

```
DECLARE  
    v_empid employees.employee_id%TYPE;  
    v_ename employees.last_name%TYPE;  
    i NUMBER := 1;  
    CURSOR c1 IS  
        SELECT employee_id, last_name  
        FROM employees;  
BEGIN  
    OPEN c1;  
    FOR i IN 1..10 LOOP  
        FETCH c1 INTO v_empid, v_ename;  
        ...  
    END LOOP;  
END;
```

# Cierre del Cursor

## Sintaxis:

```
CLOSE cursor_name;
```

- **Cerrar el cursor al terminar el procesamiento de las filas.**
- **Volver a abrir el cursor si es necesario.**
- **No intentar recuperar datos de un cursor después de cerrarlo.**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Sentencia CLOSE

La sentencia CLOSE desactiva el cursor y el juego de resultados se convierte en no definido. Cierre el cursor al terminar el procesamiento de la sentencia SELECT. Este paso permite volver a abrir el cursor si es necesario. Por lo tanto, puede establecer un juego activo varias veces.

En la sintaxis, *cursor\_name* es el nombre del cursor declarado anteriormente.

No intente recuperar datos de un cursor después de cerrarlo o se emitirá la excepción INVALID\_CURSOR.

**Nota:** La sentencia CLOSE libera el área de contexto. Aunque se puede terminar el bloque PL/SQL sin cerrar los cursores, debe cerrar siempre los cursores que declare explícitamente para liberar recursos. Hay un límite máximo en el número de cursores abiertos por usuario, que determina el parámetro OPEN\_CURSORS del campo de parámetros de la base de datos. OPEN\_CURSORS = 50 por defecto.

```
...  
FOR i IN 1..10 LOOP  
    FETCH c1 INTO v_empid, v_ename; ...  
END LOOP;  
CLOSE c1;  
END;
```

## Atributos de Cursor Explícito

**Obtener información sobre el estado de un cursor.**

Atributo	Tipo	Descripción
%ISOPEN	BOOLEAN	Se evalúa en <b>TRUE</b> si el cursor está abierto.
%NOTFOUND	BOOLEAN	Se evalúa en <b>TRUE</b> si la recuperación más reciente no devuelve una fila.
%FOUND	BOOLEAN	Se evalúa en <b>TRUE</b> si la recuperación más reciente devuelve una fila; complemento de %NOTFOUND
%ROWCOUNT	NUMBER	Se evalúa en el número total de filas devueltas hasta el momento.

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Atributos de Cursor Explícito

Igual que ocurre con los cursores implícitos, existen cuatro atributos para obtener información sobre el estado de un cursor. Cuando se agregan al cursor o a la variable del cursor, estos atributos devuelven información útil sobre la ejecución de una sentencia DML.

**Nota:** No haga referencia a los atributos de cursor directamente en una sentencia SQL.

## Bucles FOR de Cursor

**Recuperar empleados de uno en uno hasta que no quede ninguno.**

**Ejemplo:**

```
DECLARE
  CURSOR c1 IS
    SELECT employee_id, last_name
    FROM   employees;
BEGIN
  FOR emp_record IN c1 LOOP
    -- implicit open and implicit fetch occur
    IF emp_record.employee_id = 134 THEN
      ...
    END LOOP; -- implicit close occurs
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Bucles FOR de Cursor

Un bucle FOR de cursor procesa las filas de un cursor explícito. Se abre el cursor, se recuperan las filas una vez para cada iteración del bucle y se cierra el cursor automáticamente una vez procesadas todas las filas. El bucle se termina automáticamente al final de la iteración en la que se recuperó la última fila.



## Cláusula FOR UPDATE

**Recuperar los pedidos de cantidades superiores a \$1.000 que se han procesado hoy.**

**Ejemplo:**

```
DECLARE
  CURSOR c1 IS
    SELECT customer_id, order_id
    FROM   orders
    WHERE  order_date = SYSDATE
          AND order_total > 1000.00
    ORDER BY customer_id
    FOR UPDATE NOWAIT;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Cláusula FOR UPDATE

Si el servidor de base de datos no puede adquirir los bloqueos de las filas que necesita en `SELECT FOR UPDATE`, esperará de manera indefinida. Puede utilizar la cláusula `NOWAIT` en la sentencia `SELECT FOR UPDATE` y realizar pruebas para el código de error que devuelve por un fallo al adquirir los bloqueos en un bucle. Por lo tanto, puede volver a intentar abrir el cursor *n* veces antes de terminar el bloque PL/SQL.

Si desea actualizar o suprimir filas con la cláusula `WHERE CURRENT OF`, debe especificar un nombre de columna en la cláusula `FOR UPDATE OF`.

Si tiene una tabla grande, puede conseguir un mejor rendimiento si utiliza la sentencia `LOCK TABLE` para bloquear todas las filas de la tabla. Sin embargo, si utiliza `LOCK TABLE`, no puede utilizar la cláusula `WHERE CURRENT OF` y debe utilizar la notación `WHERE column = identifier`.

## Cláusula WHERE CURRENT OF

### Ejemplo:

```
DECLARE
  CURSOR c1 IS
    SELECT salary FROM employees
    FOR UPDATE OF salary NOWAIT;
BEGIN
  ...
  FOR emp_record IN c1 LOOP
    UPDATE ...
      WHERE CURRENT OF c1;
    ...
  END LOOP;
  COMMIT;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Cláusula WHERE CURRENT OF

Puede actualizar filas basándose en criterios de un cursor.

Además, puede escribir una sentencia DELETE o UPDATE que contenga una cláusula WHERE CURRENT OF cursor\_name que haga referencia a la última fila procesada por la sentencia FETCH. Al utilizar esta cláusula, el cursor al que haga referencia debe existir y contener la cláusula FOR UPDATE en la consulta del cursor; de lo contrario, se producirá un error. Esta cláusula permite aplicar actualizaciones y supresiones a la fila consultada actualmente sin tener que hacer referencia explícitamente a la pseudocolumna ROWID.

## Detección de Errores Predefinidos del Servidor de Oracle

- Referencia al nombre estándar de la rutina de manejo de excepciones
- Excepciones predefinidas de ejemplo:
  - NO\_DATA\_FOUND
  - TOO\_MANY\_ROWS
  - INVALID\_CURSOR
  - ZERO\_DIVIDE
  - DUP\_VAL\_ON\_INDEX

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Detección de Errores Predefinidos del Servidor de Oracle

Detecte un error predefinido del servidor de Oracle haciendo referencia a su nombre estándar dentro de la rutina de manejo de excepciones correspondiente.

**Nota:** PL/SQL declara excepciones predefinidas en el paquete STANDARD.

Resulta útil tener en cuenta siempre las excepciones NO\_DATA\_FOUND y TOO\_MANY\_ROWS, que son las más comunes.

## Detección de Errores Predefinidos del Servidor de Oracle: Ejemplo

### Sintaxis:

```
BEGIN  SELECT ... COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    statement1;
    statement2;
  WHEN TOO_MANY_ROWS THEN
    statement1;
  WHEN OTHERS THEN
    statement1;
    statement2;
    statement3;
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Detección de Excepciones Predefinidas del Servidor de Oracle: Ejemplo

En el ejemplo de la transparencia, se imprime un mensaje para el usuario por cada excepción. Sólo se emite y maneja una excepción cada vez.

# Error No Predefinido

**Detectar el error del servidor de Oracle número –2292, que es una violación de la restricción de integridad.**

```
DECLARE
  e_products_invalid EXCEPTION;
  PRAGMA EXCEPTION_INIT (
    e_products_invalid, -2292);
  v_message VARCHAR2(50);
BEGIN
  . . .
EXCEPTION
  WHEN e_products_invalid THEN
    :g_message := 'Product ID
                  specified is not valid.';
  . . .
END;
```

Diagrama de anotación:

- 1. Señala a la declaración de la excepción: `e_products_invalid EXCEPTION;`
- 2. Señala a la asociación de la excepción al número de error: `PRAGMA EXCEPTION_INIT (e_products_invalid, -2292);`
- 3. Señala a la referencia a la excepción en la rutina de manejo de excepciones: `WHEN e_products_invalid THEN`

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Detección de Excepciones No Predefinidas del Servidor de Oracle

1. Declare el nombre de la excepción en la sección de declaraciones.

### Sintaxis

```
exception EXCEPTION;
```

En esta sintaxis, *exception* es el nombre de la excepción.

2. Asocie la excepción declarada al número de error del servidor de Oracle estándar con la sentencia `PRAGMA EXCEPTION_INIT`.

### Sintaxis

```
PRAGMA EXCEPTION_INIT(exception, error_number);
```

En esta sintaxis:

*exception*

es la excepción declarada anteriormente.

*error\_number*

es un número de error del servidor de Oracle estándar.

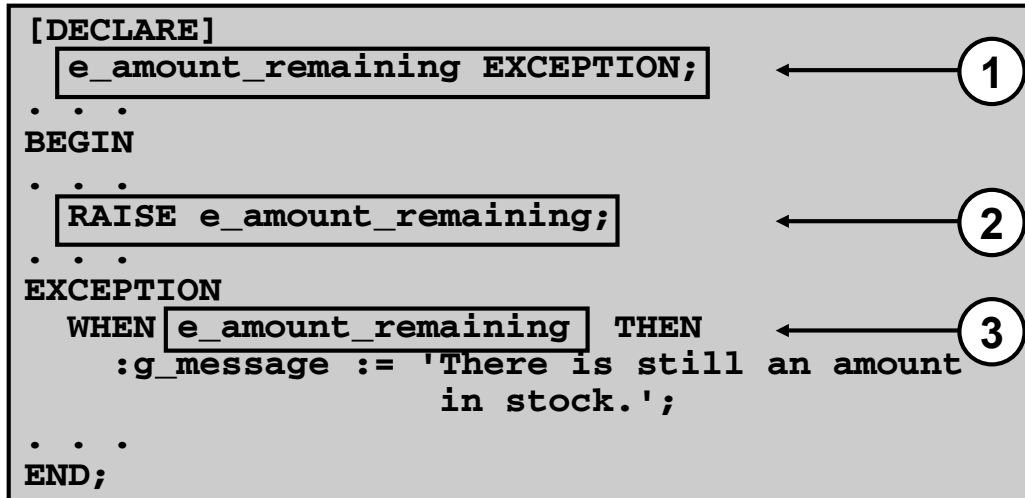
3. Haga referencia a la excepción declarada en la rutina de manejo de excepciones correspondiente.

En el ejemplo de la transparencia: Si hay material en stock, se detiene el procesamiento y se imprime un mensaje para el usuario.

# Excepciones Definidas por el Usuario

## Ejemplo:

```
[DECLARE]
  e_amount_remaining EXCEPTION;
.
.
.
BEGIN
  .
  .
  .
  RAISE e_amount_remaining;
  .
  .
  .
EXCEPTION
  WHEN e_amount_remaining THEN
    :g_message := 'There is still an amount
                  in stock.';
  .
  .
  .
END;
```



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Detección de Excepciones Definidas por el Usuario

Una excepción definida por el usuario se detecta al declararla o emitirla explícitamente.

1. Declare el nombre de la excepción definida por el usuario en la sección de declaraciones.

**Sintaxis:** *exception* EXCEPTION;

**donde:** *exception* es el nombre de la excepción.

2. Utilice la sentencia RAISE para emitir la excepción explícitamente en la sección ejecutable.

**Sintaxis:** RAISE *exception*;

**donde:** *exception* es la excepción declarada anteriormente.

3. Haga referencia a la excepción declarada en la rutina de manejo de excepciones correspondiente.

En el ejemplo de la transparencia: Este cliente tiene una regla de negocio que indica que un producto no se puede eliminar de la base de datos si aún hay stock del mismo. Puesto que no existen restricciones en vigor para forzar esta regla, el desarrollador la maneja explícitamente en la aplicación. Antes de realizar una operación DELETE en la tabla PRODUCT\_INFORMATION, el bloque consulta la tabla INVENTORIES para ver si hay stock del producto en cuestión. Si hay, emite una excepción.

**Nota:** Utilice sólo la sentencia RAISE en un manejador de excepciones para volver a emitir la misma excepción en el entorno de llamada.

## Procedimiento **RAISE\_APPLICATION\_ERROR**

### Sintaxis:

```
raise_application_error (error_number,  
                           message[, {TRUE | FALSE}]);
```

- **Permite emitir mensajes de error definidos por el usuario desde subprogramas almacenados**
- **Se llama sólo desde un subprograma almacenado de ejecución**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Procedimiento **RAISE\_APPLICATION\_ERROR**

El procedimiento **RAISE\_APPLICATION\_ERROR** permite comunicar una excepción predefinida de forma interactiva al devolver un código de error no estándar y un mensaje de error. Con **RAISE\_APPLICATION\_ERROR**, puede informar de los errores de la aplicación y evitar la devolución de excepciones no tratadas.

En la sintaxis, *error\_number* es un número especificado por el usuario para la excepción entre -20000 y -20999. *message* es el mensaje especificado por el usuario para la excepción. Es una cadena de caracteres de hasta 2.048 bytes.

TRUE | FALSE es un parámetro booleano opcional. Si está definido en TRUE, el error se coloca en la pila de errores anteriores. Si está definido en FALSE (valor por defecto), el error sustituye a todos los errores anteriores.

### Ejemplo

```
...  
EXCEPTION  
  WHEN NO_DATA_FOUND THEN  
    RAISE_APPLICATION_ERROR (-20201,  
      'Manager is not a valid employee.');
```

END;

## Procedimiento **RAISE\_APPLICATION\_ERROR**

- **Se utiliza en dos lugares diferentes:**
  - Sección **Executable**
  - Sección **Exception**
- **Devuelve condiciones de error al usuario de forma consistente con respecto a otros errores del servidor de Oracle**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### **Procedimiento RAISE\_APPLICATION\_ERROR: Ejemplo**

```
...  
DELETE FROM employees  
WHERE manager_id = v_mgr;  
IF SQL%NOTFOUND THEN  
    RAISE_APPLICATION_ERROR(-20202,  
        'This is not a valid manager');  
END IF;  
...
```





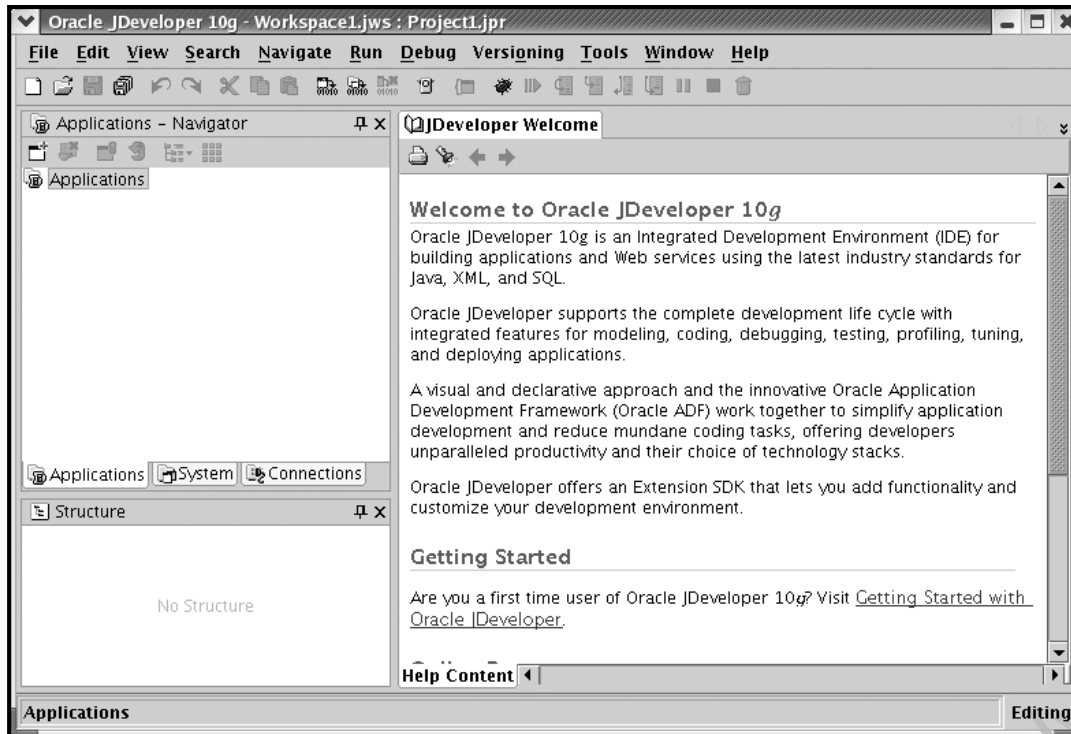
# JDeveloper

ORACLE®

Copyright © 2004, Oracle. Todos los Derechos Reservados.

Oracle Internal & OAI Use Only

# JDeveloper



ORACLE

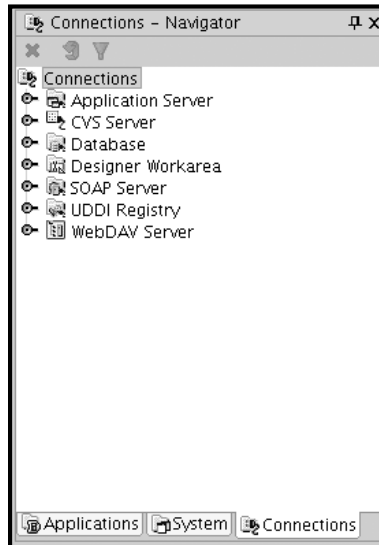
Copyright © 2004, Oracle. Todos los Derechos Reservados.

## JDeveloper

Oracle JDeveloper 10g es un entorno de desarrollo de integración (IDE) para desarrollar y desplegar aplicaciones Java y servicios Web. Soporta cada etapa del ciclo de vida de desarrollo de software (SDLC), del modelado al despliegue. Tiene funciones que permiten utilizar los últimos estándares de la industria para Java, XML y SQL y desarrollar una aplicación.

Oracle JDeveloper 10g inicia un nuevo enfoque al desarrollo J2EE con funciones que permiten un desarrollo visual y declarativo. Este enfoque innovador hace que el desarrollo J2EE sea sencillo y eficaz.

# Navegador de Conexiones



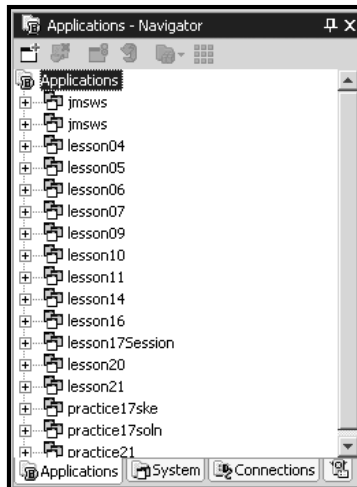
ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Navegador de Conexiones

Con Oracle JDeveloper 10g, puede almacenar la información necesaria para conectar a la base de datos en un objeto denominado “conexión”. Una conexión se almacena como parte de los valores de IDE y se puede exportar e importar para compartirla fácilmente entre grupos de usuarios. Una conexión tiene diferentes fines, desde el examen de la base de datos y la creación de aplicaciones hasta el despliegue.

# Navegador de Aplicaciones



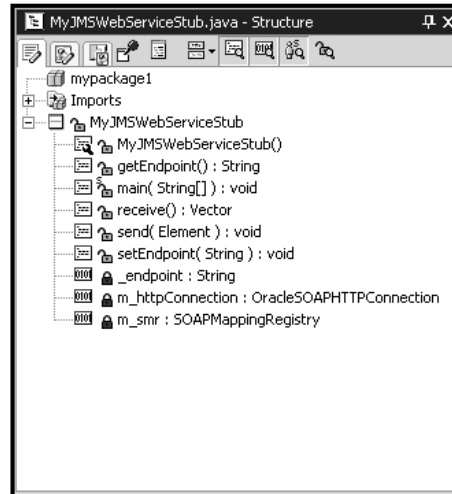
ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Navegador de Aplicaciones

El navegador de aplicaciones proporciona una vista lógica de la aplicación y de los datos que contiene. Además proporciona una infraestructura a la que las distintas extensiones pueden conectarse y utilizar para organizar los datos y menús de forma abstracta y consistente. Aunque el navegador de aplicaciones puede contener archivos individuales (como archivos de origen Java), está diseñado para consolidar datos complejos. Los tipos de dato complejos como los objetos de entidades, diagramas UML, EJB o servicios Web aparecen en el navegador como nodos únicos. Los archivos raw que componen estos nodos abstractos aparecen en la ventana Structure.

# Ventana Structure



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

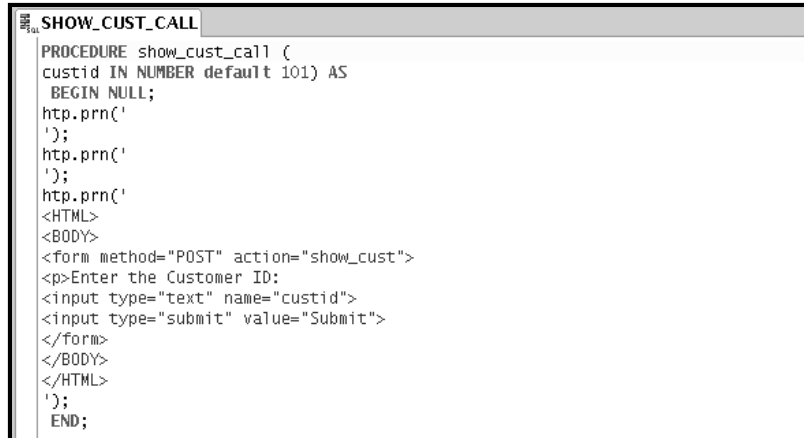
## Ventana Structure

La ventana Structure ofrece una vista estructural de los datos del documento seleccionado actualmente en la ventana activa de las ventanas que proporcionan estructura: navegadores, editores, visores y el Inspector de Propiedades.

En la ventana Structure, puede ver los datos del documento de distintas formas. Las estructuras disponibles están basadas en el tipo de documento. Para un archivo Java, puede ver la estructura del código, la estructura de la interfaz de usuario o los datos de modelo de interfaz de usuario. Para un archivo XML, puede ver la estructura XML, la estructura del diseño o los datos de modelo de interfaz de usuario.

La ventana Structure es dinámica y realiza siempre un seguimiento de la selección actual de la ventana activa (a menos que congele el contenido de la ventana en una vista concreta), ya que está relacionada con el editor que está actualmente activo. Cuando la selección actual es un nodo del navegador, se asume el editor por defecto. Para cambiar la vista en la estructura de la selección actual, seleccione un separador de estructura distinto.

# Ventana Editor



```
SQL SHOW_CUST_CALL
PROCEDURE show_cust_call (
  custid IN NUMBER default 101) AS
BEGIN NULL;
  http.prn('
');
  http.prn('
');
  http.prn('
<HTML>
<BODY>
<form method="POST" action="show_cust">
<p>Enter the Customer ID:
<input type="text" name="custid">
<input type="submit" value="Submit">
</form>
</BODY>
</HTML>
');
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Ventana Editor

Puede ver todos los archivos de proyectos en una única ventana del editor, abrir varias vistas del mismo archivo o abrir varias vistas de diferentes archivos.

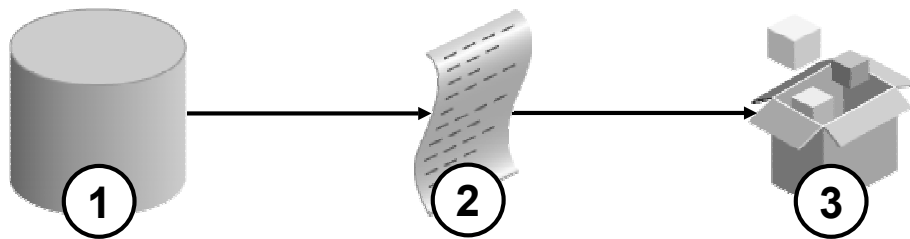
Los separadores situados en la parte superior de la ventana del editor son los separadores del documento. Al seleccionar un separador del documento, dicho documento se enfoca y se coloca en primer plano en la ventana del editor actual.

Los separadores situados en la parte inferior de la ventana del editor para un archivo concreto son los separadores del editor. Al seleccionar un separador del editor, el archivo se abre en ese editor.

# Despliegue de Procedimientos Almacenados de Java

**Antes de desplegar procedimientos almacenados de Java, realice los siguientes pasos:**

- 1. Cree una conexión de base de datos.**
- 2. Cree un perfil de despliegue.**
- 3. Despliegue los objetos.**



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

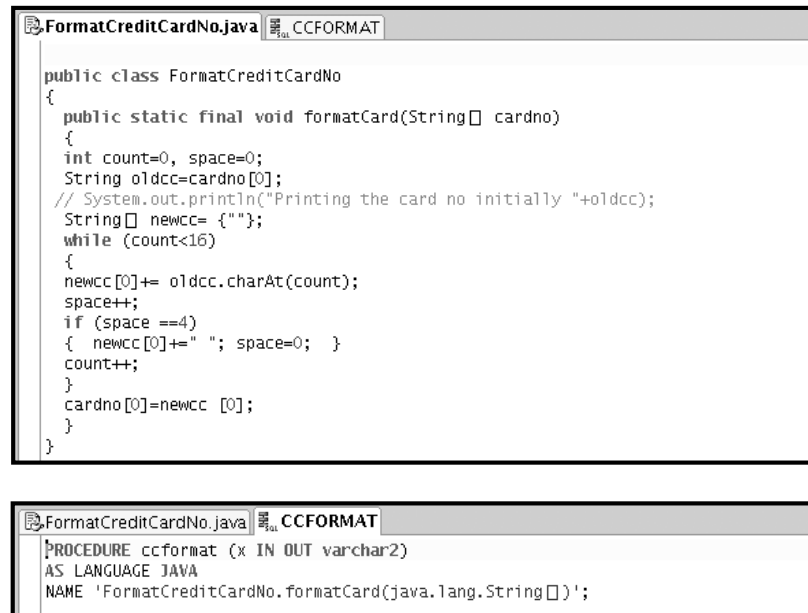
## Despliegue de Procedimientos Almacenados de Java

Cree un perfil de despliegue para los procedimientos almacenados de Java y, a continuación, despliegue las clases y, si lo desea, cualquier método estático público en JDeveloper con los valores del perfil.

El despliegue en la base de datos utiliza la información proporcionada en Deployment Profile Wizard y dos utilidades de la base de datos Oracle:

- `loadjava` carga la clase Java que contiene los procedimientos almacenados en una base de datos Oracle.
- `publish` genera los wrappers específicos de llamada PL/SQL para los métodos estáticos públicos cargados. La publicación permite que se llame a los métodos Java como funciones o procedimientos PL/SQL.

## Publicación de Java en PL/SQL



The image shows two screenshots from an Oracle SQL Developer window. The top screenshot displays a Java class named `FormatCreditCardNo` with a static method `formatCard` that takes a `String[]` array and formats it by inserting spaces every four characters. The bottom screenshot shows the same Java code wrapped in a PL/SQL procedure named `ccformat`, which is created using the `AS LANGUAGE JAVA` syntax.

```
public class FormatCreditCardNo
{
    public static final void formatCard(String[] cardno)
    {
        int count=0, space=0;
        String oldcc=cardno[0];
        // System.out.println("Printing the card no initially "+oldcc);
        String[] newcc= {" "};
        while (count<16)
        {
            newcc[0]+= oldcc.charAt(count);
            space++;
            if (space ==4)
            { newcc[0]+=" "; space=0; }
            count++;
        }
        cardno[0]=newcc [0];
    }
}
```

```
PROCEDURE ccformat (x IN OUT varchar2)
AS LANGUAGE JAVA
NAME 'FormatCreditCardNo.formatCard(java.lang.String[])';
```

ORACLE


Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Publicación de Java en PL/SQL

La transparencia muestra el código Java y cómo publicar el código Java en un procedimiento PL/SQL.



# Creación de Unidades de Programa



```
TEST_JDEV  
FUNCTION "TEST_JDEV" RETURN VARCHAR2  
AS  
BEGIN  
    RETURN('');  
END;
```

## Esqueleto de la Función

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

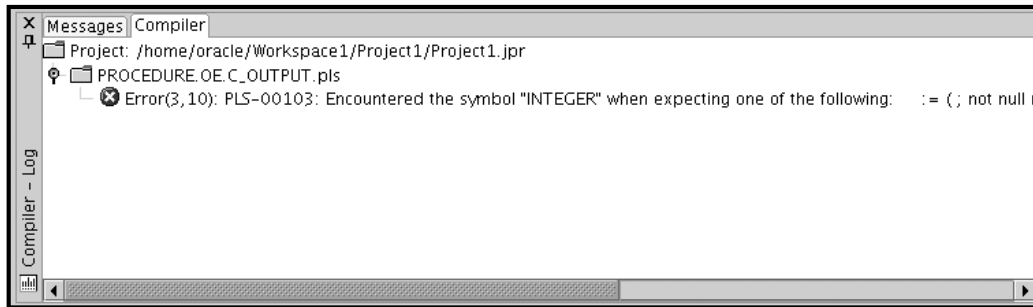
## Creación de Unidades de Programa

Para crear una unidad de programa PL/SQL:

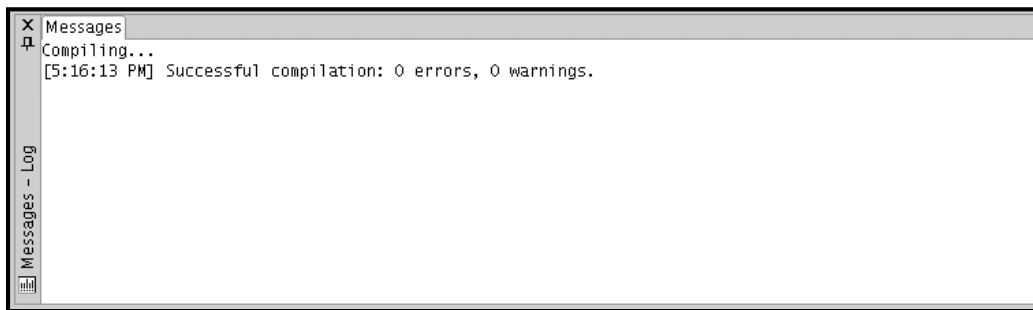
1. Seleccione View > Connection Navigator.
2. Amplíe Database y seleccione una conexión de base de datos.
3. En la conexión, amplíe un esquema.
4. Haga clic con el botón derecho del mouse en una carpeta que corresponda al tipo de objeto (Procedures, Packages, Functions).
5. Seleccione New PL/SQL object\_type. Aparece el recuadro de diálogo Create PL/SQL de la función, el paquete o el procedimiento.
6. Introduzca un nombre válido para la función, el paquete o el procedimiento y haga clic en OK.

Se crea una definición de esqueleto que se abre en la ventana Code Editor. A continuación, puede editar el subprograma para que se ajuste a sus necesidades.

# Compilación



## Compilación con Errores



## Compilación sin Errores

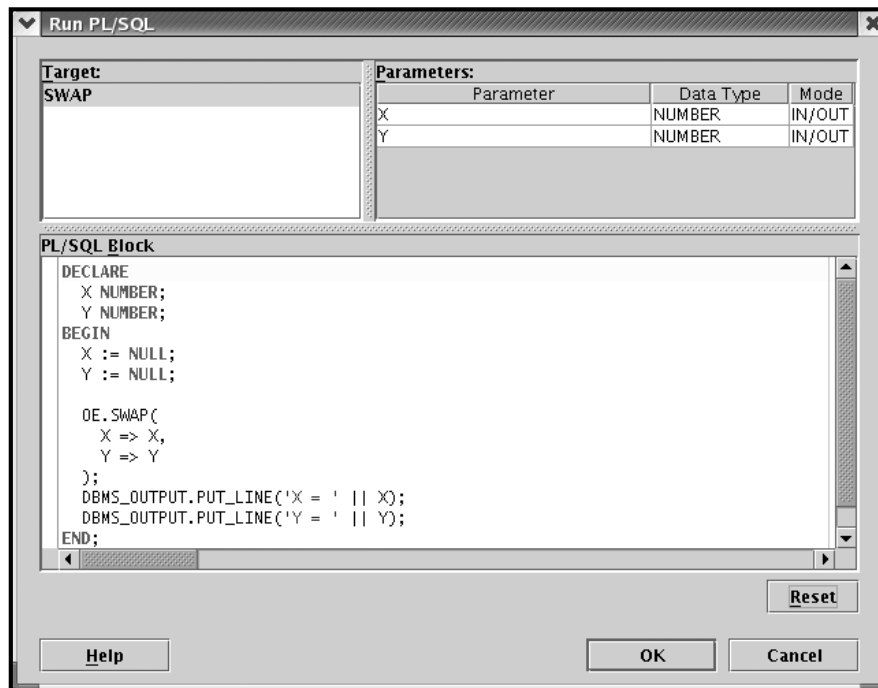
ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Compilación

Después de editar la definición de esqueleto, debe compilar la unidad de programa. Haga clic con el botón derecho del mouse en el objeto PL/SQL que necesita compilar en Connection Navigator y, a continuación, seleccione Compile. También puede pulsar CTRL + MAYÚS + F9 para compilar.

## Ejecución de una Unidad de Programa

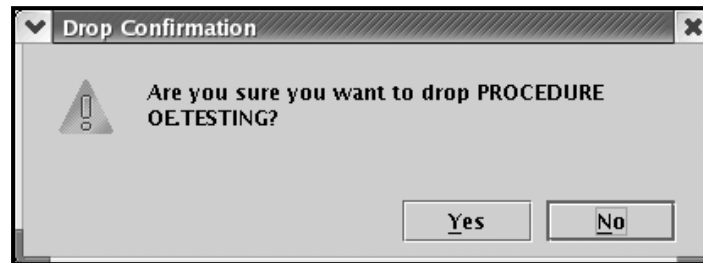


Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Ejecución de una Unidad de Programa

Para ejecutar la unidad de programa, haga clic con el botón derecho en el objeto y haga clic en Run. Aparece el recuadro de diálogo Run PL/SQL. Puede que tenga que cambiar los valores NULL por valores razonables que se transfieren a la unidad de programa. Después de cambiar los valores, haga clic en OK. La salida se mostrará en la ventana Message-Log.

## Borrado de una Unidad de Programa



ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Borrado de una Unidad de Programa

Para borrar una unidad de programa, haga clic con el botón derecho en el objeto y seleccione Drop. Aparecerá el recuadro de diálogo Drop Confirmation; haga clic en Yes. El objeto se borrará de la base de datos.

## Depuración de Programas PL/SQL

- **JDeveloper soporta dos tipos de depuración:**
  - **Local**
  - **Remota**
- **Necesita los siguientes privilegios para realizar una depuración de PL/SQL:**
  - **DEBUG ANY PROCEDURE**
  - **DEBUG CONNECT SESSION**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Depuración de Programas PL/SQL

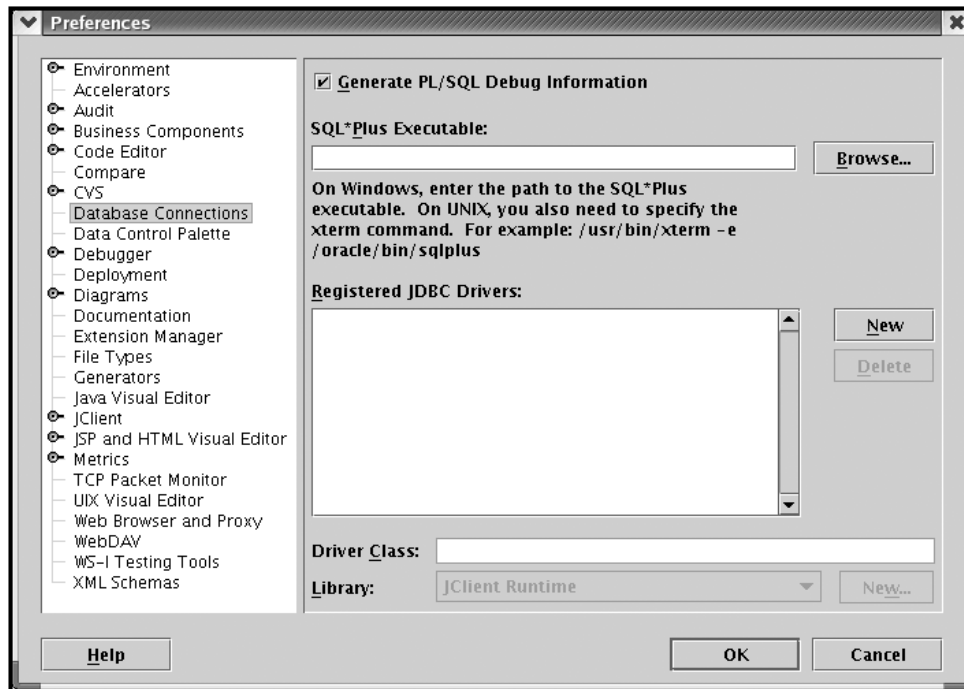
JDeveloper ofrece depuraciones tanto locales como remotas. Una sesión de depuración local se inicia al definir puntos de ruptura en archivos de origen y, a continuación, iniciar el depurador. La depuración remota necesita dos procesos de JDeveloper: un debugger y un debuggee que pueden residir en una plataforma diferente.

Para depurar un programa PL/SQL, éste debe estar compilado en modo INTERPRETED. No se puede depurar un programa PL/SQL compilado en modo NATIVE. Este modo se define en el archivo `init.ora` de la base de datos.

Los programas PL/SQL se deben compilar con la opción DEBUG activada. Esta opción se puede activar de distintas formas. Con SQL\*Plus, ejecute `ALTER SESSION SET PLSQL_DEBUG = true` para activar la opción DEBUG. A continuación, puede crear o volver a compilar el programa PL/SQL que desee depurar. Otra forma de activar la opción DEBUG es con el siguiente comando de SQL\*Plus:

```
ALTER <procedure, function, package> <name> COMPILE DEBUG;
```

# Depuración de Programas PL/SQL



Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Depuración de Programas PL/SQL (continuación)

Antes de iniciar la depuración, asegúrese de que la casilla de control Generate PL/SQL Debug Information está activada. Puede acceder al recuadro de diálogo con Tools > Preferences > Database Connections.

En lugar de probar manualmente las funciones y procedimientos PL/SQL como probablemente suele hacer desde SQL\*Plus o mediante un procedimiento ficticio en la base de datos, JDeveloper le permite probar estos objetos de forma automática. Con esta versión de JDeveloper, puede ejecutar y depurar unidades de programa PL/SQL. Por ejemplo, puede especificar parámetros que se están transfiriendo o valores de retorno de una función, lo que le ofrece más control sobre lo que se ejecuta y le proporciona detalles de salida de lo que se ha probado.

**Nota:** Los procedimientos o funciones de la base de datos Oracle pueden ser independientes o pertenecer a un paquete.

## Depuración de Programas PL/SQL (continuación)

Para ejecutar o depurar funciones, procedimientos y paquetes:

1. Cree una conexión de base de datos con Database Wizard.
2. En el navegador, amplíe el nodo Database para mostrar el nombre del esquema y el usuario concreto de la base de datos.
3. Amplíe el nodo Schema.
4. Amplíe el nodo adecuado en función de lo que desea depurar: el cuerpo de un procedimiento, función o paquete.
5. (Opcional para sólo depuración) Seleccione la función, el procedimiento o el paquete que desea depurar y haga clic dos veces para abrirlo en el editor de códigos.
6. (Opcional para sólo depuración) Defina un punto de ruptura en el código PL/SQL haciendo clic a la izquierda del margen.

**Nota:** El punto de ruptura se debe definir en una línea ejecutable del código. Si el depurador no se para, puede que el punto de ruptura no se haya definido en una línea ejecutable del código; compruebe que el punto de ruptura se ha verificado. Verifique también que se cumplen los requisitos de depuración de PL/SQL. Concretamente, asegúrese de que el programa PL/SQL se ha compilado en modo INTERPRETED.

7. Asegúrese de que están seleccionados actualmente el editor de códigos o el procedimiento en el navegador.
8. Haga clic en el botón Debug de la barra de herramientas o, si desea ejecutar sin depuración, haga clic en el botón Run de la barra de herramientas.
9. Se muestra el recuadro de diálogo Run PL/SQL.
  - Seleccione un destino con el nombre del procedimiento o función que desea depurar. Tenga en cuenta que el contenido de los recuadros Parameters y PL/SQL Block cambian dinámicamente cuando cambia el destino.

**Nota:** Sólo podrá elegir el destino si selecciona ejecutar o depurar un paquete que contiene más de una unidad de programa.
  - El recuadro Parameters muestra los argumentos del destino (en caso que corresponda).
  - El recuadro PL/SQL Block muestra el código que ha generado JDeveloper de forma personalizada para el destino seleccionado. Según la operación que realice la función o el procedimiento, puede que tenga que cambiar los valores NULL por valores razonables para que estos se transfieran al procedimiento, la función o el paquete. En algunos casos, puede que tenga que escribir más código para inicializar los valores que se van a transferir como argumentos. En este caso, puede editar el texto de bloque PL/SQL según sea necesario.
10. Haga clic en OK para ejecutar o depurar el destino.
11. Analice la información de salida que se muestra en la ventana Log.

En el caso de las funciones, se muestra el valor de retorno. También se muestran los mensajes DBMS\_OUTPUT.

# Definición de Puntos de Ruptura

```
PROCEDURE "TEST_DEBUG" (p_cust_id IN NUMBER)
AS
v_cust customers%ROWTYPE;
BEGIN
  SELECT * into v_cust
  FROM customers
  where customer_id = p_cust_id;
  dbms_output.put_line('Customer ID is ' || v_cust.customer_id);
  dbms_output.put_line('Customer Name is ' || v_cust.cust_first_name);
END;
```

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

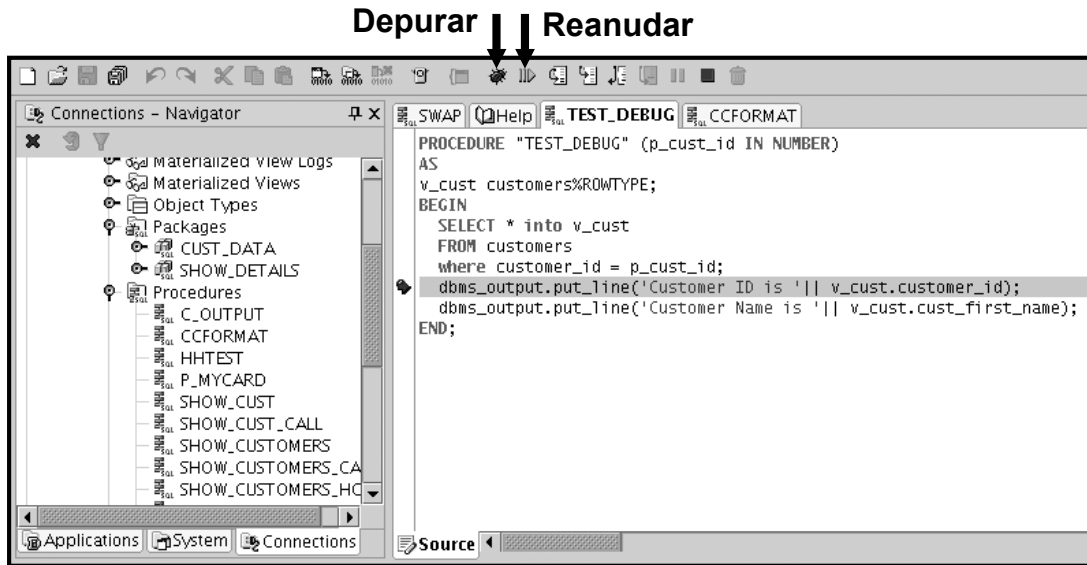
## Definición de Puntos de Ruptura

Los puntos de ruptura ayudan a examinar los valores de las variables en el programa. Se trata de un disparador del programa que, al alcanzarlo, pone en pausa la ejecución del programa. Esto permite al usuario examinar los valores de algunas o todas las variables del programa. Al definir puntos de ruptura en áreas potencialmente problemáticas del código de origen, puede ejecutar el programa hasta que la ejecución alcance una ubicación que desee depurar. Cuando la ejecución del programa encuentra un punto de ruptura, el programa realiza una pausa y el depurador muestra la línea que contiene el punto de ruptura en el editor de códigos. A continuación, puede utilizar el depurador para ver el estado del programa. Los puntos de ruptura son flexibles en el sentido de que se pueden definir antes de iniciar la ejecución del programa o en cualquier momento durante la depuración.

Para definir un punto de ruptura en el editor de códigos, haga clic en el margen izquierdo junto a una línea de código ejecutable. El depurador no verifica los puntos de ruptura definidos en líneas de comentarios, líneas en blanco, declaraciones o cualquier otro tipo de líneas no ejecutables de código sino que se tratan como no válidas.



# Análisis de Código



ORACLE

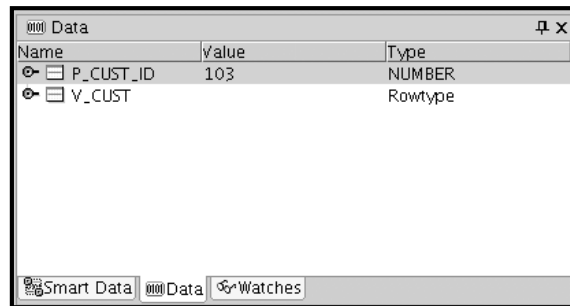
Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Análisis de Código

Después de definir un punto de ruptura, inicie el depurador haciendo clic en el icono Debug.

El depurador realizará una pausa en la ejecución del programa en el punto en el que está definido el punto de ruptura. En este punto, puede comprobar los valores de las variables. Puede continuar con la ejecución del programa haciendo clic en el icono Resume. El depurador avanzará hasta el siguiente punto de ruptura. Después de ejecutar todos los puntos de ruptura, el depurador detendrá la ejecución del programa y mostrará los resultados en el área Debugging – Log.

# Examen y Modificación de Variables



**Data window**

ORACLE

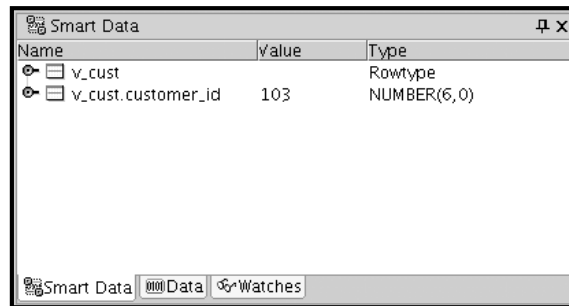
Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Examen y Modificación de Variables

Cuando la depuración está en modo ON, puede examinar y modificar el valor de las variables con las ventanas Data, Smart Data y Watches. Puede modificar los valores de datos del programa durante una sesión de depuración para probar las correcciones de bugs hipotéticos durante una ejecución del programa. Si percibe que una modificación corrige un error del programa, puede salir de la sesión de depuración, corregir el código del programa de acuerdo con esto y volver a compilar el programa para que la corrección sea permanente.

La ventana Data se utiliza para mostrar información sobre las variables del programa. La ventana Data muestra los argumentos, las variables locales y los campos estáticos para el contexto actual, que se controla con la selección de la ventana Stack. Si pasa a un nuevo contexto, la ventana Data se actualiza para mostrar los datos del nuevo contexto. Si el programa actual se ha compilado sin información de depuración, no podrá ver las variables locales.

## Examen y Modificación de Variables



Name	Value	Type
v_cust		Rowtype
v_cust.customer_id	103	NUMBER(6,0)

**Smart Data window**

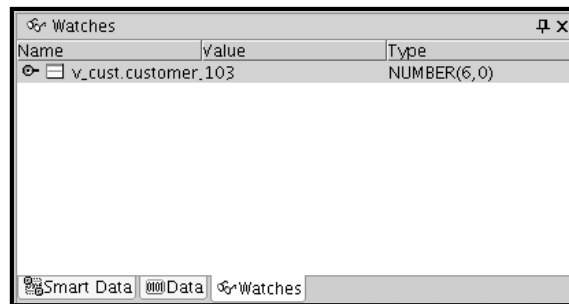
ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

### Examen y Modificación de Variables (continuación)

A diferencia de la ventana Data que muestra todas las variables del programa, la ventana Smart Data muestra sólo los datos relevantes para el código de origen que está analizando.

# Examen y Modificación de Variables



**Ventana Watches**

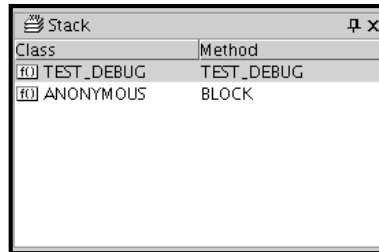
ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Examen y Modificación de Variables (continuación)

Una comprobación permite controlar los valores que cambian de las variables o expresiones durante la ejecución del programa. Al introducir una expresión de comprobación, la ventana de comprobaciones 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.

# Examen y Modificación de Variables



**Ventana Stack**

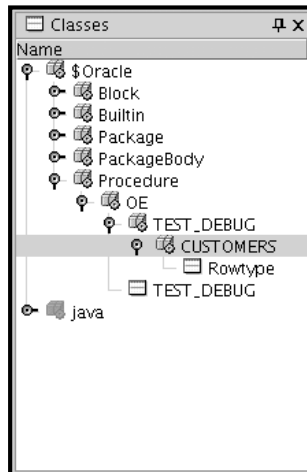
ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Examen y Modificación de Variables (continuación)

Puede activar la ventana Stack utilizando View > Debugger > Stack. Esto muestra la pila de llamadas del thread actual. Al seleccionar una línea en la ventana Stack, la ventana Data, la ventana Watch y todas las demás ventanas se actualizan para mostrar los datos de la clase seleccionada.

# Examen y Modificación de Variables



**Ventana Classes**

ORACLE

Copyright © 2004, Oracle. Todos los Derechos Reservados.

## Examen y Modificación de Variables (continuación)

La ventana Classes muestra todas las clases que se están cargando actualmente para ejecutar el programa. Si se utiliza con Oracle Java Virtual Machine (OJVM), también muestra el número de instancias de una clase y la memoria utilizada por esas instancias.

# Índice

Oracle Internal & OAI Use Only

## **A**

Acción del Disparador 10-6, 10-17, 10-18

Ajustar 1-25, 4-19, 4-21, 4-23

Ajustar Paquetes 4-21

Análisis 1-26, 6-3

## **B**

BFILE 5-17, 5-19, 9-3, 9-9, 9-10, 9-11, 9-12, 9-13, 9-14, 9-15, 9-16, 9-17, 9-20, 9-21, 9-22, 9-23

Bibliotecas Compartidas 12-4, 12-5, 12-6, 12-8

Bloque de Inicialización de Paquete 4-10

Bloque PL/SQL I-9, I-10, I-11, I-12, I-14, I-15, 1-3, 1-4, 1-12, 2-3, 2-4, 5-4, 5-21, 5-24, 5-25, 6-6, 6-12, 6-14, 7-11, 7-12, 7-14, 10-03, 10-10, 11-04, D-3, D-12, D-22

BULK\_EXCEPTIONS 7-15, 7-16

## **C**

CALL 10-5, 10-10, 11-7, C-8

Compilador PL/SQL 4-21, 7-12, 7-21, 7-23, 8-25, 12-6, 12-10, 12-11

Crear Objetos DIRECTORY 9-12

Cualificador OLD y NEW 10-15, 10-16

Cuerpo del Paquete I-13, 3-4, 3-5, 3-6, 3-7, 3-8, 3-9, 3-10, 3-11, 3-12, 3-13, 3-14, 3-15, 3-16, 4-9, 4-10, 4-21, 6-13, 8-3, 8-25, 8-29, 8-30

## **D**

DBMS\_LOB 5-6, 9-8, 9-13, 9-16, 9-17, 9-23, 9-24, 9-25, 9-26, 9-28, 9-29, 9-30, 9-31, 9-32

DBMS\_METADATA 6-18, 6-20

DBMS\_SQL 6-4, 6-5, 6-14, 6-15, 6-16, 6-17, 6-32, 7-9

DBMS\_WARNING 12-10, 12-13, 12-14, 12-15, 12-16, 12-17, 12-18

Declaración Anticipada 3-10, 4-08, 4-09

Dependencias Locales 8-5, 8-6, 8-7, 8-11

Dependencias Remotas 8-13, 8-14, 8-15, 8-17, 8-18

DEPTREE 8-9, 8-10, 8-32

DIRECTORY 5-6, 5-9, 5-10, 5-17, 9-11, 9-12, 9-13, 9-14, 9-15, 9-17

Disparador de Fila 10-5, 10-6, 10-8, 10-9, 10-14, 10-15, 10-17, 10-18, 10-20, 10-28, 11-8

Disparador de Sentencia 10-5, 10-6, 10-8, 10-9, 10-11, 10-18, 11-8, 11-10



## **F**

FETCH\_DDL 6-21

FILE\_TYPE 5-9, 5-10

Firma 8-15, 8-16, 8-24

Función BFILENAME 9-13, 9-15

Función I-9, I-12, I-13, 1-25, 2-3, 2-4, 2-5, 2-7, 2-10, 2-11, 2-14, 3-3, 3-10, 3-11, 4-4, 4-7, 4-11, 4-12, 4-14, 7-7, 7-24, 9-5, 9-13, 9-15

Función PL/SQL Definida por el Usuario 2-11

## **I**

IDEPTREE 8-9, 8-10, 8-32

Interpretada 12-3, 12-6, 12-7, 12-8, 12-9

IS\_OPEN 5-7, 5-9

## **L**

LOB 6-21, 9-3, 9-4, 9-5, 9-6, 9-7, 9-18, 9-19, 9-20, 9-24, 9-25, 9-26, 9-30, 9-31, 9-32, D-3

LOB Internos 9-3, 9-7, 9-8, 9-31

LOB Temporales 9-31, 9-32

Localizador LOB 9-6, 9-8, 9-10, 9-13, 9-15, 9-16, 9-19, 9-24, 9-25, 9-26, 9-32

Llamar a un Procedimiento 1-5, 1-9, 1-19

## **M**

Mecanismo de Seguridad 9-10

Modo de Parámetro 1-8, 1-18, 2-4, 6-6

## **N**

Nivel de Pureza 4-11, 4-12

## **O**

OCI 9-8, 9-9, 9-11, 9-15, 9-18, 9-25

## **P**

Procedimiento I-9, I-11, 1-3, 1-4, 1-5, 1-19, 1-24, 1-25, 1-26, 2-11, 2-16

Parámetros Formales 1-07, 1-08, 1-14, 1-16, 4-03

PLSQL\_NATIVE\_LIBRARY\_DIR 12-5, 12-6

PLSQL\_NATIVE\_LIBRARY\_SUBDIR\_COUNT 12-5, 12-6

Parámetros Reales 1 – 7

PARAM\_VALUE 12-8

PLSQL\_COMPILER\_FLAGS 12-6, 12-7, 12-8, 12-9

Privilegio READ 9-12, 9-13, 9-14

Privilegio CREATE ANY DIRECTORY 09-14

Paquete DBMS\_LOB 5-6, 9-8, 9-9, 9-15, 9-20, 9-21, 9-22

Paquete UTL\_FILE 5-4, 5-6, 5-7, 5-8, 5-10

Paquete DBMS\_OUTPUT 5-3, 5-4, 5-5, 5-14, 10-28, I-11, I-17, I-18

PLSQL\_WARNINGS 12-10, 12-11, 12-12, 12-13

## **R**

Recuperar 4-15, 6-3, 6-5, 6-10, 6-20, 7-14, 7-19, D-18, D-20, D-21, D-24

Registro de Hora 8-15, 8-17, 8-18, 8-19, 8-20, 8-21, 8-22, 8-23, 8-24

## **S**

Sentencia CREATE PROCEDURE I-11, 1-4, 1-5

Sentencia RETURN I-12, 2-3, 2-4, 2-5, 2-6, 2-16

SESSION\_MAX\_OPEN\_FILES 9-14

Sobrecarga 3-17, 4-3, 4-5, 4-6, 4-7, 4-22

Sobrecargar 4-4

SQL Dinámico 6-3, 6-4, 6-5, 6-6, 6-7, 6-8, 6-9, 6-10, 6-13, 6-14, 6-17, 7-9, 12-16

## **T**

Tabla Mutante 11-8, 11-9, 11-10, 11-18

Temporización del Disparador 10-7

Tipo de Dato RETURN I-12, 2-4, 2-11

Tipo de Disparador 10-6

TO\_BLOB() 9-19

TO\_CLOB() 9-19

Transacciones Autónomas 7-10, 7-11, 7-12

## U

USER\_DEPENDENCIES 8-8

USER\_ERRORS 11-14

USER\_PLSQL\_OBJECTS 12-8

user\_stored\_settings 12-8

USER\_TRIGGERS 11-14, 11-15, 11-16

UTL\_FILE 5-4, 5-6, 5-7, 5-8, 5-9, 5-10, 5-20

utl\_file\_dir 5-6, 5-9

UTL\_MAIL 5-4, 5-15, 5-16, 5-17, 5-19, C-19

## V

Valor LOB 9-5, 9-6, 9-7, 9-20, 9-21, 9-22, 9-25, 9-26, 9-30

Variables de Host 1-11, 1-12

Variables Ligadas 1-11, 2-4, 6-3, 6-4, 6-5, 6-8, 6-12, 6-14, 6-16

Oracle Internal & OAI Use Only

Oracle Internal & OAI Use Only