

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

Volumen II • Guía del Alumno

D64250CS10
Edición 1.0
Enero de 2010
D73735

ORACLE®

Autor

Lauran Serhal

**Colaboradores Técnicos
y Responsables de Revisión**

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

Redactores

Arijit Ghosh
 Raj Kumar

Editores

Pavithran Adka
 Shaik Mahaboob Basha
 Sheryl Domingue

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

Renuncia

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

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

Aviso sobre Restricción de Derechos

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

U.S. GOVERNMENT RIGHTS

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

Aviso de Marca Registrada

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

Contenido

I Introducción

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

1 Creación de Procedimientos

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

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

2 Creación de Funciones y Depuración de Subprogramas

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

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

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

3 Creación de Paquetes

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

4 Trabajar con Paquetes

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

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

Prueba 5-27

Resumen 5-28

Práctica 5: Visión General 5-29

6 Uso de SQL Dinámico

Objetivos 6-2

Agenda de la Lección 6-3

Flujo de Ejecución de SQL 6-4

Trabajar con SQL Dinámico 6-5

Uso de SQL Dinámico 6-6

SQL Dinámico Nativo (NDS) 6-7

Uso de la Sentencia EXECUTE IMMEDIATE 6-8

Métodos Disponibles para Usar NDS 6-9

SQL Dinámico con una Sentencia DDL: Ejemplos 6-11

SQL Dinámico con Sentencias DML 6-12

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

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

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

Agenda de la Lección 6-16

Uso del Paquete DBMS_SQL 6-17

Uso de los Subprogramas del Paquete DBMS_SQL 6-18

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

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

Prueba 6-22

Resumen 6-23

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

7 Consideraciones de Diseño para Código PL/SQL

Objetivos 7-2

Agenda de la Lección 7-3

Estandarización de Constantes y Excepciones 7-4

Estandarización de Excepciones 7-5

Estandarización del Manejo de Excepciones 7-6

Estandarización de Constantes 7-7

Subprogramas Locales 7-8

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

Especificación de Derechos del Invocador: Definición de AUTHID en

CURRENT_USER 7-10

Transacciones Autónomas 7-11

Funciones de las Transacciones Autónomas 7-12

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

8 Creación de Disparadores

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

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

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

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

Sentencias CALL en Disparadores	9-19
Ventajas de los Disparadores de Eventos de Base de Datos	9-20
Privilegios del Sistema Necesarios para Gestionar Disparadores	9-21
Instrucciones para el Diseño de Disparadores	9-22
Prueba	9-23
Resumen	9-24
Práctica 9: Visión General	9-25

10 Uso del Compilador PL/SQL

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

Procedimientos DBMS_WARNING: Ejemplo	10-29
Funciones DBMS_WARNING: Sintaxis, Parámetros y Valores Permitidos	10-30
Funciones DBMS_WARNING: Ejemplo	10-31
Uso de DBMS_WARNING: Ejemplo	10-32
Uso del Mensaje de Advertencia PLW 06009	10-34
Advertencia PLW 06009: Ejemplo	10-35
Prueba	10-36
Resumen	10-37
Práctica 10: Visión General	10-38

11 Gestión del Código PL/SQL

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

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

12 Gestión de Dependencias

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

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

Apéndice A: Prácticas y Soluciones

Apéndice AP: Adicional: Prácticas y Soluciones

Apéndice B: Descripciones de las Tablas

Apéndice C: Uso de SQL Developer

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

Apéndice D: Uso de SQL*Plus

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

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

Apéndice E: Uso de JDeveloper

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

Apéndice F: Revisión de PL/SQL

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

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

Apéndice G: Estudios para Implantación de Disparadores

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

Apéndice H: Uso de los Paquetes DBMS_SCHEDULER y HTP

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

Jose Enrique Lozano (lozanojose@gmail.com) has a
non-transferable license to use this Student Guide.

Apéndice A

Prácticas y Soluciones

Jose Enrique Lozano (lozanojose@gmail.com) has a
non-transferable license to use this Student Guide.

Tabla de Contenido

Prácticas y Soluciones de la Lección I.....	4
Práctica I-1: Identificación de los Recursos Disponibles de SQL Developer	5
Práctica I-2: Creación y Uso de una Nueva Conexión de Base de Datos de SQL Developer.....	6
Práctica I-3: Exploración de las Tablas de Esquema y Creación y Ejecución de un Bloque Anónimo Simple.....	7
Práctica I-4: Definición de Algunas Preferencias de SQL Developer	8
Práctica I-5: Acceso a la Biblioteca de Documentación en Línea de Oracle Database 11g Versión 2	9
Soluciones a la Práctica I-1: Identificación de los Recursos Disponibles de SQL Developer	10
Soluciones a la Práctica I-2: Creación y Uso de una Nueva Conexión de Base de Datos de SQL Developer.....	12
Soluciones a la Práctica I-3: Exploración de las Tablas de Esquema y Creación y Ejecución de un Bloque Anónimo Simple.....	15
Soluciones a la Práctica I-4: Definición de Algunas Preferencias de SQL Developer	19
Soluciones a la Práctica I-5: Acceso a la Biblioteca de Documentación en Línea de Oracle Database 11g Versión 2	23
Prácticas y Soluciones de la Lección 1	24
Práctica 1-1: Creación, Compilación y Llamada de Procedimientos	25
Soluciones a la Práctica 1-1: Creación, Compilación y Llamada de Procedimientos.....	27
Prácticas y Soluciones de la Lección 2	38
Práctica 2-1: Creación de Funciones.....	39
Práctica 2-2: Introducción al Depurador de SQL Developer.....	41
Soluciones a la Práctica 2-1: Creación de Funciones	42
Soluciones a la Práctica 2-2: Introducción al Depurador de SQL Developer.....	48
Prácticas y Soluciones de la Lección 3	58
Práctica 3-1: Creación y Uso de Paquetes	59
Soluciones a la Práctica 3-1: Creación y Uso de Paquetes	61
Prácticas y Soluciones de la Lección 4	68
Práctica 4-1: Trabajar con Paquetes.....	69
Soluciones a la Práctica 4-1: Trabajar con Paquetes	73
Prácticas y Soluciones de la Lección 5	102
Práctica 5-1: Uso del Paquete UTL_FILE	103
Soluciones a la Práctica 5-1: Uso del Paquete UTL_FILE.....	104
Prácticas y Soluciones de la Lección 6	108
Práctica 6-1: Uso de SQL Dinámico Nativo.....	109
Soluciones a la Práctica 6-1: Uso de SQL Dinámico Nativo.....	111
Prácticas y Soluciones de la Lección 7	121
Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas	122
Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas	124

Prácticas y Soluciones de la Lección 8	145
Práctica 8-1: Creación de Disparadores de Sentencia y de Fila.....	146
Soluciones a la Práctica 8-1: Creación de Disparadores de Sentencia y de Fila	148
Prácticas y Soluciones de la Lección 9	157
Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes	158
Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes	161
Prácticas y Soluciones de la Lección 10	174
Práctica 10-1: Uso de Parámetros y Advertencias del Compilador PL/SQL.....	175
Soluciones a la Práctica 10-1: Uso de Parámetros y Advertencias del Compilador PL/SQL	176
Prácticas y Soluciones de la Lección 11	185
Práctica 11-1: Uso de la Compilación Condicional.....	186
Soluciones a la Práctica 11-1: Uso de la Compilación Condicional.....	188
Prácticas y Soluciones de la Lección 12	195
Práctica 12-1: Gestión de Dependencias en el Esquema	196
Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema	197

Prácticas y Soluciones de la Lección I

Ésta es la primera de varias prácticas de este curso. Las soluciones (si las necesita) se pueden encontrar en el “Apéndice A: Prácticas y Soluciones”. Las prácticas están pretendiendo abarcar la mayoría de los temas que se presentan en la lección correspondiente.

Nota: si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

En esta práctica, revisará los recursos disponibles de SQL Developer. También aprenderá sobre su cuenta de usuario, que utilizará en este curso. A continuación, iniciará SQL Developer, creará una nueva conexión de base de datos, examinará las tablas del esquema, además de crear y ejecutar un bloque anónimo simple. También definirá algunas preferencias de SQL Developer, ejecutará sentencias SQL y ejecutará un bloque PL/SQL anónimo mediante la hoja de trabajo de SQL. Por último, accederá y marcará la documentación de Oracle Database 11g y de otros sitios web útiles que puede utilizar en este curso.

Práctica I-1: Identificación de los Recursos Disponibles de SQL Developer

En esta práctica, revisará los recursos disponibles de SQL Developer.

- 1) Familiarícese con Oracle SQL Developer según sea necesario mediante el Apéndice C: Uso de SQL Developer.
- 2) Acceda a la página inicial de SQL Developer disponible en línea en la dirección: http://www.oracle.com/technology/products/database/sql_developer/index.html
- 3) Marque la página para acceder más fácilmente a ella en el futuro.
- 4) Acceda al tutorial de SQL Developer disponible en línea en la dirección: <http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>
- 5) Realice una vista previa y pruebe con los enlaces y las demostraciones disponibles del tutorial todo lo necesario, sobre todo con los enlaces “Creación de una Conexión a la Base de Datos” y “Acceso a los Datos”.

Práctica I-2: Creación y Uso de una Nueva Conexión de Base de Datos de SQL Developer

En esta práctica, iniciará SQL Developer mediante la información de conexión y creará una nueva conexión de base de datos.

- 1) Inicie SQL Developer mediante el identificador y la contraseña de usuario que le ha proporcionado el instructor, por ejemplo ora61.
- 2) Cree una conexión de base de datos con la siguiente información:
 - a) Connection Name: MyDBConnection
 - b) Username: ora61
 - c) Password: ora61
 - d) Hostname: introduzca el nombre de host de la computadora
 - e) Port: 1521
 - f) SID: ORCL
- 3) Pruebe la nueva conexión. Si el estado es Success, conecte a la base de datos con esta nueva conexión:
 - a) Haga clic dos veces en el ícono MyDBConnection de la página con separadores Connections.
 - b) Haga clic en el botón Test de la ventana New>Select Database Connection. Si el estado es Success, haga clic en el botón Connect.

Práctica I-3: Exploración de las Tablas de Esquema y Creación y Ejecución de un Bloque Anónimo Simple

En esta práctica, examinará las tablas de esquema y creará y ejecutará un bloque anónimo simple.

- 1) Examine la estructura de la tabla EMPLOYEES y muestre sus datos.
 - a) Amplíe la conexión MyDBConnection. Para ello, haga clic en el signo más situado junto a ella.
 - b) Amplíe el ícono Tables. Para ello, haga clic en el signo más situado junto a él.
 - c) Visualice la estructura de la tabla EMPLOYEES.
 - 2) Examine la tabla EMPLOYEES y muestre sus datos.
 - 3) Utilice la hoja de trabajo de SQL para seleccionar los apellidos y los salarios de todos los empleados cuyo salario anual sea mayor de 10.000 dólares. Utilice tanto el ícono Execute Statement (F9) como el ícono Run Script (F5) para ejecutar la sentencia SELECT. Revise los resultados de ambos métodos de ejecución de las sentencias SELECT en los separadores adecuados.
- Nota:** dedique unos minutos a familiarizarse con los datos o a consultar el Apéndice B, donde se proporciona la descripción y los datos de todas las tablas del esquema HR que utilizará en este curso.
- 4) Cree y ejecute un bloque anónimo simple cuya salida sea “Hello World”.
 - a) Active SET SERVEROUTPUT ON para mostrar la salida de las sentencias del paquete DBMS_OUTPUT.
 - b) Utilice el área SQL Worksheet para introducir el código del bloque anónimo.
 - c) Haga clic en el ícono Run Script (F5) para ejecutar el bloque anónimo.

Práctica I-4: Definición de Algunas Preferencias de SQL Developer

En esta práctica, definirá algunas preferencias de SQL Developer.

- 1) En el menú SQL Developer, acceda a Tools > Preferences. Se mostrará la ventana Preferences.
- 2) Amplíe la opción Code Editor y, a continuación, haga clic en la opción Display para que aparezca la sección “Code Editor: Display”. Esta sección contiene opciones generales de la apariencia y el comportamiento del editor de códigos.
 - a) Introduzca 100 en el cuadro de texto Right Margin Column de la sección Show Visible Right Margin. Esto hace que aparezca un margen derecho que puede definir para controlar la longitud de las líneas de código.
 - b) Haga clic en la opción Line Gutter. Esta opción especifica opciones para el canal de línea (margen izquierdo del editor de códigos). Seleccione la casilla de control Show Line Numbers para mostrar los números de línea de código.
- 3) Haga clic en la opción Worksheet Parameters de Database. En el cuadro de texto “Select default path to look for scripts”, especifique la carpeta /home/oracle/labs/plpu. Esta carpeta contiene los scripts de soluciones, los scripts de ejemplos de código y cualquier ejercicio práctico o demostración que se utilicen en este curso.
- 4) Haga clic en OK para aceptar los cambios y salir de la ventana Preferences.
- 5) Familiarícese con la carpeta /home/oracle/labs/plpu.
 - a) Haga clic en el separador **Files** (junto al separador **Connections**).
 - b) Acceda a la carpeta /home/oracle/labs/plpu. ¿Cuántas subcarpetas ve en la carpeta labs?
 - c) Desplácese por las carpetas y abra un archivo de script sin ejecutar el código.
 - d) Borre el código que aparece en el área SQL Worksheet.

Práctica I-5: Acceso a la Biblioteca de Documentación en Línea de Oracle Database 11g Versión 2

En esta práctica, accederá y marcará algunas referencias a la documentación de Oracle Database 11g Versión 2 que va a utilizar en este curso.

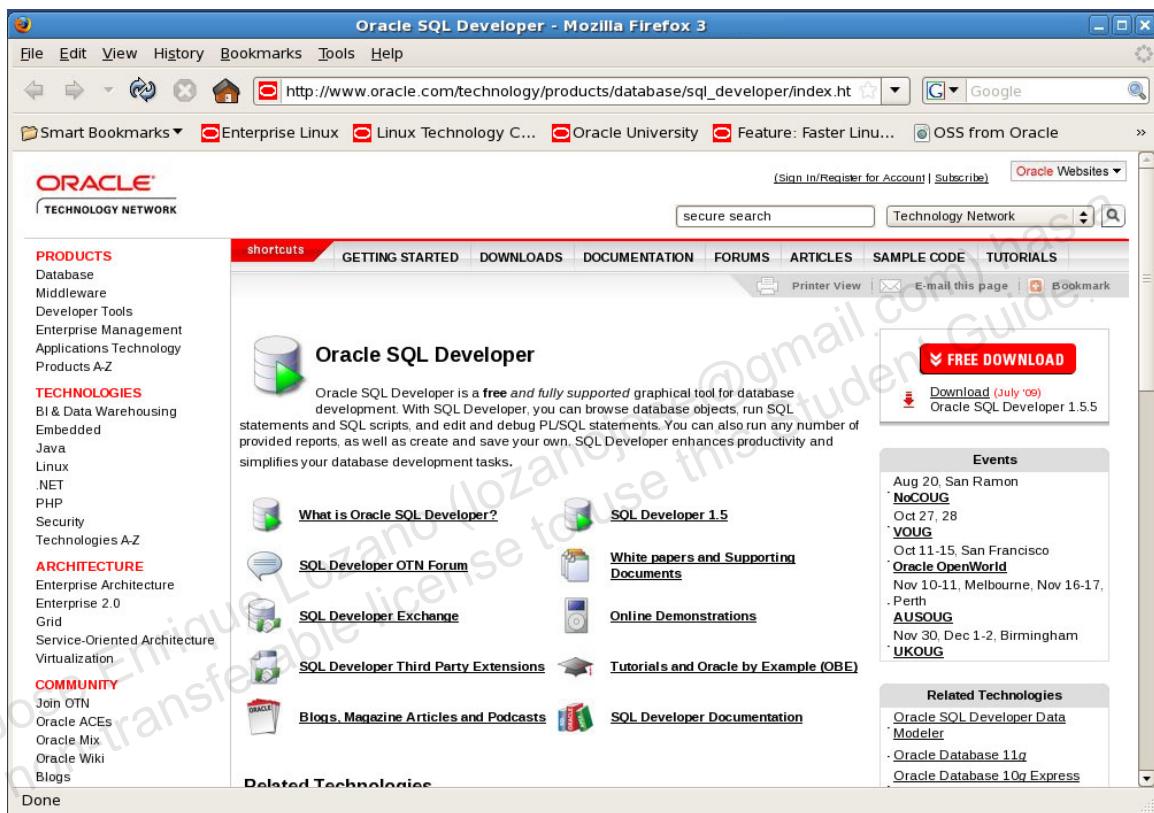
- 1) Acceda a la página web de documentación de Oracle Database 11g Versión 2 en:
<http://www.oracle.com/pls/db111/homepage>.
- 2) Marque la página para acceder más fácilmente a ella en el futuro.
- 3) Vea la lista completa de libros disponibles para Oracle Database 11g Versión 2.
- 4) Anote las siguientes referencias de documentación que va a utilizar en este curso, según sea necesario:
 - a) *Advanced Application Developer's Guide*
 - b) *New Features Guide*
 - c) *PL/SQL Language Reference*
 - d) *Oracle Database Reference*
 - e) *Oracle Database Concepts*
 - f) *SQL Developer User's Guide*
 - g) *SQL Language Reference Guide*
 - h) *SQL*Plus User's Guide and Reference*

Soluciones a la Práctica I-1: Identificación de los Recursos Disponibles de SQL Developer

En esta práctica, revisará los recursos disponibles de SQL Developer.

- 1) Familiarícese con Oracle SQL Developer según sea necesario mediante el Apéndice C: Uso de SQL Developer.
- 2) Acceda a la página inicial de SQL Developer disponible en línea en la dirección: http://www.oracle.com/technology/products/database/sql_developer/index.html

La página inicial de SQL Developer se muestra de la siguiente forma:



- 3) Marque la página para acceder más fácilmente a ella en el futuro.

No existe una única solución. El enlace se agrega a la barra de herramientas Links, de la siguiente forma:

- 4) Acceda al tutorial de SQL Developer disponible en línea en la dirección: <http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>

Acceda al tutorial de SQL Developer con la dirección URL anterior. Aparece la siguiente página:

Soluciones a la Práctica I-1: Identificación de los Recursos Disponibles de SQL Developer (continuación)

The screenshot shows the Oracle SQL Developer Tutorial website. The left sidebar has a tree view of topics. The 'Accessing Data' section is expanded and highlighted with a red box. The main content area is titled 'Welcome to the Oracle SQL Developer Tutorial!' and contains the following text:

This tutorial prepares a developer to use Oracle SQL Developer to perform common database development tasks. This tutorial was developed using Oracle SQL Developer 1.5.4.

Learning Objectives

After completing this tutorial, you should be able to:

- Install Oracle SQL Developer
- Create a Database Connection
- Manage Database Objects
- Access and Manipulate Data
- Add, edit, and debug PL/SQL Components

[!\[\]\(bf80706fd45b2e83a49a353977575631_img.jpg\) Start Tutorial](#) [!\[\]\(42809253693285004295f1400ff05296_img.jpg\) Download Tutorial](#)

Last updated: May 11, 2009

- 5) Realice una vista previa y pruebe con los enlaces y las demostraciones disponibles del tutorial todo lo necesario, sobre todo con los enlaces “Creación de una Conexión a la Base de Datos” y “Acceso a los Datos”.

Para revisar la sección sobre la creación de una conexión de base de datos, haga clic en el signo más “+” situado junto al enlace “Pasos Iniciales” para que aparezca el enlace “Creación de una Conexión a la Base Datos”. Para revisar el tema Creación de una Conexión a la Base de Datos, haga clic en el enlace del tema. Para revisar la sección sobre el acceso a los datos, haga clic en el signo más “+” situado junto al enlace “Acceso a los Datos” para que aparezca la lista de temas disponibles. Para revisar cualquiera de los temas, haga clic en el enlace del tema.

Soluciones a la Práctica I-2: Creación y Uso de una Nueva Conexión de Base de Datos de SQL Developer

En esta práctica, iniciará SQL Developer mediante la información de conexión y creará una nueva conexión de base de datos.

- 1) Inicie SQL Developer mediante el identificador y la contraseña de usuario que le ha proporcionado el instructor, por ejemplo ora61.

Haga clic en el ícono SQL Developer del escritorio.

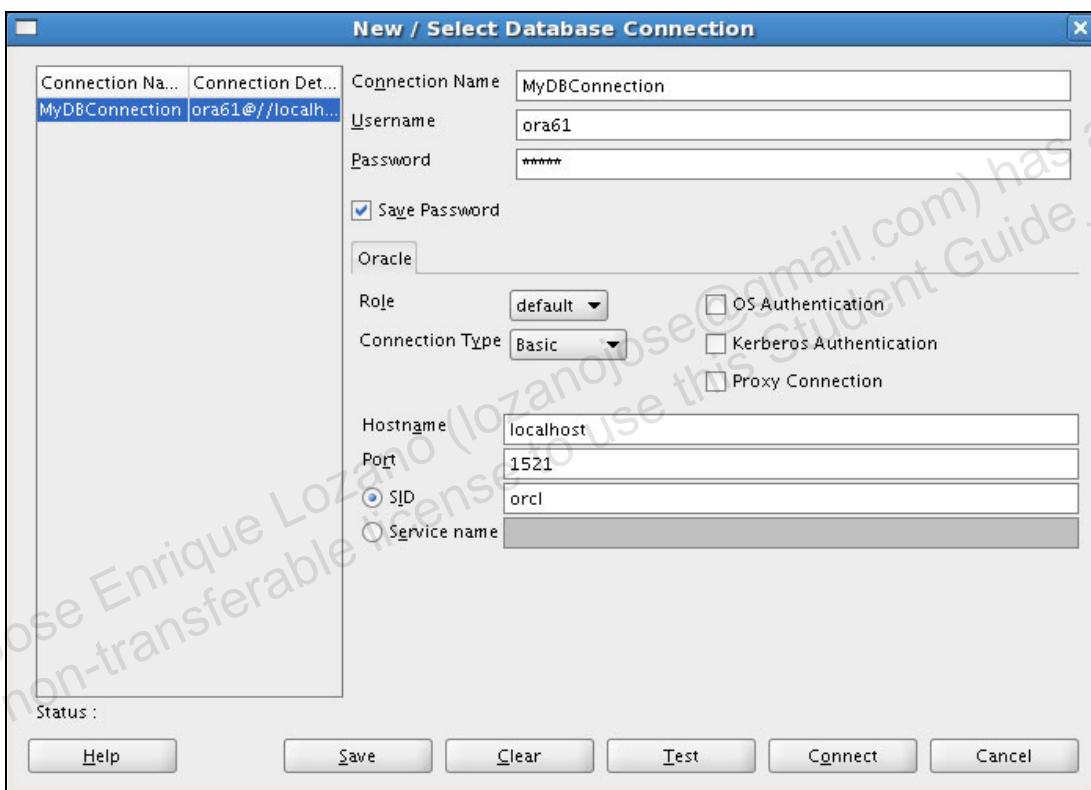


- 2) Cree una conexión de base de datos con la siguiente información:
 - a) Connection Name: MyDBConnection
 - b) Username: ora61
 - c) Password: ora61
 - d) Hostname: introduzca el nombre de host de la computadora
 - e) Port: 1521
 - f) SID: ORCL

Soluciones a la Práctica I-2: Creación y Uso de una Nueva Conexión de Base de Datos de SQL Developer (continuación)

Haga clic con el botón derecho en el ícono Connections de la página con separadores Connections y, a continuación, seleccione la opción New Connection en el menú de acceso directo. Se muestra la ventana New>Select Database Connection. Utilice la información que se ha proporcionado anteriormente para crear la nueva conexión de base de datos.

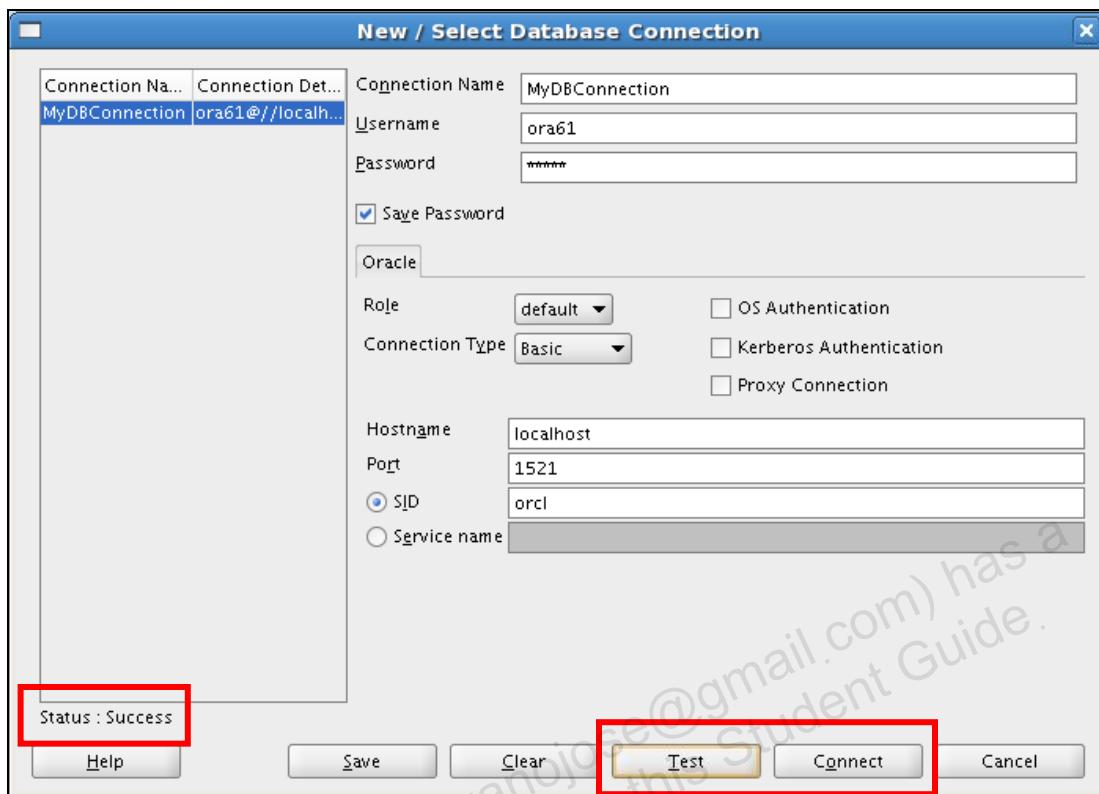
Nota: para mostrar las propiedades de la conexión que se acaba de crear, haga clic con el botón derecho en el nombre de la conexión y, a continuación, seleccione Properties en el menú de acceso directo. Sustituya el nombre de usuario, la contraseña, el nombre de host y el nombre de servicio por la información correspondiente que le haya proporcionado su instructor. A continuación se muestra un ejemplo de la conexión de base de datos que se acaba de crear para el estudiante ora61:



- 3) Pruebe la nueva conexión. Si el estado es Success, conecte a la base de datos con esta nueva conexión:
 - a) Haga clic dos veces en el ícono MyDBConnection de la página con separadores Connections.
 - b) Haga clic en el botón Test de la ventana New>Select Database Connection. Si el estado es Success, haga clic en el botón Connect.

Soluciones a la Práctica I-2: Creación y Uso de una Nueva Conexión de Base de Datos de SQL Developer (continuación)

Unauthorized reproduction or distribution prohibited. Copyright© 2013, Oracle and/or its affiliates.



Soluciones a la Práctica I-3: Exploración de las Tablas de Esquema y Creación y Ejecución de un Bloque Anónimo Simple

En esta práctica, examinará las tablas de esquema y creará y ejecutará un bloque anónimo simple.

- 1) Examine la estructura de la tabla EMPLOYEES y muestre sus datos.
 - a) Amplíe la conexión MyDBConnection. Para ello, haga clic en el signo más situado junto a ella.
 - b) Amplíe el ícono Tables. Para ello, haga clic en el signo más situado junto a él.
 - c) Visualice la estructura de la tabla EMPLOYEES.

Haga clic dos veces en la tabla EMPLOYEES. En el separador Columns se muestran las columnas de la tabla EMPLOYEES, de la siguiente forma:

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
EMPLOYEE_ID	NUMBER(6,0)	No	(null)	1	1	Primary key of employees table.
FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2	(null)	First name of the employee. A not null column.
LAST_NAME	VARCHAR2(25 BYTE)	No	(null)	3	(null)	Last name of the employee. A not null column.
EMAIL	VARCHAR2(25 BYTE)	No	(null)	4	(null)	Email id of the employee
PHONE_NUMBER	VARCHAR2(20 BYTE)	Yes	(null)	5	(null)	Phone number of the employee; includes country code and area code.
HIRE_DATE	DATE	No	(null)	6	(null)	Date when the employee started on this job. A not null column.
JOB_ID	VARCHAR2(10 BYTE)	No	(null)	7	(null)	Current job of the employee; foreign key to job_id column.
SALARY	NUMBER(8,2)	Yes	(null)	8	(null)	Monthly salary of the employee. Must be greater than zero.
COMMISSION_PCT	NUMBER(2,2)	Yes	(null)	9	(null)	Commission percentage of the employee; Only employees with commission_pct > 0 have non-null values in this column.
MANAGER_ID	NUMBER(6,0)	Yes	(null)	10	(null)	Manager id of the employee; has same domain as employee_id column.
DEPARTMENT_ID	NUMBER(4,0)	Yes	(null)	11	(null)	Department id where employee works; foreign key to department_id column.

- 2) Examine la tabla EMPLOYEES y muestre sus datos.

Para que aparezcan los datos de los empleados, haga clic en el separador Data. Aparecen los datos de la tabla EMPLOYEES, como se muestra a continuación:

Soluciones a la Práctica I-3: Exploración de las Tablas de Esquema y Creación y Ejecución de un Bloque Anónimo Simple (continuación)

The screenshot shows the Oracle SQL Developer interface with the title bar "Oracle SQL Developer : TABLE ORA61.EMPLOYEES@MyDBConnection". The left sidebar shows the database structure under "MyDBConnection / Tables / EMPLOYEES". The main pane displays the "EMPLOYEES" table with columns: EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, SALARY, and COMMISSION_PCT. The table contains 107 rows of employee data. At the bottom left of the main pane, a red box highlights the status bar which reads "All Rows Fetched: 107".

- 3) Utilice la hoja de trabajo de SQL para seleccionar los apellidos y los salarios de todos los empleados cuyo salario anual sea mayor de 10.000 dólares. Utilice tanto el icono Execute Statement (F9) como el icono Run Script (F5) para ejecutar la sentencia SELECT. Revise los resultados de ambos métodos de ejecución de las sentencias SELECT en los separadores adecuados.

Nota: dedique unos minutos a familiarizarse con los datos o a consultar el Apéndice B, donde se proporciona la descripción y los datos de todas las tablas del esquema HR que utilizará en este curso.

Para mostrar la hoja de trabajo de SQL, utilice uno de los dos métodos siguientes:

1. Seleccione Tools > SQL Worksheet o haga clic en el ícono Open SQL Worksheet. Aparece la ventana Select Connection.
2. Seleccione el nuevo MyDBConnection en la lista desplegable Connection (si no está seleccionado) y, a continuación, haga clic en OK.

Abra el archivo **sol_I_03.sql** en la carpeta **/home/oracle/labs/plpu/solns**, como se detalla a continuación con uno de los dos métodos siguientes:

1. En el separador Files, seleccione (o acceda al) archivo de script que desea abrir.

Soluciones a la Práctica I-3: Exploración de las Tablas de Esquema y Creación y Ejecución de un Bloque Anónimo Simple (continuación)

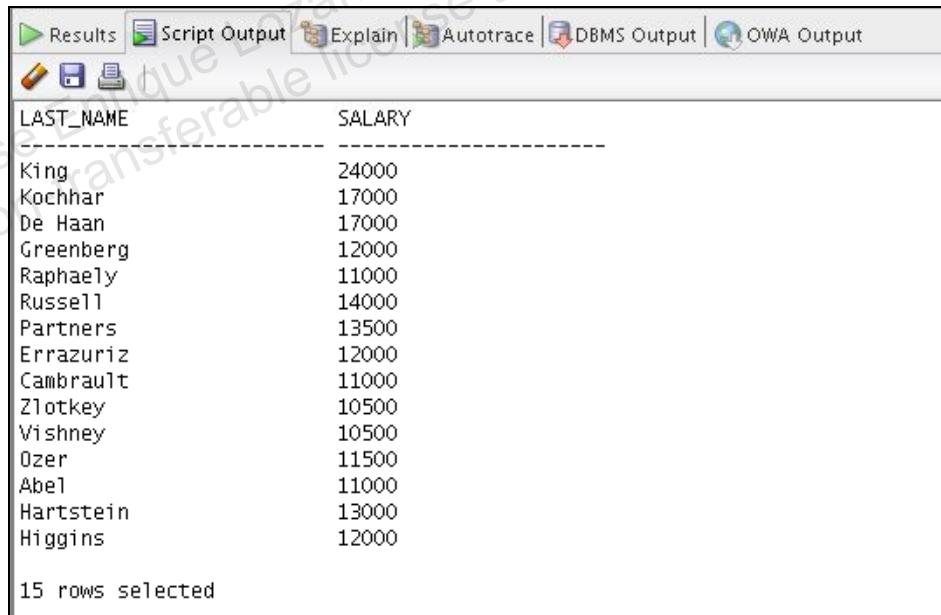
2. Haga clic dos veces en el nombre del archivo para abrirlo. El código del archivo de script se muestra en el área SQL Worksheet.
3. Para ejecutar el código, haga clic en el icono Run Script (F5) en la barra de herramientas de SQL Worksheet.

También puede:

1. Seleccionar Open en el menú File. Aparece el cuadro de diálogo Open.
2. En el cuadro de diálogo Open, seleccionar (o acceder al) archivo de script que desea abrir.
3. Hacer clic en Open. El código del archivo de script se muestra en el área SQL Worksheet.
4. Para ejecutar el código, haga clic en el icono Run Script (F5) en la barra de herramientas de SQL Worksheet.

Para ejecutar una sola sentencia SELECT, haga clic en el icono Execute Statement (F9) (asegurándose de que el cursor esté en cualquiera de las líneas de la sentencia SELECT) en la barra de herramientas de SQL Worksheet para ejecutar la sentencia. El código y el resultado se muestran de la siguiente forma:

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE SALARY > 10000;
```



The screenshot shows the Oracle SQL Worksheet interface. At the top, there is a toolbar with several icons: Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. Below the toolbar, the results of a query are displayed in a grid. The grid has two columns: LAST_NAME and SALARY. The data consists of 15 rows, each showing a last name and its corresponding salary. At the bottom of the grid, it says "15 rows selected".

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Greenberg	12000
Raphaely	11000
Russell	14000
Partners	13500
Errazuriz	12000
Cambrault	11000
Zlotkey	10500
Vishney	10500
Ozer	11500
Abel	11000
Hartstein	13000
Higgins	12000

15 rows selected

- 4) Cree y ejecute un bloque anónimo simple cuya salida sea “Hello World”.
 - a) Active SET SERVEROUTPUT ON para mostrar la salida de las sentencias del paquete DBMS_OUTPUT.

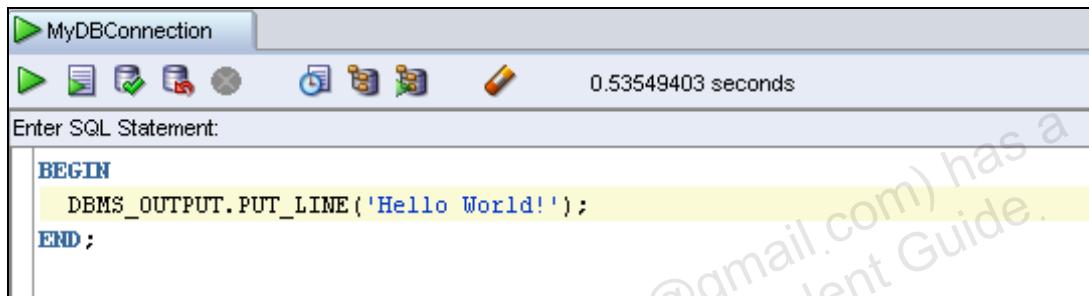
Soluciones a la Práctica I-3: Exploración de las Tablas de Esquema y Creación y Ejecución de un Bloque Anónimo Simple (continuación)

Introduzca el siguiente comando en el área SQL Worksheet y, a continuación, haga clic en el ícono Run Script (F5).

```
SET SERVEROUTPUT ON
```

- b) Utilice el área SQL Worksheet para introducir el código del bloque anónimo.

Introduzca el código siguiente en el área SQL Worksheet, como se muestra a continuación. También puede abrir el archivo **sol_I_04.sql** en la carpeta **/home/oracle/labs/plpu/solns**. El código se muestra de la siguiente forma:

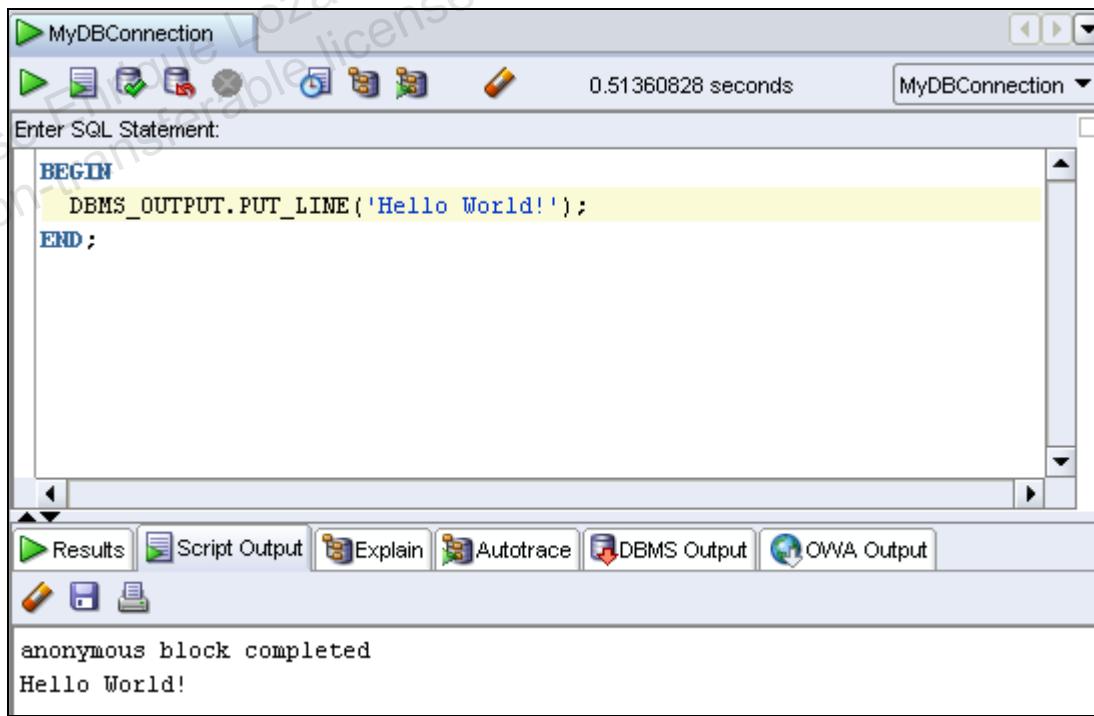


The screenshot shows the Oracle SQL Developer interface. The title bar says "MyDBConnection". Below it is a toolbar with various icons. The main area is labeled "Enter SQL Statement:" and contains the following PL/SQL code:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('Hello World!');
END;
```

- c) Haga clic en el ícono Run Script (F5) para ejecutar el bloque anónimo.

En el separador Script Output, se muestra la salida del bloque anónimo, de la siguiente forma:

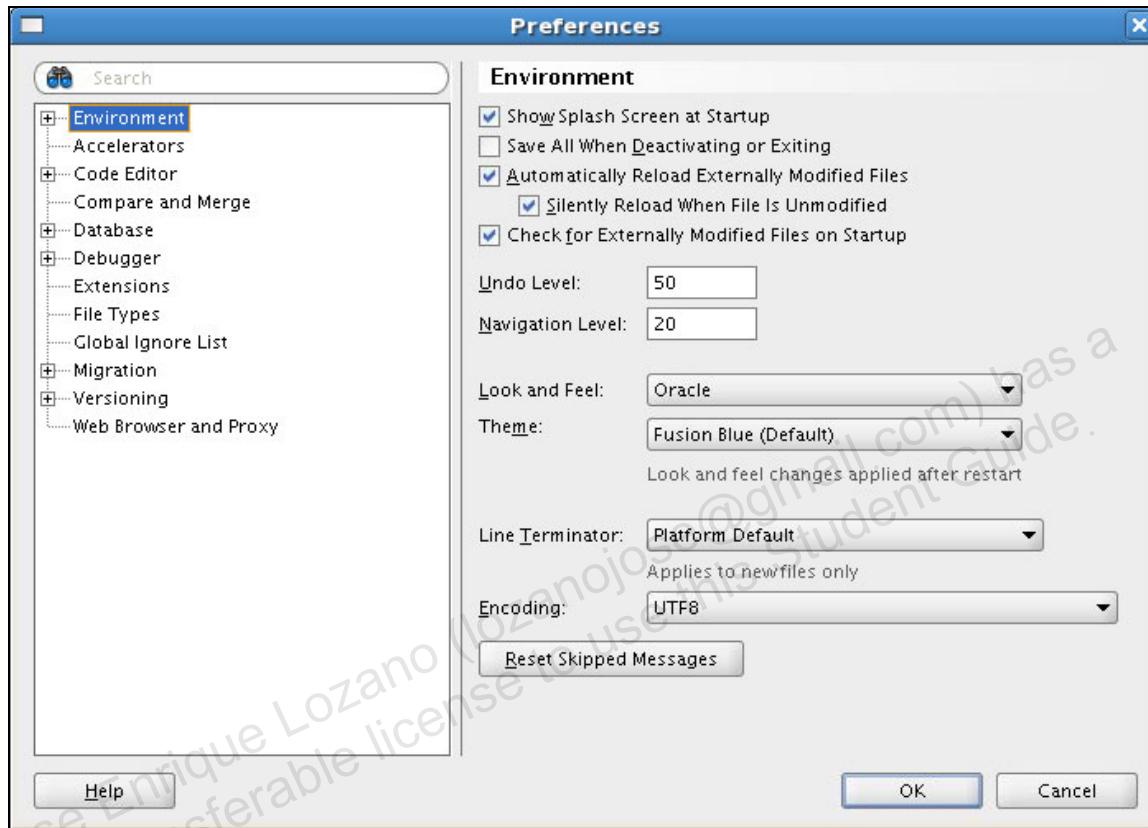


The screenshot shows the Oracle SQL Developer interface with the "Script Output" tab selected. At the bottom of the screen, there is a status message: "anonymous block completed". Above it, the output shows the text "Hello World!".

Soluciones a la Práctica I-4: Definición de Algunas Preferencias de SQL Developer

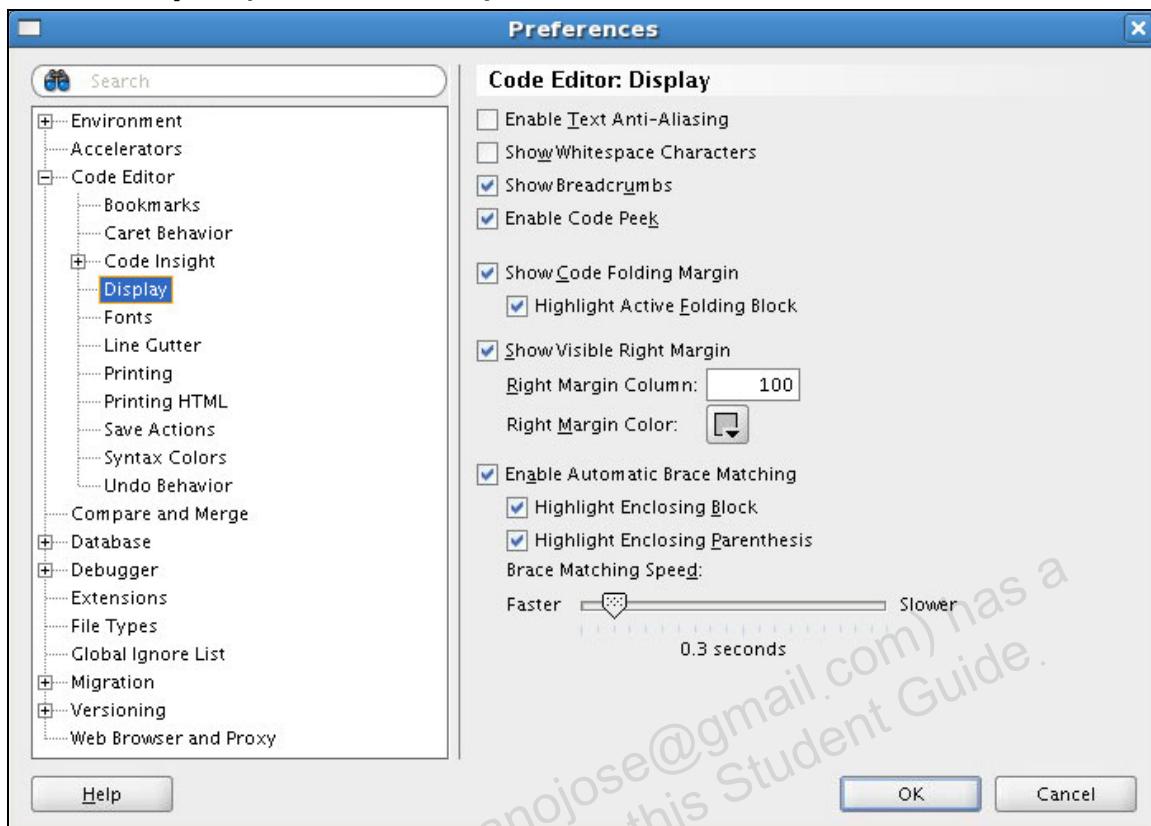
En esta práctica, definirá algunas preferencias de SQL Developer.

- 1) En el menú SQL Developer, acceda a Tools > Preferences. Se mostrará la ventana Preferences.



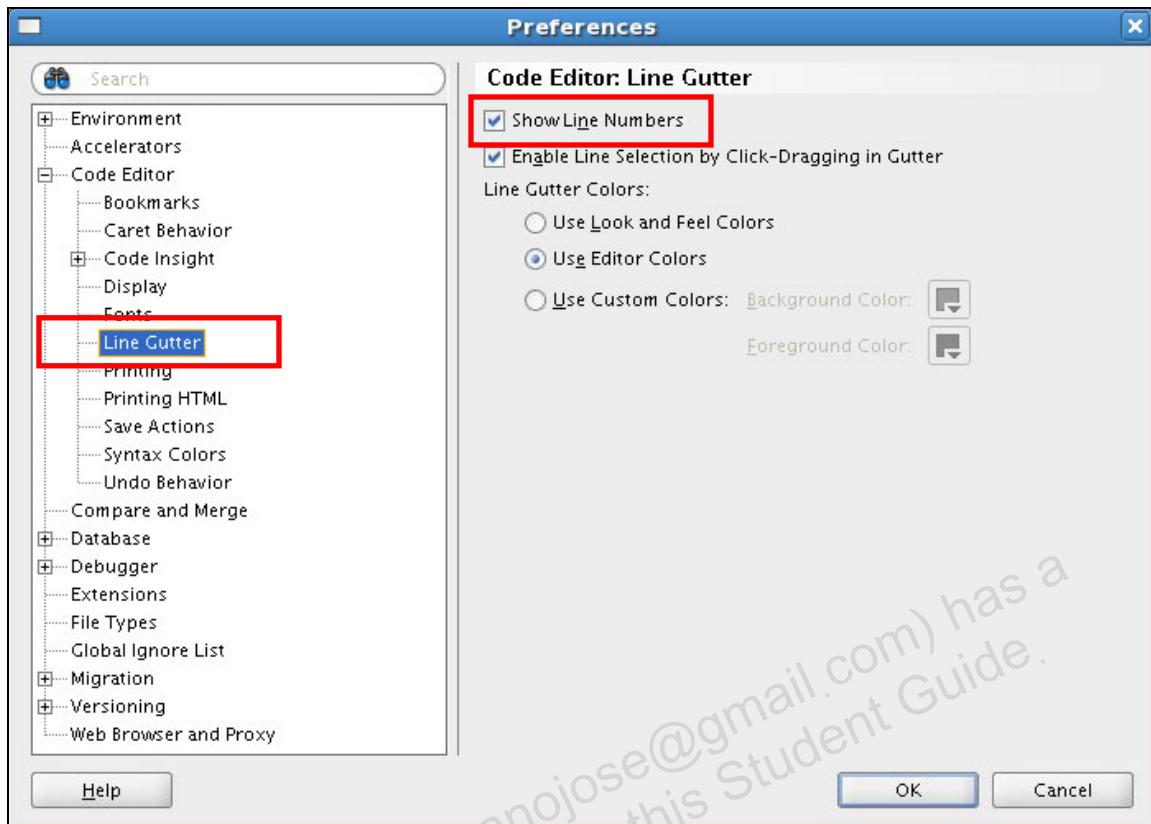
- 2) Amplíe la opción Code Editor y, a continuación, haga clic en la opción Display para que aparezca la sección “Code Editor: Display”. Esta sección contiene opciones generales de la apariencia y el comportamiento del editor de códigos.
 - a) Introduzca 100 en el cuadro de texto Right Margin Column de la sección Show Visible Right Margin. Esto hace que aparezca un margen derecho que puede definir para controlar la longitud de las líneas de código.

Soluciones a la Práctica I-4: Definición de Algunas Preferencias de SQL Developer (continuación)



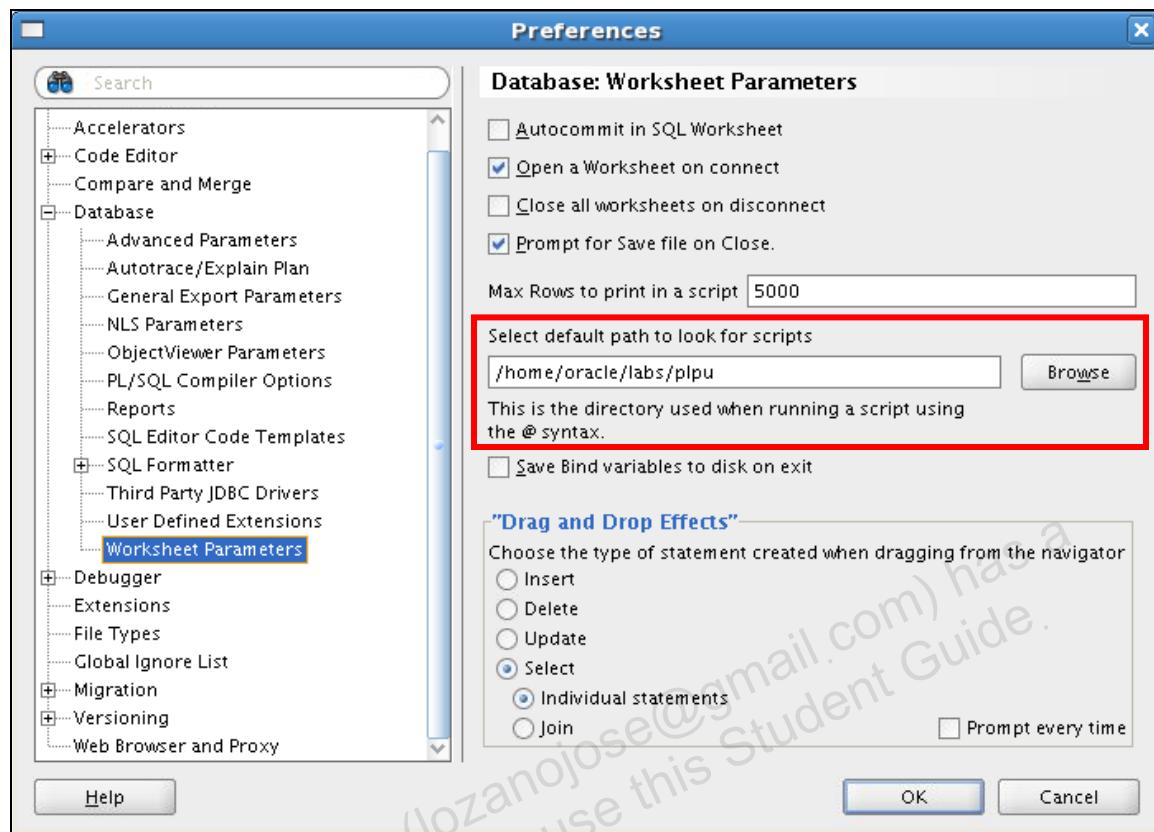
- b) Haga clic en la opción Line Gutter. Esta opción especifica opciones para el canal de línea (margen izquierdo del editor de códigos). Seleccione la casilla de control Show Line Numbers para mostrar los números de línea de código.

Soluciones a la Práctica I-4: Definición de Algunas Preferencias de SQL Developer (continuación)



- 3) Haga clic en la opción Worksheet Parameters de Database. En el cuadro de texto “Select default path to look for scripts”, especifique la carpeta /home/oracle/labs/plpu. Esta carpeta contiene los scripts de soluciones, los scripts de ejemplos de código y cualquier ejercicio práctico o demostración que se utilicen en este curso.

Soluciones a la Práctica I-4: Definición de Algunas Preferencias de SQL Developer (continuación)



- 4) Haga clic en OK para aceptar los cambios y salir de la ventana Preferences.
- 5) Familiarícese con la carpeta labs en la carpeta /home/oracle/labs/plpu.
 - a) Haga clic en el separador **Files** (junto al separador **Connections**).
 - b) Acceda a la carpeta /home/oracle/labs/plpu.
 - c) ¿Cuántas subcarpetas ve en la carpeta labs?
 - d) Desplácese por las carpetas y abra un archivo de script sin ejecutar el código.
 - e) Borre el código que aparece en el área SQL Worksheet. En el menú SQL Developer, acceda a Tools > Preferences.

Soluciones a la Práctica I-5: Acceso a la Biblioteca de Documentación en Línea de Oracle Database 11g Versión 2

En esta práctica, accederá y marcará algunas referencias a la documentación de Oracle Database 11g Versión 2 que va a utilizar en este curso.

- 1) Acceda a la página web de documentación de Oracle Database 11g Versión 2 en:
<http://www.oracle.com/pls/db111/homepage>
- 2) Marque la página para acceder más fácilmente a ella en el futuro.
- 3) Vea la lista completa de libros disponibles para Oracle Database 11g Versión 2.
- 4) Anote las siguientes referencias de documentación que va a utilizar en este curso, según sea necesario:
 - a) *Advanced Application Developer's Guide*
 - b) *New Features Guide*
 - c) *PL/SQL Language Reference*
 - d) *Oracle Database Reference*
 - e) *Oracle Database Concepts*
 - f) *SQL Developer User's Guide*
 - g) *SQL Language Reference Guide*
 - h) *SQL*Plus User's Guide and Reference*

Prácticas y Soluciones de la Lección 1

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

Nota: si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

Práctica 1-1: Creación, Compilación y Llamada de Procedimientos

En esta práctica, creará y llamará al procedimiento ADD_JOB, además de revisar el resultado. También creará y llamará a un procedimiento denominado UPD_JOB para modificar un trabajo en la tabla JOBS; asimismo, creará y llamará a un procedimiento denominado DEL_JOB para suprimir un trabajo de la tabla JOBS. Por último, creará 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.

- 1) Cree, compile y llame al procedimiento ADD_JOB y revise el resultado.
 - a) Cree un procedimiento denominado ADD_JOB para insertar un nuevo trabajo en la tabla JOBS. Proporcione el identificador y el cargo utilizando dos parámetros.

Nota: puede crear el procedimiento (así como otros objetos) mediante la introducción del código en el área SQL Worksheet y, a continuación, hacer clic en el icono Run Script (F5). De esta forma, se crea y compila el procedimiento. Para saber si el procedimiento contiene o no errores, haga clic en el nombre del mismo en el nodo de procedimientos y, a continuación, seleccione Compile en el menú emergente.

- b) Llame al procedimiento con IT_DBA como identificador de trabajo y Database Administrator como cargo. Consulte la tabla JOBS y vea el resultado.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Database Administrator		
1 rows selected			

- c) Llame al procedimiento de nuevo y transfiera un identificador de trabajo ST_MAN y un cargo Stock Manager. ¿Qué sucede? ¿Por qué?
- 2) Cree un procedimiento denominado UPD_JOB para modificar un trabajo en la tabla JOBS.
 - a) Cree un procedimiento denominado UPD_JOB para actualizar el cargo. Proporcione el identificador de trabajo y un cargo nuevo utilizando dos parámetros. Incluya el manejo de excepciones necesario si no se ha producido la actualización.
 - b) Llame al procedimiento para cambiar el cargo del identificador de trabajo IT_DBA a Data Administrator. Consulte la tabla JOBS y vea el resultado.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Data Administrator		
1 rows selected			

- c) Pruebe la sección de manejo de excepciones del procedimiento intentando actualizar un trabajo que no existe. Puede utilizar el identificador de trabajo IT_WEB y el cargo Web Master.

Práctica 1-1: Creación, Compilación y Llamada de Procedimientos (continuación)

```
Error starting at line 1 in command:
EXECUTE upd_job ('IT_WEB', 'Web Master')
Error report:
ORA-20202: No job updated.
ORA-06512: at "ORA80.UPD_JOB", line 9
ORA-06512: at line 1
```

- 3) Cree un procedimiento denominado `DEL_JOB` para suprimir un trabajo de la tabla `JOBS`.

- Cree un procedimiento denominado `DEL_JOB` para suprimir un trabajo. Incluya el código de manejo de excepciones necesario si no se ha suprimido ningún trabajo.
- Llame al procedimiento mediante el identificador de trabajo `IT_DBA`. Consulte la tabla `JOBS` y vea el resultado.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
<hr/> 0 rows selected			

- Pruebe la sección de manejo de excepciones del procedimiento intentando suprimir un trabajo que no existe. Utilice `IT_WEB` como identificador de trabajo. Aparecerá el mensaje que haya incluido en la sección de manejo de excepciones del procedimiento como salida.

```
Error starting at line 1 in command:
EXECUTE del_job ('IT_WEB')
Error report:
ORA-20203: No jobs deleted.
ORA-06512: at "ORA80.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.

- Cree un procedimiento que devuelva un valor de las columnas `SALARY` y `JOB_ID` para el identificador de empleado especificado. Elimine los errores de sintaxis, si los hay y, a continuación, recompile el código.
- 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.

```
v_salary
-----
8000

v_job
-----
ST_MAN
```

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

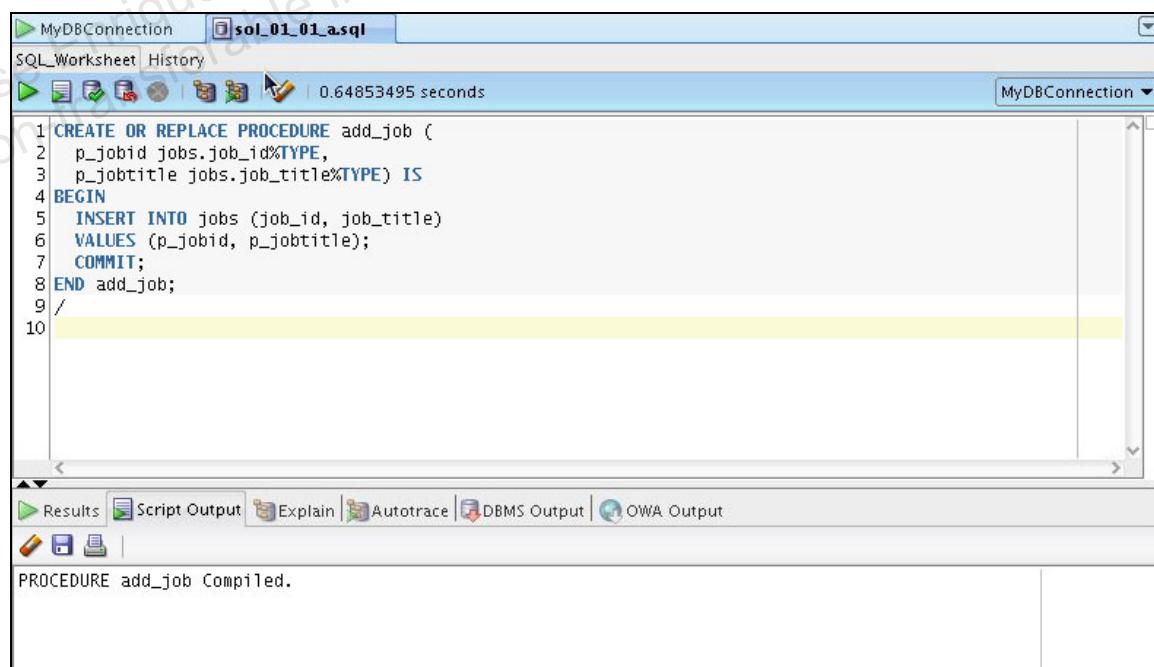
Soluciones a la Práctica 1-1: Creación, Compilación y Llamada de Procedimientos

En esta práctica, creará y llamará al procedimiento ADD_JOB, además de revisar el resultado. También creará y llamará a un procedimiento denominado UPD_JOB para modificar un trabajo en la tabla JOBS; asimismo, creará y llamará a un procedimiento denominado DEL_JOB para suprimir un trabajo de la tabla JOBS. Por último, creará 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.

- 1) Cree, compile y llame al procedimiento ADD_JOB y revise el resultado.
 - a) Cree un procedimiento denominado ADD_JOB para insertar un nuevo trabajo en la tabla JOBS. Proporcione el identificador y el cargo utilizando dos parámetros.

Nota: puede crear el procedimiento (así como otros objetos) mediante la introducción del código en el área SQL Worksheet y, a continuación, hacer clic en el icono Run Script (F5). De esta forma, se crea y compila el procedimiento. Si el procedimiento genera un mensaje de error al crearlo, haga clic en el nombre del mismo en el nodo de procedimientos, edite el procedimiento y, a continuación, seleccione Compile en el menú emergente.

Abra el archivo **sol_01_01_a.sql** en la carpeta **/home/oracle/labs/plpu/solns**. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar el procedimiento. El código y el resultado se muestran de la siguiente forma:



The screenshot shows the Oracle SQL Worksheet interface. The title bar says "MyDBConnection" and the tab is "sol_01_01_a.sql". The main area contains the following PL/SQL code:

```

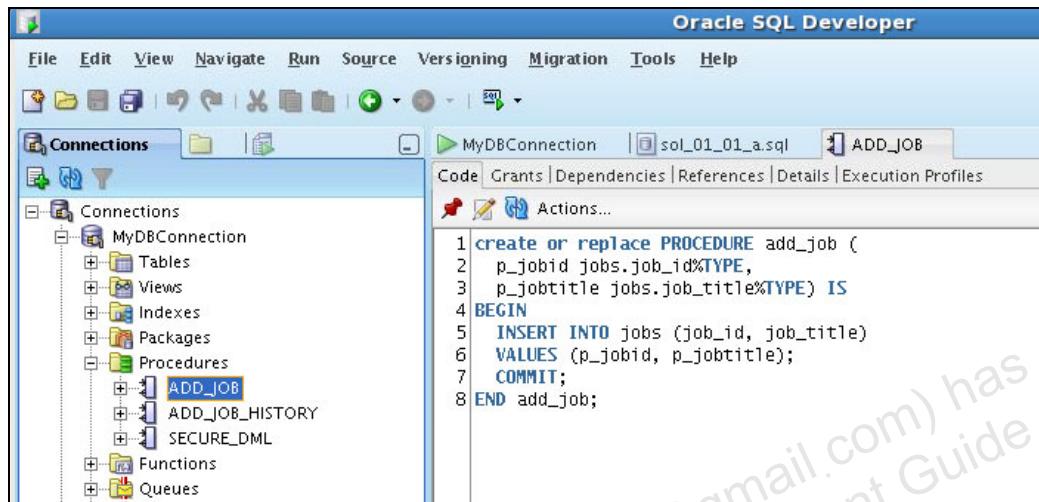
1 CREATE OR REPLACE PROCEDURE add_job (
2   p_jobid jobs.job_id%TYPE,
3   p_jobtitle jobs.job_title%TYPE) IS
4 BEGIN
5   INSERT INTO jobs (job_id, job_title)
6   VALUES (p_jobid, p_jobtitle);
7   COMMIT;
8 END add_job;
9 /
10

```

Below the code, the status bar shows "0.64853495 seconds". At the bottom, the "Results" tab is selected, showing the message "PROCEDURE add_job Compiled."

Soluciones a la Práctica 1-1: Creación, Compilación y Llamada de Procedimientos (continuación)

Para ver el procedimiento que acaba de crear, haga clic en el nodo Procedures de Object Navigator. Si no aparece el procedimiento que se acaba de crear, haga clic con el botón derecho en el nodo Procedures y, a continuación, seleccione Refresh en el menú de acceso directo. El nuevo procedimiento se muestra de la siguiente forma:



- b) Llame al procedimiento con IT_DBA como identificador de trabajo y Database Administrator como cargo. Consulte la tabla JOBS y vea el resultado.

Ejecute el script /home/oracle/labs/plpu/soln/sol_01_01_b.sql. El código y el resultado se muestran de la siguiente forma:

```

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

```

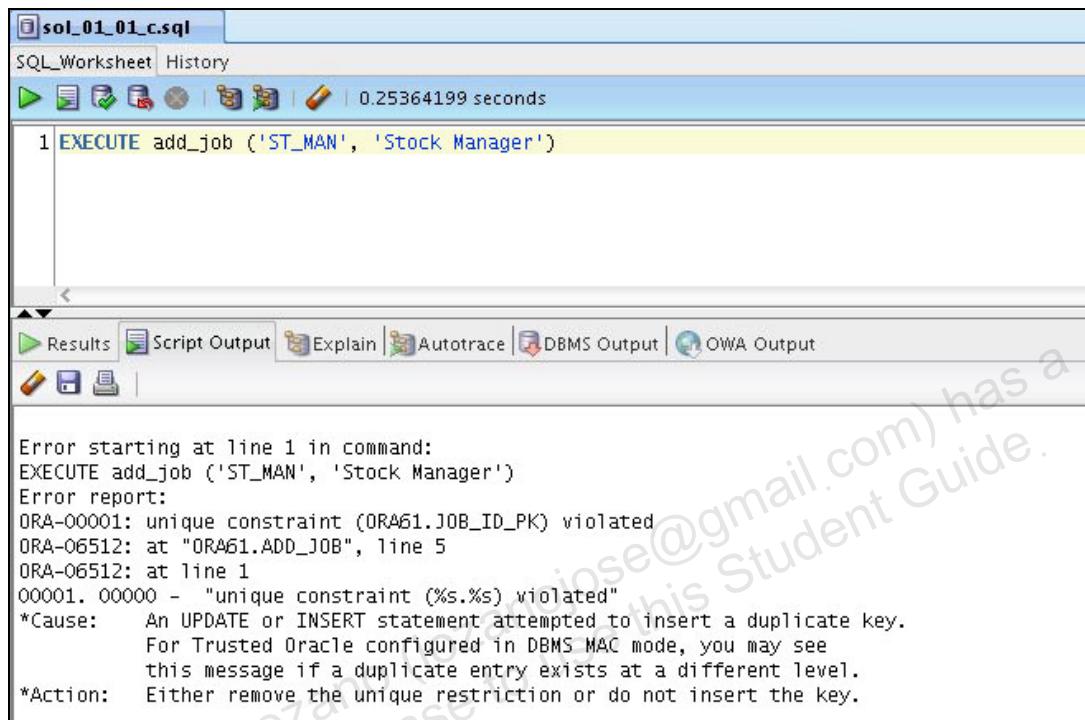
JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Database Administrator		

1 rows selected

Soluciones a la Práctica 1-1: Creación, Compilación y Llamada de Procedimientos (continuación)

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

Se ha producido una excepción porque hay una restricción de integridad de clave única en la columna JOB_ID.



The screenshot shows the Oracle SQL Developer interface. In the SQL Worksheet tab, a single line of code is highlighted in yellow: `EXECUTE add_job ('ST_MAN', 'Stock Manager');`. Below the worksheet, the Error window displays the following message:

```
Error starting at line 1 in command:
EXECUTE add_job ('ST_MAN', 'Stock Manager')
Error report:
ORA-00001: unique constraint (ORA61.JOB_ID_PK) violated
ORA-06512: at "ORA61.ADD_JOB", line 5
ORA-06512: at line 1
00000 - "unique constraint (%s.%s) violated"
*Cause: An UPDATE or INSERT statement attempted to insert a duplicate key.
        For Trusted Oracle configured in DBMS MAC mode, you may see
        this message if a duplicate entry exists at a different level.
*Action: Either remove the unique restriction or do not insert the key.
```

- 2) Cree un procedimiento denominado UPD_JOB para modificar un trabajo en la tabla JOBS.
- Cree un procedimiento denominado UPD_JOB para actualizar el cargo. Proporcione el identificador de trabajo y un cargo nuevo utilizando dos parámetros. Incluya el manejo de excepciones necesario si no se ha producido la actualización.
- Ejecute el script `/home/oracle/labs/plpu/soln/sol_01_02_a.sql`. El código y el resultado se muestran de la siguiente forma:

Soluciones a la Práctica 1-1: Creación, Compilación y Llamada de Procedimientos (continuación)

The screenshot shows the SQL Worksheet interface in Oracle SQL Developer. The code area contains the following PL/SQL procedure:

```

1 CREATE OR REPLACE PROCEDURE upd_job(
2   p_jobid IN jobs.job_id%TYPE,
3   p_jobtitle IN jobs.job_title%TYPE) IS
4 BEGIN
5   UPDATE jobs
6     SET job_title = p_jobtitle
7   WHERE job_id = p_jobid;
8   IF SQL%NOTFOUND THEN
9     RAISE_APPLICATION_ERROR(-20202, 'No job updated.');
10  END IF;
11 END upd_job;
12 /
13

```

The results pane at the bottom shows the message: "PROCEDURE upd_job(Compiled."

- b) Llame al procedimiento para cambiar el cargo del identificador de trabajo IT_DBA a Data Administrator. Consulte la tabla JOBS y vea el resultado.
Ejecute el script /home/oracle/labs/plpu/soln/sol_01_02_b.sql.
El código y el resultado se muestran de la siguiente forma:

The screenshot shows the SQL Worksheet interface in Oracle SQL Developer. The code area contains the following PL/SQL block:

```

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

```

The results pane at the bottom shows the output of the SELECT query:

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Data Administrator		

Below the table, it says "1 rows selected".

Soluciones a la Práctica 1-1: Creación, Compilación y Llamada de Procedimientos (continuación)

- c) Pruebe la sección de manejo de excepciones del procedimiento intentando actualizar un trabajo que no existe. Puede utilizar el identificador de trabajo IT_WEB y el cargo Web Master.

```

sol_01_02_C.sql
SQL Worksheet History
EXECUTE upd_job ('IT_WEB', 'Web Master')
SELECT * FROM jobs WHERE job_id = 'IT_WEB';

Error starting at line 1 in command:
EXECUTE upd_job ('IT_WEB', 'Web Master')
Error report:
ORA-20202: No job updated.
ORA-06512: at "ORA61.UPD_JOB", line 9
ORA-06512: at line 1

JOB_ID      JOB_TITLE          MIN_SALARY      MAX_SALARY
-----      -----
0 rows selected

```

- 3) Cree un procedimiento denominado DEL_JOB para suprimir un trabajo de la tabla JOBS.
- Cree un procedimiento denominado DEL_JOB para suprimir un trabajo. Incluya el código de manejo de excepciones necesario si no se ha suprimido ningún trabajo.
- Ejecute el script /home/oracle/labs/plpu/soln/sol_01_03_a.sql. El código y el resultado se muestran de la siguiente forma:

Soluciones a la Práctica 1-1: Creación, Compilación y Llamada de Procedimientos (continuación)

The screenshot shows the Oracle SQL Worksheet interface. The title bar says "sol_01_03_a.sql". The main area contains the following PL/SQL code:

```
1 CREATE OR REPLACE PROCEDURE del_job (p_jobid jobs.job_id%TYPE) IS
2 BEGIN
3   DELETE FROM jobs
4   WHERE job_id = p_jobid;
5   IF SQL%NOTFOUND THEN
6     RAISE_APPLICATION_ERROR(-20203, 'No jobs deleted.');
7   END IF;
8 END del_job;
9 /
```

Below the code, the status bar indicates "0.20295501 seconds". At the bottom, there are tabs for "Results", "Script Output", "Explain", "Autotrace", "DBMS Output", and "OWA Output". The "Script Output" tab is selected. It displays the message "PROCEDURE del_job Compiled.".

- b) Para llamar al procedimiento y, a continuación, consultar la tabla JOBS, cargue el archivo **sol_01_03_b.sql** en la carpeta **/home/oracle/labs/plpu/solns**. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para llamar al procedimiento. El código y el resultado se muestran de la siguiente forma:

Soluciones a la Práctica 1-1: Creación, Compilación y Llamada de Procedimientos (continuación)

The screenshot shows the Oracle SQL Worksheet interface. The title bar says "sol_01_03_b.sql". The main area contains the following code:

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

Below the code, the results pane shows:

anonymous block completed

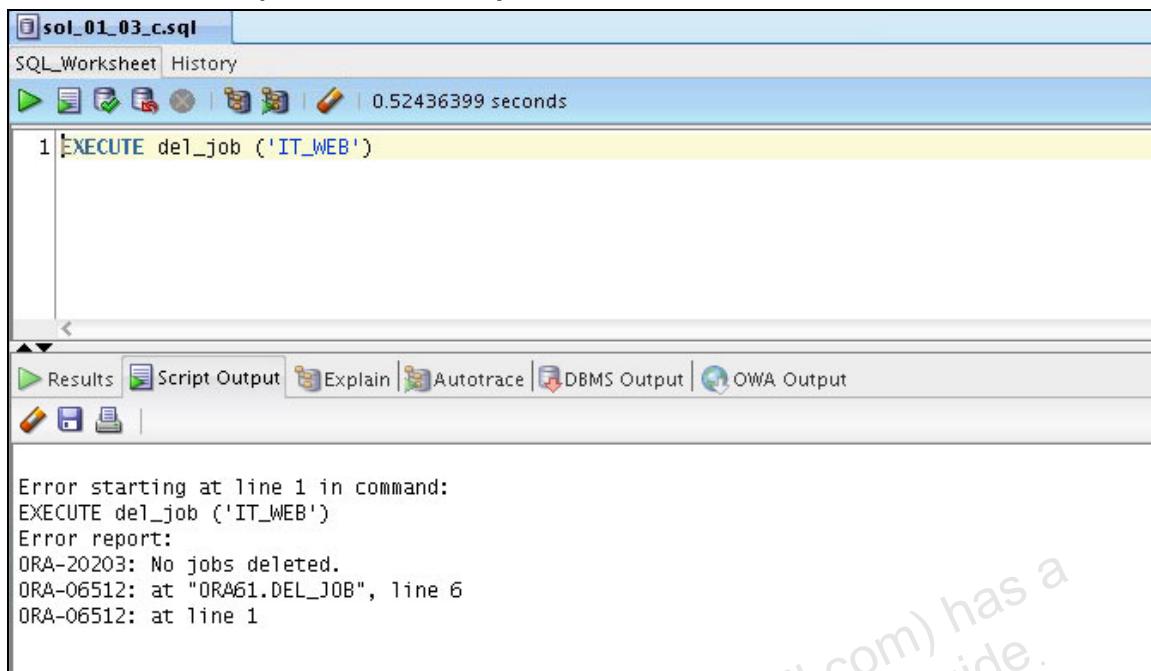
JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
0 rows selected			

The results pane has tabs: Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. The "Results" tab is selected.

- c) Pruebe la sección de manejo de excepciones del procedimiento intentando suprimir un trabajo que no existe. Utilice IT_WEB como identificador de trabajo. Aparecerá el mensaje que haya incluido en la sección de manejo de excepciones del procedimiento como salida.

Para llamar al procedimiento y, a continuación, consultar la tabla JOBS, cargue el archivo **sol_01_03_c.sql** en la carpeta **/home/oracle/labs/plpu/solns**. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para llamar al procedimiento. El código y el resultado se muestran de la siguiente forma:

Soluciones a la Práctica 1-1: Creación, Compilación y Llamada de Procedimientos (continuación)



The screenshot shows a SQL Worksheet window titled "sol_01_03_c.sql". The command entered is "EXECUTE del_job ('IT_WEB')". The results tab shows an error message:

```
Error starting at line 1 in command:
EXECUTE del_job ('IT_WEB')
Error report:
ORA-20203: No jobs deleted.
ORA-06512: at "ORA61.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.
- Cree un procedimiento que devuelva un valor de las columnas SALARY y JOB_ID para el identificador de empleado especificado. Elimine los errores de sintaxis, si los hay y, a continuación, recompile el código.

Abra el script /home/oracle/labs/plpu/solns/sol_01_04_a.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar el procedimiento. El código y el resultado se muestran de la siguiente forma:

Soluciones a la Práctica 1-1: Creación, Compilación y Llamada de Procedimientos (continuación)

The screenshot shows the Oracle SQL Worksheet interface. The code area contains the following PL/SQL procedure:

```

1 CREATE OR REPLACE PROCEDURE get_employee
2   (p_empid IN employees.employee_id%TYPE,
3    p_sal    OUT employees.salary%TYPE,
4    p_job    OUT employees.job_id%TYPE) IS
5 BEGIN
6   SELECT salary, job_id
7   INTO   p_sal, p_job
8   FROM   employees
9   WHERE  employee_id = p_empid;
10 END get_employee;
11 /
12

```

The status bar at the bottom indicates "PROCEDURE get_employee Compiled." Below the code area, there are tabs for Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output.

Nota

Si no aparece el procedimiento que se acaba de crear en Object Navigator, haga clic con el botón derecho en el nodo Procedures de Object Navigator y, a continuación, seleccione Refresh en el menú de acceso directo. En Object Navigator, haga clic con el botón derecho en el nombre del procedimiento y seleccione Compile en el menú de acceso directo. El procedimiento se compila.

The screenshot shows the Oracle Messages - Log window. It displays the message "GET_EMPLOYEE Compiled".

- 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.

Abra el script /home/oracle/labs/plpu/solns/sol_01_04_b.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para llamar al procedimiento. El código y el resultado se muestran de la siguiente forma:

Soluciones a la Práctica 1-1: Creación, Compilación y Llamada de Procedimientos (continuación)

```

sol_01_04_b.sql
SQL Worksheet History
0.57418901 seconds

1 VARIABLE v_salary NUMBER
2 VARIABLE v_job    VARCHAR2(15)
3 EXECUTE get_employee(120, :v_salary, :v_job)
4 PRINT v_salary v_job
5

Results Script Output Explain Autotrace DBMS Output OWA Output
anonymous block completed
v_salary
-----
8000
v_job
-----
ST_MAN

```

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

Ningún empleado de la tabla EMPLOYEES tiene un EMPLOYEE_ID 300. La sentencia SELECT no ha recuperado ningún dato de la base de datos y ha generado un error fatal PL/SQL: NO DATA FOUND de la siguiente forma:

Soluciones a la Práctica 1-1: Creación, Compilación y Llamada de Procedimientos (continuación)

The screenshot shows the Oracle SQL Worksheet interface. The code entered is:

```
1 VARIABLE v_salary NUMBER
2 VARIABLE v_job    VARCHAR2(15)
3 EXECUTE get_employee(300, :v_salary, :v_job)
```

The results pane shows the output of the anonymous block:

```
anonymous block completed
v_salary
-----
8000

v_job
-----
ST_MAN

Error starting at line 1 in command:
EXECUTE get_employee(300, :v_salary, :v_job)
Error report:
ORA-01403: no data found
ORA-06512: at "ORA61.GET_EMPLOYEE", line 6
ORA-06512: at line 1
01403. 00000 -  "no data found"
*Cause:
*Action:
```

A watermark is present across the image: "Jose Enrique Lozano (lozanojose@gmail.com) has a non-transferable license to use this Student Guide."

Prácticas y Soluciones de la Lección 2

En la práctica 2-1, creará, compilará y utilizará lo siguiente:

- Una función denominada GET_JOB para devolver un cargo.
- 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.
- Un procedimiento denominado ADD_EMPLOYEE para insertar un nuevo empleado en la tabla EMPLOYEES.

En la práctica 2-2, se presentará la funcionalidad básica del depurador de SQL Developer:

- Cree un procedimiento y una función.
- Inserte puntos de división en el procedimiento que acaba de crear.
- Compile el procedimiento y la función para el modo de depuración.
- Depure el procedimiento y desplácese a la primera línea ejecutable del código.
- Muestre y modifique las variables de los subprogramas.

Nota: si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

Práctica 2-1: Creación de Funciones

En esta práctica, creará, compilará y utilizará las funciones almacenadas y un procedimiento.

- 1) Cree y llame a la función GET_JOB para devolver un cargo.
 - a) Cree y compile la función denominada GET_JOB para devolver un cargo.
 - b) Cree una variable de host VARCHAR2 denominada b_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 y, a continuación, imprima la variable de host para ver el resultado.

```

Results Script Output Explain Autotrace DBMS Output OWA Output
anonymous block completed
b_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) Cree la función GET_ANNUAL_COMP, que acepta valores de parámetros del salario mensual y la 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:

$$(salary * 12) + (commission_pct * salary * 12)$$
 - b) Utilice la función en una sentencia SELECT en la tabla EMPLOYEES para los empleados del departamento 30.

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

6 rows selected

- 3) Cree un procedimiento, ADD_EMPLOYEE, para insertar un nuevo empleado en la tabla EMPLOYEES. El procedimiento llamará a una función VALID_DEPTID para comprobar si el identificador (ID) de departamento especificado para el nuevo empleado existe en la tabla DEPARTMENTS.
- Cree una función denominada VALID_DEPTID para validar el identificador de departamento especificado y devolver un valor BOOLEAN TRUE si existe el departamento.
 - Cree el procedimiento ADD_EMPLOYEE para agregar un empleado a la tabla EMPLOYEES. La fila se debe agregar 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:
 - first_name
 - last_name
 - email
 - job: utilice 'SA REP' como valor por defecto.
 - mgr: utilice 145 como valor por defecto.
 - sal: utilice 1000 como valor por defecto.
 - comm: utilice 0 como valor por defecto.
 - deptid: utilice 30 como valor por defecto.
 - Utilice la secuencia EMPLOYEES_SEQ para definir la columna employee_id.
 - Defina la columna hire_date en TRUNC(SYSDATE).
 - Llame a ADD_EMPLOYEE para el nombre 'Jane Harris' del departamento 15, dejando otros parámetros con los valores por defecto. ¿Cuál es el resultado?
 - Agregue otro empleado llamado Joe Harris en el departamento 80, dejando los parámetros restantes con sus valores por defecto. ¿Cuál es el resultado?

Práctica 2-2: Introducción al Depurador de SQL Developer

En esta práctica, practicará con la funcionalidad básica del depurador de SQL Developer.

- 1) Active SERVEROUTPUT.
- 2) Ejecute el script `sol_02_02_02.sql` para crear el procedimiento `emp_list`. Examine el código del procedimiento y compile el procedimiento. ¿Por qué aparece el error del compilador?
- 3) Ejecute el script `sol_02_02_03.sql` para crear la función `get_location`. Examine el código de la función, compile la función y, a continuación, corrija los posibles errores.
- 4) Recompile el procedimiento `emp_list`. El procedimiento debe compilarse correctamente.
- 5) Edite el procedimiento `emp_list` y la función `get_location`.
- 6) Agregue cuatro puntos de división al procedimiento `emp_list` a las siguientes líneas de código:
 - a) `OPEN emp_cursor;`
 - b) `WHILE (emp_cursor%FOUND) AND (i <= pMaxRows) LOOP`
 - c) `v_city := get_location (emp_record.department_name);`
 - d) `CLOSE emp_cursor;`
- 7) Compile el procedimiento `emp_list` para la depuración.
- 8) Depure el procedimiento.
- 9) Introduzca 100 como valor del parámetro `PMAXROWS`.
- 10) Examine el valor de las variables en el separador Data. ¿Cuáles son los valores asignados a `REC_EMP` y `EMP_TAB`? ¿Por qué?
- 11) Utilice la opción de depuración Step Into para ir a cada línea de código de `emp_list` y pasar por el bucle while sólo una vez.
- 12) Examine el valor de las variables en el separador Data. ¿Cuáles son los valores asignados a `REC_EMP`?
- 13) Siga pulsando F7 hasta que se ejecute la línea `emp_tab(i) := rec_emp;`. Examine el valor de las variables en el separador Data. ¿Cuáles son los valores asignados a `EMP_TAB`?
- 14) Utilice el separador Data para modificar el valor del contador `i` a 98.
- 15) Siga pulsando F7 hasta que vea la lista de empleados en el separador Debugging – Log. ¿Cuántos empleados aparecen?
- 16) Si utiliza la opción del depurador Step Over para desplazarse por el código, ¿se desplaza por la función `get_location`? ¿Por qué?

Soluciones a la Práctica 2-1: Creación de Funciones

En esta práctica, creará, compilará y utilizará las funciones almacenadas y un procedimiento.

- 1) Cree y llame a la función GET_JOB para devolver un cargo.

- a) Cree y compile la función denominada GET_JOB para devolver un cargo.

Abra el script /home/oracle/labs/plpu/solns/sol_02_01_01_a.sql.

Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL

Worksheet para crear y compilar la función. El código y el resultado se muestran de la siguiente forma:

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



- b) Cree una variable de host VARCHAR2 denominada b_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 y, a continuación, imprima la variable de host para ver el resultado.

Abra el script /home/oracle/labs/plpu/solns/sol_02_01_01_b.sql.

Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL

Worksheet para crear y compilar la función. El código y el resultado se muestran de la siguiente forma:

```
VARIABLE b_title VARCHAR2(35)
EXECUTE :b_title := get_job ('SA REP');
PRINT b_title
```

Soluciones a la Práctica 2-1: Creación de Funciones (continuación)

```

Results Script Output Explain Autotrace DBMS Output OWA Output
anonymous block completed
b_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.
- Cree la función GET_ANNUAL_COMP, que acepta valores de parámetros del salario mensual y la 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:

$$(salary * 12) + (commission_pct * salary * 12)$$

Abra el script /home/oracle/labs/plpu/solns/sol_02_01_02_a.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar la función. El código y el resultado se muestran de la siguiente forma:

```

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

```

```

Results Script Output Explain Autotrace DBMS Output OWA Output
FUNCTION get_annual_comp( Compiled.

```

- Utilice la función en una sentencia SELECT en la tabla EMPLOYEES para los empleados del departamento 30.
- Abra el script /home/oracle/labs/plpu/solns/sol_02_01_02_b.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar la función. El código y el resultado se muestran de la siguiente forma:

Soluciones a la Práctica 2-1: Creación de Funciones (continuación)

```
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 (ID) de departamento especificado para el nuevo empleado existe en la tabla DEPARTMENTS.

- a) Cree una función denominada VALID_DEPTID para validar el identificador de departamento especificado y devolver un valor BOOLEAN TRUE si existe el departamento.

Abra el script /home/oracle/labs/plpu/solns/sol_02_01_03_a.sql.
Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear la función. El código y el resultado se muestran de la siguiente forma:

```
CREATE OR REPLACE FUNCTION valid_deptid(
  p_deptid IN departments.department_id%TYPE)
RETURN BOOLEAN IS
  v_dummy  PLS_INTEGER;

BEGIN
  SELECT 1
  INTO   v_dummy
  FROM   departments
  WHERE  department_id = p_deptid;
  RETURN TRUE;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END valid_deptid;
/
```

Soluciones a la Práctica 2-1: Creación de Funciones (continuación)

The screenshot shows a toolbar with tabs: Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. Below the toolbar, there are icons for Save, Undo, Redo, and Print. The main area displays the message: "FUNCTION valid_deptid(Compiled." followed by a blank line.

- b) Cree el procedimiento ADD_EMPLOYEE para agregar un empleado a la tabla EMPLOYEES. La fila se debe agregar 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:

- first_name
- last_name
- email
- job: utilice 'SA REP' como valor por defecto.
- mgr: utilice 145 como valor por defecto.
- sal: utilice 1000 como valor por defecto.
- comm: utilice 0 como valor por defecto.
- deptid: utilice 30 como valor por defecto.
- Utilice la secuencia EMPLOYEES_SEQ para definir la columna employee_id.
- Defina la columna hire_date en TRUNC(SYSDATE).

Abra el script /home/oracle/labs/plpu/solns/sol_02_01_03_b.sql.

Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL

Worksheet para crear y compilar el procedimiento. El código y el resultado se muestran de la siguiente forma:

```
CREATE OR REPLACE PROCEDURE add_employee (
    p_first_name employees.first_name%TYPE,
    p_last_name  employees.last_name%TYPE,
    p_email      employees.email%TYPE,
    p_job        employees.job_id%TYPE          DEFAULT
'SA REP',
    p_mgr        employees.manager_id%TYPE       DEFAULT 145,
    p_sal        employees.salary%TYPE           DEFAULT 1000,
    p_comm       employees.commission_pct%TYPE   DEFAULT 0,
    p_deptid     employees.department_id%TYPE   DEFAULT 30)
IS
BEGIN
    IF valid_deptid(p_deptid) THEN
```

Soluciones a la Práctica 2-1: Creación de Funciones (continuación)

```

    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, p_first_name,
p_last_name, p_email,
p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
ELSE
    RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
END IF;
END add_employee;
/

```

The screenshot shows a SQL Worksheet interface with several tabs at the top: Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. The Script Output tab is active, displaying the compiled code:

```
PROCEDURE add_employee( Compiled.
```

- 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?

Abra el script /home/oracle/labs/plpu/solns/sol_02_01_03_c.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para llamar al procedimiento. El código y el resultado se muestran de la siguiente forma:

```
EXECUTE add_employee('Jane', 'Harris', 'JAHARRIS', p_deptid=> 15)
```

The screenshot shows the SQL Worksheet interface with the same tabs at the top. The Script Output tab is active, displaying the following error message:

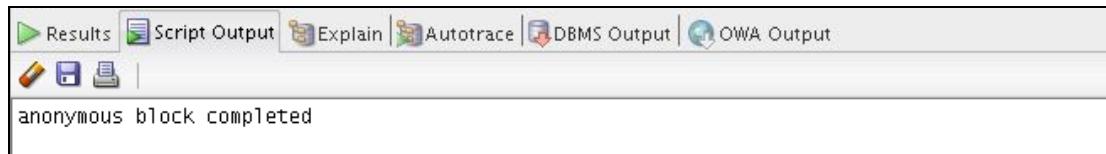
```
Error starting at line 1 in command:
EXECUTE add_employee('Jane', 'Harris', 'JAHARRIS', p_deptid=> 15)
Error report:
ORA-20204: Invalid department ID. Try again.
ORA-06512: at "ORA61.ADD_EMPLOYEE", line 17
ORA-06512: at line 1
```

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

Abra el script /home/oracle/labs/plpu/solns/sol_02_01_03_d.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para llamar al procedimiento. El código y el resultado se muestran de la siguiente forma:

Soluciones a la Práctica 2-1: Creación de Funciones (continuación)

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



Soluciones a la Práctica 2-2: Introducción al Depurador de SQL Developer

En esta práctica, probará la funcionalidad básica del depurador de SQL Developer.

- 1) Active SERVEROUTPUT.

Introduzca el siguiente comando en el área SQL Worksheet y, a continuación, haga clic en el icono Run Script (F5). Haga clic en el icono en la barra de herramientas de SQL Worksheet.

```
SET SERVEROUTPUT ON
```

- 2) Ejecute el script `sol_02_02_02.sql` para crear el procedimiento `emp_list`. Examine el código del procedimiento y compile el procedimiento. ¿Por qué aparece el error del compilador?

Abra el script `/home/oracle/labs/plpu/solns/sol_02_02_02.sql`. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar el procedimiento. El código y el resultado se muestran de la siguiente forma:

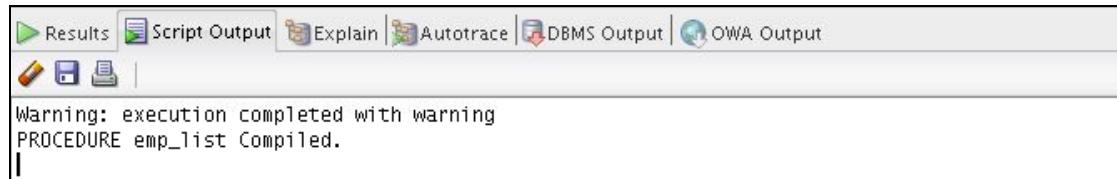
```
CREATE OR REPLACE PROCEDURE emp_list
(p_maxrows IN NUMBER)
IS
CURSOR cur_emp IS
  SELECT d.department_name, e.employee_id, e.last_name,
         e.salary, e.commission_pct
    FROM departments d, employees e
   WHERE d.department_id = e.department_id;
  rec_emp cur_emp%ROWTYPE;
  TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY
BINARY_INTEGER;
  emp_tab emp_tab_type;
  i NUMBER := 1;
  v_city VARCHAR2(30);
BEGIN
  OPEN cur_emp;
  FETCH cur_emp INTO rec_emp;
  emp_tab(i) := rec_emp;
  WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
    i := i + 1;
    FETCH cur_emp INTO rec_emp;
    emp_tab(i) := rec_emp;
    v_city := get_location (rec_emp.department_name);
    dbms_output.put_line('Employee ' || rec_emp.last_name ||
      ' works in ' || v_city );
  END LOOP;
  CLOSE cur_emp;
```

Soluciones a la Práctica 2-2: Introducción al Depurador de SQL Developer (continuación)

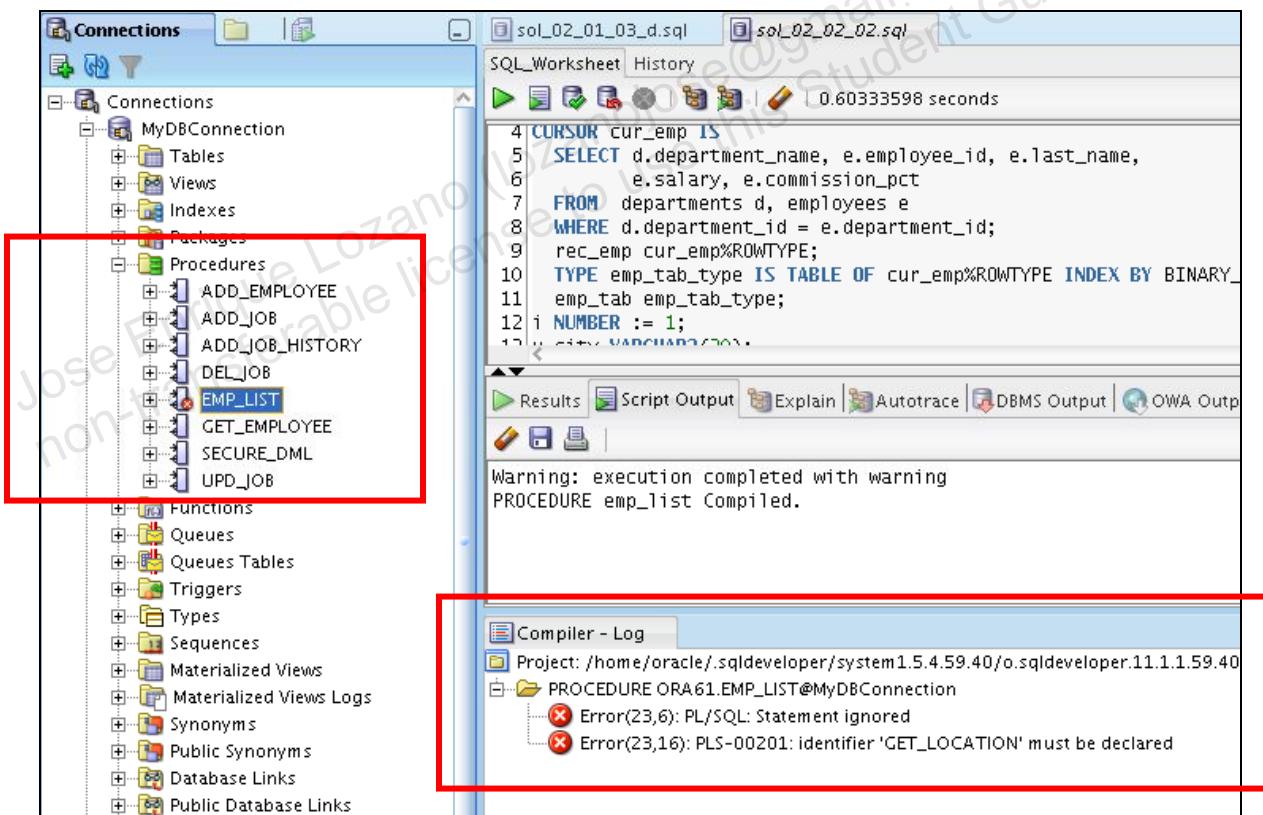
```

FOR j IN REVERSE 1..i LOOP
    DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
END LOOP;
END emp_list;
/

```



La advertencia de compilación se produce porque no se ha declarado aún la función `get_location`. Para que aparezca el error de compilación con más detalles, haga clic con el botón derecho en el procedimiento `EMP_LIST` del nodo Procedures (puede que tenga que refrescar la lista de procedimientos para ver el procedimiento `EMP_LIST` que se acaba de crear) y, a continuación, seleccione Compile en el menú emergente. El mensaje de advertencia detallado se muestra en el separador Compiler-Log, de la siguiente forma:



- 3) Ejecute el script `sol_02_02_03.sql` para crear la función `get_location`. Examine el código de la función, compile la función y, a continuación, corrija los posibles errores.

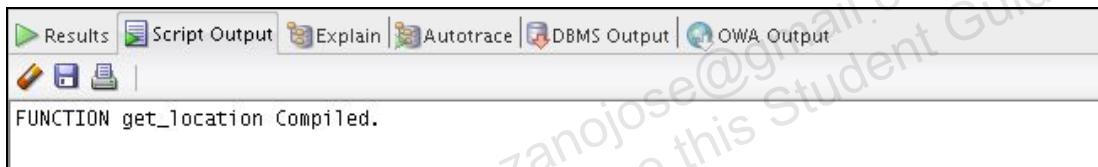
Soluciones a la Práctica 2-2: Introducción al Depurador de SQL Developer (continuación)

Abra el script `/home/oracle/labs/plpu/solns/sol_02_02_03.sql`.

Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL

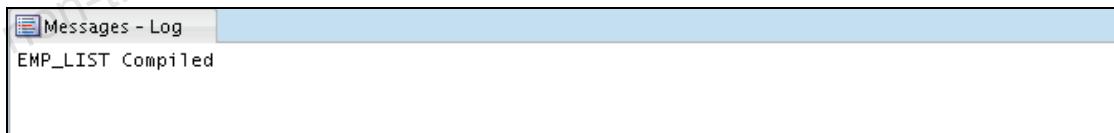
Worksheet para crear y compilar el procedimiento. El código y el resultado se muestran de la siguiente forma:

```
CREATE OR REPLACE FUNCTION get_location
( p_deptname IN VARCHAR2) RETURN VARCHAR2
AS
    v_loc_id NUMBER;
    v_city    VARCHAR2(30);
BEGIN
    SELECT d.location_id, l.city INTO v_loc_id, v_city
    FROM departments d, locations l
    WHERE upper(department_name) = upper(p_deptname)
    and d.location_id = l.location_id;
    RETURN v_city;
END GET_LOCATION;
/
```



- 4) Recompile el procedimiento `emp_list`. El procedimiento debe compilarse correctamente.

Para recompilar el procedimiento, haga clic con el botón derecho en el nombre del procedimiento y, a continuación, seleccione Compile en el menú de acceso directo.



- 5) Edite el procedimiento `emp_list` y la función `get_location`.

Haga clic con el botón derecho en el nombre del procedimiento `emp_list` en Object Navigator y, a continuación, seleccione Edit. Se abre el procedimiento `emp_list` en el modo de edición. Si el procedimiento ya aparece en el área SQL Worksheet, pero está en modo de sólo lectura, haga clic en el ícono Edit (ícono de lápiz) en el separador Code.

Haga clic con el botón derecho en el nombre de la función `get_location` en Object Navigator y, a continuación, seleccione Edit. Se abre la función `get_location` en el modo de edición. Si la función ya aparece en el área SQL Worksheet, pero está en modo de sólo lectura, haga clic en el ícono Edit (ícono de lápiz) en el separador Code.

Soluciones a la Práctica 2-2: Introducción al Depurador de SQL Developer (continuación)

- 6) Agregue cuatro puntos de división al procedimiento emp_list a las siguientes líneas de código:

- a) OPEN emp_cursor;
- b) WHILE (emp_cursor%FOUND) AND (i <= pMaxRows) LOOP
- c) v_city := get_location (emp_record.department_name);
- d) CLOSE emp_cursor;

Para agregar un punto de división, haga clic en el canal de línea situado junto a cada una de las líneas mencionadas anteriormente, como se muestra a continuación:

```

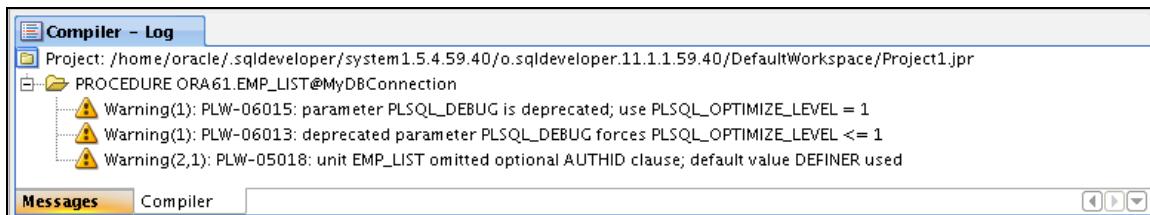
create or replace
PROCEDURE emp_list
(p_maxrows IN NUMBER)
IS
CURSOR cur_emp IS
  SELECT d.department_name, e.employee_id, e.last_name,
         e.salary, e.commission_pct
    FROM departments d, employees e
   WHERE d.department_id = e.department_id;
  rec_emp CURSOR%ROWTYPE;
  TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY BINARY_INTEGER;
  emp_tab emp_tab_type;
  i NUMBER := 1;
  v_city VARCHAR2(30);
BEGIN
  OPEN cur_emp;
  FETCH cur_emp INTO rec_emp;
  emp_tab(i) := rec_emp;
  WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
    i := i + 1;
    FETCH cur_emp INTO rec_emp;
    emp_tab(i) := rec_emp;
    v_city := get_location (rec_emp.department_name);
    dbms_output.put_line('Employee ' || rec_emp.last_name ||
      ' works in ' || v_city );
  END LOOP;
  CLOSE cur_emp;
  FOR j IN REVERSE 1..i LOOP
    DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
  END LOOP;
END emp_list;

```

- 7) Compile el procedimiento emp_list para la depuración.

Haga clic en el ícono “Compile for Debug” de la barra de herramientas del procedimiento, como se muestra a continuación:

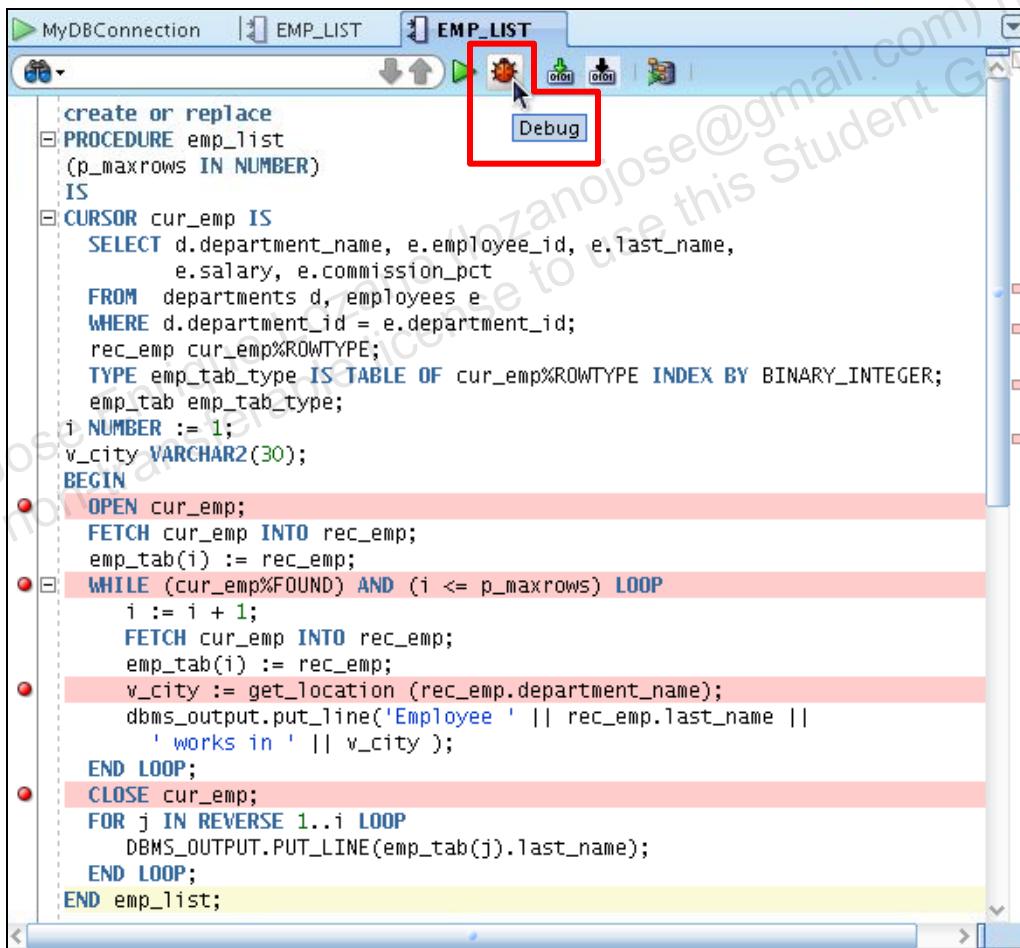
Soluciones a la Práctica 2-2: Introducción al Depurador de SQL Developer (continuación)



Nota: que aparezca la advertencia anterior es algo esperado. Las dos primeras advertencias se producen porque el parámetro PLSQL_DEBUG ya no se utiliza en Oracle Database 11g, mientras que SQL Developer sigue usándolo. La última advertencia hace referencia al uso de la cláusula AUTHID con un procedimiento. Esta cláusula se tratará en una lección posterior.

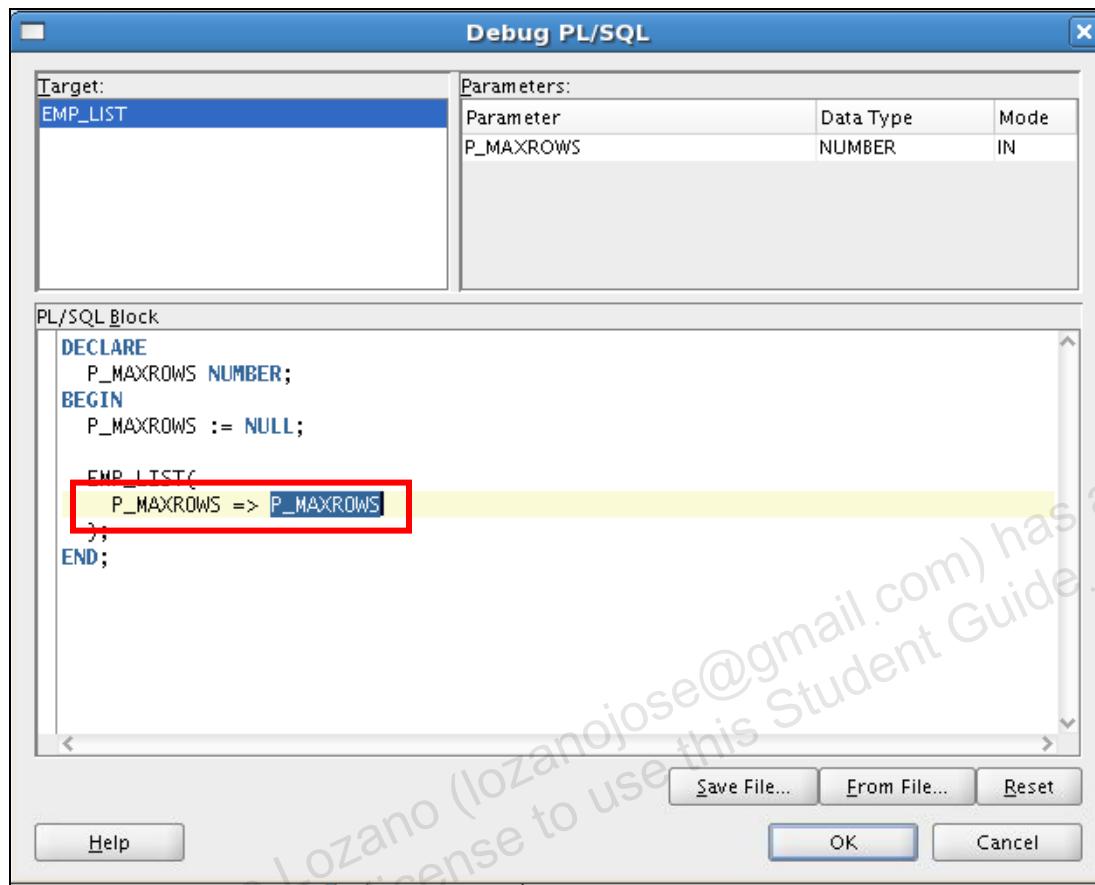
- 8) Depure el procedimiento.

Haga clic en el icono Debug de la barra de herramientas del procedimiento, como se muestra a continuación:



Soluciones a la Práctica 2-2: Introducción al Depurador de SQL Developer (continuación)

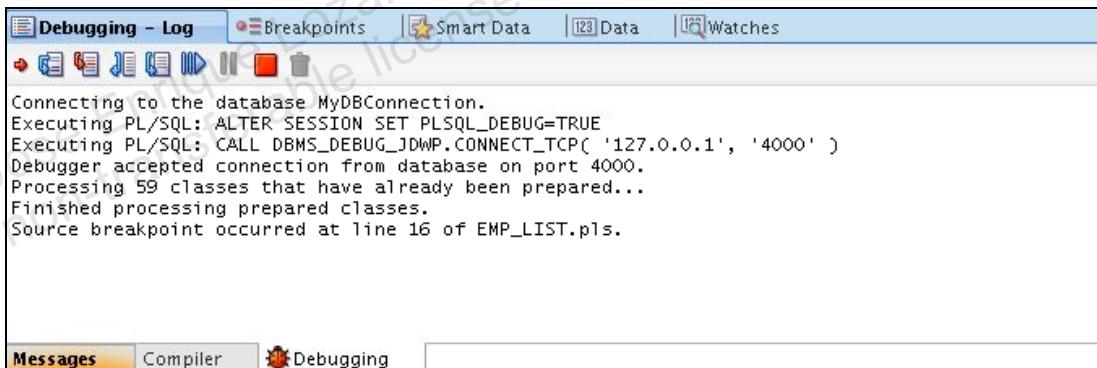
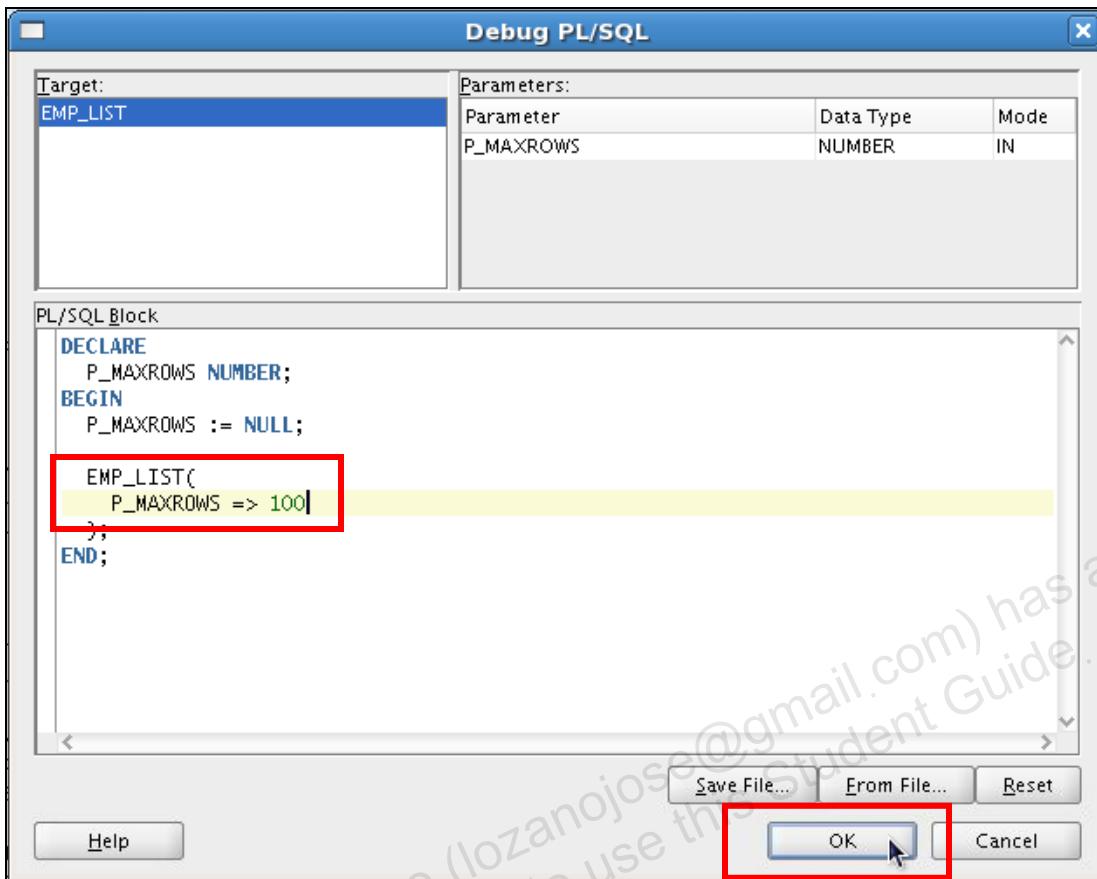
La ventana Debug PL/SQL se muestra de la siguiente forma:



- 9) Introduzca 100 como valor del parámetro PMAXROWS.

Sustituya el segundo P_MAXROWS por 100 y, a continuación, haga clic en OK. Tenga en cuenta cómo el control del programa se para en el primer punto de división del procedimiento, como indica el color azul resaltado y la flecha roja que señala a esa línea del código. Los demás separadores de depuración se muestran en la parte inferior de la página.

Soluciones a la Práctica 2-2: Introducción al Depurador de SQL Developer (continuación)



- 10) Examine el valor de las variables en el separador Data. ¿Cuáles son los valores asignados a REC_EMP y EMP_TAB? ¿Por qué?

Ambos se definen en NULL porque los datos aún no se han recuperado en el cursor.

Soluciones a la Práctica 2-2: Introducción al Depurador de SQL Developer (continuación)

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

- 11) Utilice la opción de depuración Step Into para ir a cada línea de código de `emp_list` y pasar por el bucle while sólo una vez.

Pulse [F7] para ir a la primera línea ejecutable del código una sola vez.

- 12) Examine el valor de las variables en el separador Data. ¿Cuáles son los valores asignados a `REC_EMP` y `EMP_TAB`?

Tenga en cuenta que cuando se ejecuta la línea `FETCH cur_emp INTO rec_emp;`, `rec_emp` se inicializa como se muestra a continuación:

Name	Value	Type
P_MAXROWS	100	NUMBER
REC_EMP		Rowtype
DEPARTMENT_NAME	'Administration'	VARCHAR2(30)
EMPLOYEE_ID	200	NUMBER(6,0)
LAST_NAME	"Whalen"	VARCHAR2(25)
SALARY	4400	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)
EMP_TAB	indexed table	EMP_TAB_TYPE
_values		EMP_TAB_TYPE element[0]
I	1	NUMBER
V_CITY	NULL	VARCHAR2(30)

- 13) Siga pulsando F7 hasta que se ejecute la línea `emp_tab(i) := rec_emp;`. Examine el valor de las variables en el separador Data. ¿Cuáles son los valores asignados a `EMP_TAB`?

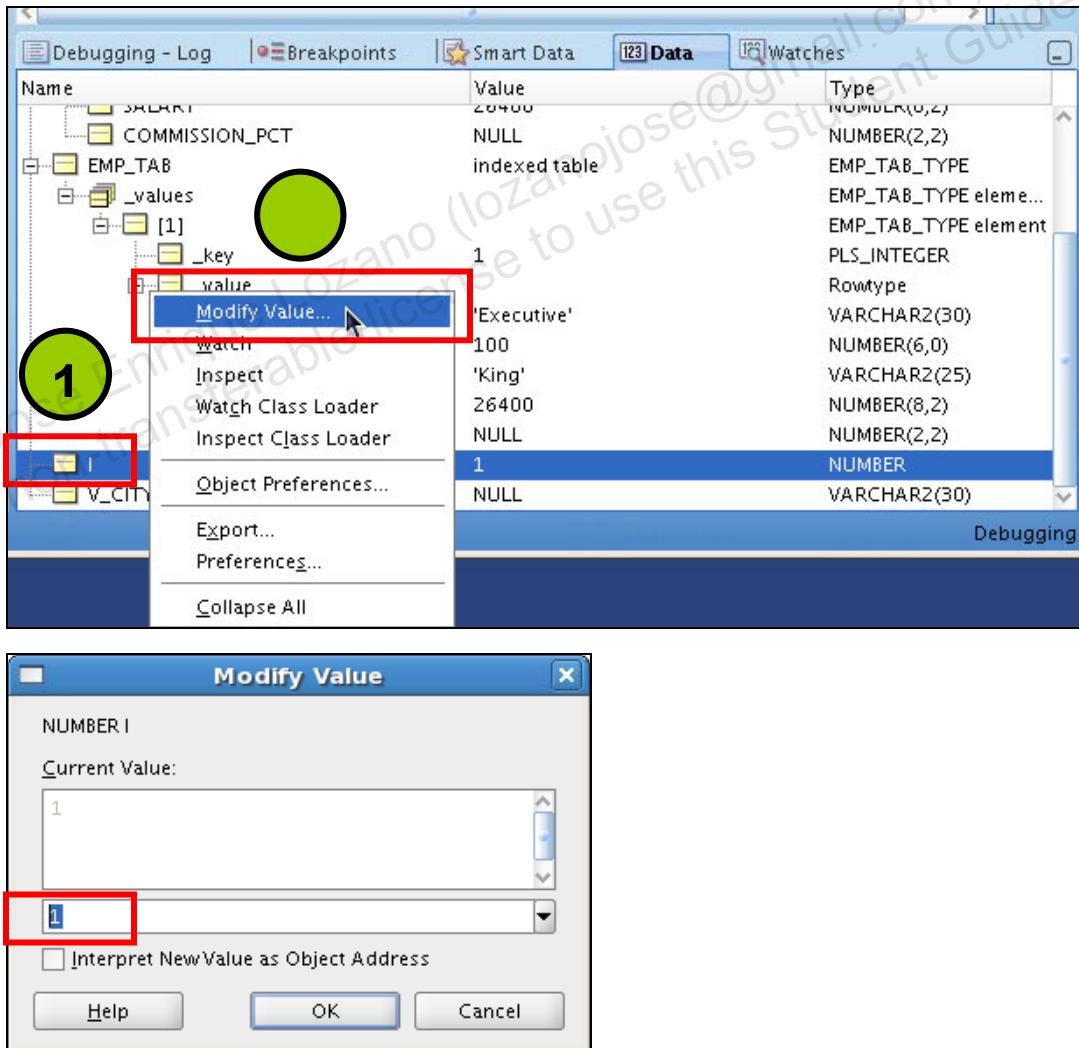
Cuando se ejecuta la línea `emp_tab(i) := rec_emp;`, `emp_tab` se inicializa en `rec_emp`, como se muestra a continuación:

Soluciones a la Práctica 2-2: Introducción al Depurador de SQL Developer (continuación)

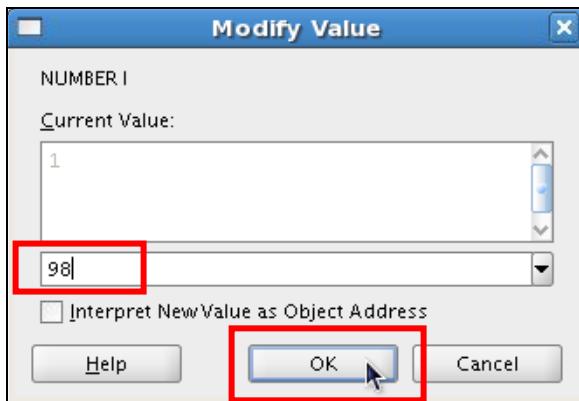
Name	Value	Type
SALARY	4400	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)
EMP_TAB	indexed table	EMP_TAB_TYPE
_values		EMP_TAB_TYPE element[1]
[1]	1	EMP_TAB_TYPE element
_key		PLS_INTEGER
_value		Rowtype
DEPARTMENT_NAME	'Administration'	VARCHAR2(30)
EMPLOYEE_ID	200	NUMBER(6,0)
LAST_NAME	'Whalen'	VARCHAR2(25)
SALARY	4400	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)

- 14) Utilice el separador Data para modificar el valor del contador i a 98.

En el separador Data, haga clic con el botón derecho en I y, a continuación, seleccione Modify Value en el menú de acceso directo. Aparece la ventana Modify Value. Sustituya el valor 1 por 98 en el cuadro de texto y, a continuación, haga clic en OK como se muestra a continuación:



Soluciones a la Práctica 2-2: Introducción al Depurador de SQL Developer (continuación)



- 15) Siga pulsado F7 hasta que vea la lista de empleados en el separador Debugging – Log. ¿Cuántos empleados aparecen?

A continuación se muestra la salida de la sesión de depuración, donde aparecen tres empleados:

```

Debugging - Log
Exception breakpoint occurred at line 29 of EMP_LIST.pls.
$Oracle.EXCEPTION_ORA_1403:
ORA-01403: no data found
ORA-06512: at "ORA61.EMP_LIST", line 28
ORA-06512: at line 6
Executing PL/SQL: CALL DBMS_DEBUG_JDWP.DISCONNECT()
Employee Fay works in Toronto
Employee Hartstein works in Toronto
Employee Colmenares works in Seattle
Colmenares
Hartstein
Fay
Process exited.
Disconnecting from the database MyDBConnection.
Debugger disconnected from database.

```

- 16) Si utiliza la opción del depurador Step Over para desplazarse por el código, ¿se desplaza por la función `get_location`? ¿Por qué?

Si bien la línea de código donde está definido el tercer punto de división contiene una llamada a la función `get_location`, con Step Over (F8) se ejecuta la línea de código y se recupera el valor devuelto de la función (igual que con [F7]); sin embargo, el control no se transfiere a la función `get_location`.

Prácticas y Soluciones de la Lección 3

En esta práctica, creará una especificación del paquete y un cuerpo del paquete denominados JOB_PKG, con una copia de los procedimientos ADD_JOB, UPD_JOB y DEL_JOB, así como su función GET_JOB. También creará y llamará a un paquete que contenga construcciones privadas y públicas mediante los datos de ejemplo.

Nota: si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

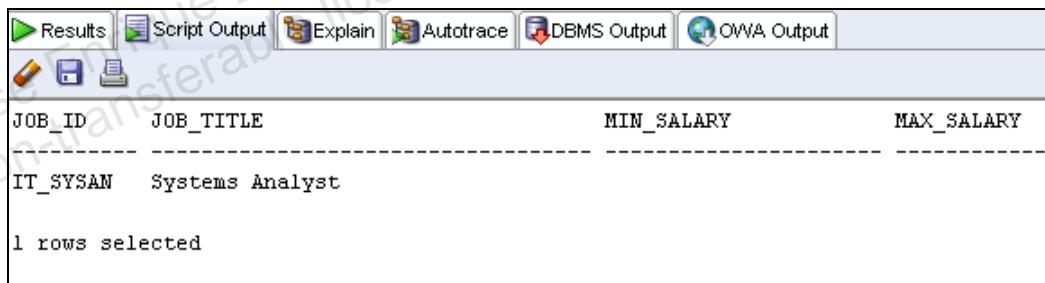
Práctica 3-1: Creación y Uso de Paquetes

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

- Cree una especificación del paquete y un cuerpo del paquete denominados JOB_PKG, con una copia de los procedimientos ADD_JOB, UPD_JOB y DEL_JOB, así como su función GET_JOB.

Nota: utilice el código de los procedimientos y las funciones guardados previamente al crear el paquete. Puede copiar el código de un procedimiento o una función para, a continuación, pegarlo en la sección adecuada del paquete.

- Cree la especificación del paquete, incluidos los procedimientos y las cabeceras de función como construcciones públicas.
- Cree el cuerpo del paquete con las implantaciones de cada uno de los subprogramas.
- Suprima los procedimientos y función autónomos siguientes que acaba de empaquetar con los nodos Procedures y Functions del árbol Object Navigation:
 - Los procedimientos ADD_JOB, UPD_JOB y DEL_JOB
 - La función GET_JOB
- Llame al procedimiento empaquetado ADD_JOB transfiriendo como parámetros los valores IT_SYSAN y SYSTEMS ANALYST.
- Consulte la tabla JOBS para ver el resultado.



The screenshot shows the results of a SQL query on the JOBS table. The results are as follows:

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_SYSAN	Systems Analyst		

1 rows selected

- Cree y llame a un paquete que contenga construcciones públicas y privadas.
 - Cree una especificación del paquete y un cuerpo del paquete denominados EMP_PKG, que contenga los siguientes procedimientos y función creados anteriormente:
 - Procedimiento ADD_EMPLOYEE como construcción pública
 - Procedimiento GET_EMPLOYEE como construcción pública
 - Función VALID_DEPTID como construcción privada

Práctica 3-1: Creación y Uso de Paquetes (continuación)

- b) Llame al procedimiento EMP_PKG.ADD_EMPLOYEE, con el identificador de departamento 15 para la empleada Jane Harris con identificador de correo electrónico JAHARRIS. Como el identificador de departamento 15 no existe, recibirá un mensaje de error como se especifica en el manejador de excepciones del procedimiento.
- c) Llame al procedimiento empaquetado ADD_EMPLOYEE utilizando el identificador de departamento 80 para el empleado David Smith con el identificador de correo electrónico DASMITH.
- d) Consulte la tabla EMPLOYEES para verificar que se ha agregado el nuevo empleado.

Soluciones a la Práctica 3-1: Creación y Uso de Paquetes

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

- Cree una especificación del paquete y un cuerpo del paquete denominados JOB_PKG, con una copia de los procedimientos ADD_JOB, UPD_JOB y DEL_JOB, así como su función GET_JOB.

Nota: utilice el código de los procedimientos y las funciones guardados previamente al crear el paquete. Puede copiar el código de un procedimiento o una función para, a continuación, pegarlo en la sección adecuada del paquete.

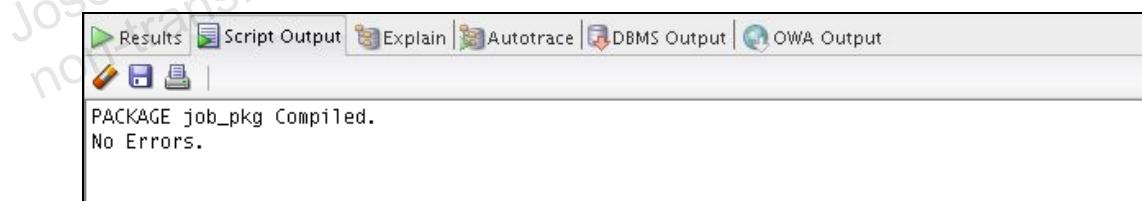
- Cree la especificación del paquete, incluidos los procedimientos y las cabeceras de función como construcciones públicas.

Abra el script /home/oracle/labs/plpu/solns/sol_03_01_a.sql.

Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL

Worksheet para crear y compilar la especificación del paquete. El código y el resultado se muestran de la siguiente forma:

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



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

Abra el script /home/oracle/labs/plpu/solns/sol_03_01_b.sql.

Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL

Worksheet para crear y compilar el cuerpo del paquete. El código y el resultado se muestran de la siguiente forma:

```
CREATE OR REPLACE PACKAGE BODY job_pkg IS
    PROCEDURE add_job (
        p_jobid jobs.job_id%TYPE,
```

Soluciones a la Práctica 3-1: Creación y Uso de Paquetes (continuación)

```

p_jobtitle jobs.job_title%TYPE) IS
BEGIN
    INSERT INTO jobs (job_id, job_title)
    VALUES (p_jobid, p_jobtitle);
    COMMIT;
END add_job;

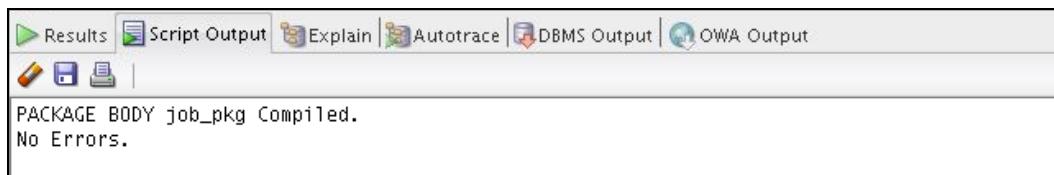
PROCEDURE del_job (p_jobid jobs.job_id%TYPE) IS
BEGIN
    DELETE FROM jobs
    WHERE job_id = p_jobid;
    IF SQL%NOTFOUND THEN
        RAISE_APPLICATION_ERROR(-20203, 'No jobs
deleted.');
    END IF;
END DEL_JOB;

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

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

END job_pkg;
/
SHOW ERRORS

```



Soluciones a la Práctica 3-1: Creación y Uso de Paquetes (continuación)

- c) Suprime los procedimientos y función autónomos siguientes que acaba de empaquetar con los nodos Procedures y Functions del árbol Object Navigation:

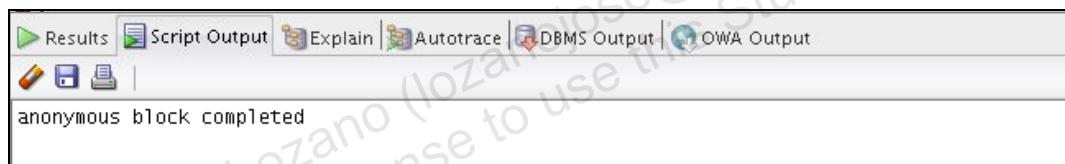
- Los procedimientos ADD_JOB, UPD_JOB y DEL_JOB
- La función GET_JOB

Para suprimir un procedimiento o una función, haga clic con el botón derecho en el nombre del procedimiento o de la función en el árbol Object Navigation y, a continuación, seleccione Drop en el menú emergente. Se mostrará la ventana Drop. Haga clic en Apply para borrar el procedimiento o la función. Aparece una ventana de confirmación; haga clic en OK.

- d) Llame al procedimiento empaquetado ADD_JOB transfiriendo como parámetros los valores IT_SYSAN y SYSTEMS ANALYST.

Abra el script /home/oracle/labs/plpu/solns/sol_03_01_d.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para llamar al procedimiento del paquete. El código y el resultado se muestran de la siguiente forma:

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



- e) Consulte la tabla JOBS para ver el resultado.

Abra el script /home/oracle/labs/plpu/solns/sol_03_01_e.sql. Haga clic en el icono Run Script (F5) o en Execute Statement (F9) en la barra de herramientas de SQL Worksheet para consultar la tabla JOBS. El código y el resultado (utilizando el icono Run Script) se muestran de la siguiente forma:

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

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_SYSAN	Systems Analyst		
1 rows selected			

Soluciones a la Práctica 3-1: Creación y Uso de Paquetes (continuación)

- 2) Cree y llame a un paquete que contenga construcciones públicas y privadas.
- Cree una especificación del paquete y un cuerpo del paquete denominados EMP_PKG, que contenga los siguientes procedimientos y función creados anteriormente:
 - Procedimiento ADD_EMPLOYEE como construcción pública
 - Procedimiento GET_EMPLOYEE como construcción pública
 - Función VALID_DEPTID como construcción privada

Abra el script /home/oracle/labs/plpu/solns/sol_03_02_a.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para llamar al procedimiento del paquete. El código y el resultado se muestran de la siguiente forma:

```

CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);
  PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);
END emp_pkg;
/
SHOW ERRORS

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

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,

```

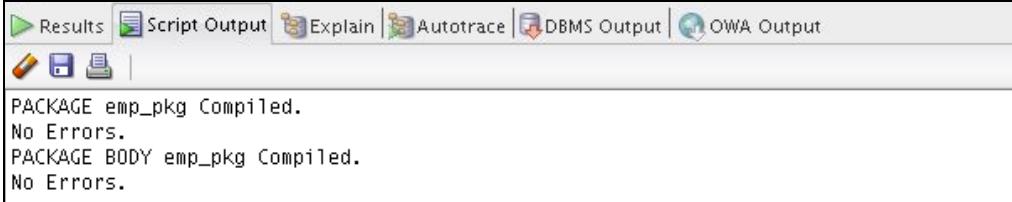
Soluciones a la Práctica 3-1: Creación y Uso de Paquetes (continuación)

```

p_last_name employees.last_name%TYPE,
p_email employees.email%TYPE,
p_job employees.job_id%TYPE DEFAULT 'SA_REP',
p_mgr employees.manager_id%TYPE DEFAULT 145,
p_sal employees.salary%TYPE DEFAULT 1000,
p_comm employees.commission_pct%TYPE DEFAULT 0,
p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_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, p_first_name,
p_last_name, p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid
department ID. Try again.');
    END IF;
END add_employee;

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

```



The screenshot shows the Oracle SQL Developer interface with the 'Script Output' tab selected. The output window displays the following messages:

```

PACKAGE emp_pkg Compiled.
No Errors.
PACKAGE BODY emp_pkg Compiled.
No Errors.

```

Soluciones a la Práctica 3-1: Creación y Uso de Paquetes (continuación)

- b) Llame al procedimiento EMP_PKG.ADD_EMPLOYEE, con el identificador de departamento 15 para la empleada Jane Harris con identificador de 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.

Abra el script /home/oracle/labs/plpu/solns/sol_03_02_b.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para llamar al procedimiento del paquete. El código y el resultado se muestran de la siguiente forma:

Nota: debe realizar el paso 3-2-a antes de realizar este paso. Si no ha realizado el paso 3-2-a, ejecute antes el script sol_03_02_a.sql.

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

```
Error starting at line 1 in command:
EXECUTE emp_pkg.add_employee('Jane', 'Harris', 'JAHARRIS', p_deptid => 15)
Error report:
ORA-20204: Invalid department ID. Try again.
ORA-06512: at "ORA01.EMP_PKG", line 31
ORA-06512: at line 1
```

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

Abra el script /home/oracle/labs/plpu/solns/sol_03_02_c.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para llamar al procedimiento del paquete. El código y el resultado se muestran de la siguiente forma:

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

```
anonymous block completed
```

- d) Consulte la tabla EMPLOYEES para verificar que se ha agregado el nuevo empleado.

Soluciones a la Práctica 3-1: Creación y Uso de Paquetes (continuación)

Abra el script `/home/oracle/labs/plpu/solns/sol_03_02_d.sql`.

Haga clic en el ícono Run Script (F5) o en el ícono Execute Statement (F9) (asegurándose de que el cursor esté en cualquier parte del código de la sentencia SELECT) en la barra de herramientas de SQL Worksheet para consultar la tabla EMPLOYEES. El código y el resultado (ícono Execute Statement) se muestran de la siguiente forma:

```
SELECT *
FROM employees
WHERE last_name = 'Smith';
```

La siguiente salida se muestra en el separador Results, porque hemos ejecutado el código con el ícono F9.

Results										
Script Output Explain Autotrace DBMS Output OWA Output										
Results:										
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	208	David	Smith	DASMITH (null)	19-AUG-09	SA_REP	1000	0	145	80
2	159	Lindsey	Smith	LSMITH	011.44.1345.729268	10-MAR-97	SA_REP	8000	0.3	146
3	171	William	Smith	WSMITH	011.44.1343.629268	23-FEB-99	SA_REP	7400	0.15	148

Prácticas y Soluciones de la Lección 4

En esta práctica, modificará un paquete existente para que contenga subprogramas sobrecargados y utilizará declaraciones anticipadas. También creará un bloque de inicialización de paquete en el cuerpo del paquete para llenar una tabla PL/SQL.

Nota: si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

Práctica 4-1: Trabajar con Paquetes

En esta práctica, modificará el código del paquete EMP_PKG que ha creado anteriormente y, a continuación, sobrecargará el procedimiento ADD_EMPLOYEE. A continuación, creará dos funciones sobrecargadas denominadas GET_EMPLOYEE en el paquete EMP_PKG. También agregará un procedimiento público a EMP_PKG para llenar una tabla PL/SQL privada de identificadores de departamento válidos, además de modificar la función VALID_DEPTID para utilizar el contenido de la tabla PL/SQL privada con el fin de validar los valores de identificador válidos. También cambiará la función de procesamiento de validación VALID_DEPTID para utilizar la tabla PL/SQL privada de identificadores de departamento. Por último, reorganizará los subprogramas en el cuerpo y la especificación del paquete para que estén en secuencia alfabética.

- 1) Modifique el código del paquete EMP_PKG que ha creado en la Práctica 4, paso 2 y sobrecargue el procedimiento ADD_EMPLOYEE.
 - a) En la especificación del paquete, agregue un nuevo procedimiento denominado ADD_EMPLOYEE, que acepte los tres parámetros siguientes:
 - i) First name
 - ii) Last name
 - iii) Department ID
 - b) Haga clic en Run Script (F5) para crear y compilar el paquete.
 - c) Implante el nuevo procedimiento ADD_EMPLOYEE en el cuerpo del paquete, de la siguiente forma:
 - i) Formatee la dirección de correo electrónico en caracteres en mayúscula, utilizando la primera letra del nombre concatenado con las siete primeras letras del apellido.
 - ii) El procedimiento llamará al procedimiento ADD_EMPLOYEE existente para realizar la operación INSERT real utilizando los parámetros y el correo electrónico formateado para proporcionar los valores.
 - iii) Haga clic en Run Script para crear el paquete. Compile el paquete.
 - d) Llame al nuevo procedimiento ADD_EMPLOYEE utilizando el nombre Samuel Joplin para agregarlo al departamento 30.
 - e) Confirme que el nuevo empleado se ha agregado a la tabla EMPLOYEES.
- 2) En el paquete EMP_PKG, cree dos funciones sobrecargadas denominadas GET_EMPLOYEE:
 - a) En la especificación del paquete, agregue las siguientes funciones:
 - i) La función GET_EMPLOYEE que acepta el parámetro denominado p_emp_id basado en el tipo employees.employee_id%TYPE. Esta función debe devolver EMPLOYEES%ROWTYPE.
 - ii) La función GET_EMPLOYEE que acepta el parámetro denominado p_family_name de tipo employees.last_name%TYPE. Esta función debe devolver EMPLOYEES%ROWTYPE.

Práctica 4-1: Trabajar con Paquetes (continuación)

- b) Haga clic en Run Script para volver a crear y compilar el paquete.
- c) En el cuerpo del paquete:
- i) Implante la primera función GET_EMPLOYEE para consultar un empleado mediante el identificador del mismo.
 - ii) Implante la segunda función GET_EMPLOYEE para utilizar el operador de igualdad en el valor suministrado en el parámetro p_family_name.
- d) Haga clic en Run Script para volver a crear y compilar el paquete.
- e) Agregue un procedimiento de utilidad PRINT_EMPLOYEE al paquete EMP_PKG, de la siguiente forma:
- i) El procedimiento acepta EMPLOYEES%ROWTYPE como parámetro.
 - ii) El procedimiento muestra lo siguiente para un empleado en una línea, mediante el paquete DBMS_OUTPUT:
- department_id
 - employee_id
 - first_name
 - last_name
 - job_id
 - salary
- f) Haga clic en Run Script (F5) para crear y compilar el paquete.
- g) Utilice un bloque anónimo para llamar a la función EMP_PKG.GET_EMPLOYEE con un identificador de empleado 100 y con apellido 'Joplin'. Utilice el procedimiento PRINT_EMPLOYEE para mostrar los resultados para cada fila devuelta.
- 3) Como la compañía no cambia con frecuencia sus datos de departamento, puede mejorar el rendimiento de EMP_PKG agregando un procedimiento público, INIT_DEPARTMENTS, para llenar una tabla PL/SQL privada de identificadores de departamento válidos. Modifique la función VALID_DEPTID para utilizar el contenido de la tabla PL/SQL privada con el fin de validar los valores de los identificadores de departamento.
- Nota:** el script del archivo de soluciones sol_04_03.sql contiene el código de los pasos a, b y c.
- a) En la especificación del paquete, cree un procedimiento denominado INIT_DEPARTMENTS sin parámetros. Para ello, agregue lo siguiente a la sección de especificación del paquete antes de la especificación PRINT_EMPLOYEES:

```
PROCEDURE init_departments;
```

 - b) En el cuerpo del paquete, implante el procedimiento INIT_DEPARTMENTS para almacenar todos los identificadores de departamento en una tabla de índice PL/SQL privada denominada valid_departments que contiene valores BOOLEAN.

Práctica 4-1: Trabajar con Paquetes (continuación)

- i) Declare la variable `valid_departments` y su definición de tipo `boolean_tab_type` antes que todos los procedimientos del cuerpo. Introduzca lo siguiente al comienzo del cuerpo del paquete:

```
TYPE boolean_tab_type IS TABLE OF BOOLEAN
INDEX BY BINARY_INTEGER;
valid_departments boolean_tab_type;
```

- ii) Utilice el valor de la columna `department_id` como índice para crear la entrada en la tabla de índice para indicar su presencia y asignar a la entrada un valor de `TRUE`. Introduzca la declaración del procedimiento `INIT_DEPARTMENTS` al final del cuerpo del paquete (justo después del procedimiento `print_employees`), de la siguiente forma:

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

- c) En el cuerpo, cree un bloque de inicialización que llame al procedimiento `INIT_DEPARTMENTS` para inicializar la tabla, de la siguiente forma:
- ```
BEGIN
 init_departments;
END;
```
- d) Haga clic en Run Script (F5) para crear y compilar el paquete.
- 4) Cambie la función de procesamiento de validación `VALID_DEPTID` para utilizar la tabla PL/SQL privada de identificadores de departamento.
- Modifique la función `VALID_DEPTID` para realizar la validación utilizando la tabla PL/SQL de valores de identificadores de departamento. Haga clic en Run Script (F5) para crear el paquete. Compile el paquete.
  - Pruebe el código llamando a `ADD_EMPLOYEE` con el nombre James Bond en el departamento 15. ¿Qué sucede?
  - Inserte un nuevo departamento. Especifique 15 como identificador del departamento y 'Security' como nombre del departamento. Confirme y verifique los cambios.
  - Pruebe el código llamando a `ADD_EMPLOYEE` con el nombre James Bond en el departamento 15. ¿Qué sucede?
  - Ejecute el procedimiento `EMP_PKG.INIT_DEPARTMENTS` para actualizar la tabla interna PL/SQL con los últimos datos del departamento.
  - Pruebe el código llamando a `ADD_EMPLOYEE` con el nombre de empleado James Bond, que trabaja en el departamento 15. ¿Qué sucede?

**Práctica 4-1: Trabajar con Paquetes (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. Asegúrese de introducir SET SERVEROUTPUT ON antes.
- 5) Reorganice los subprogramas en el cuerpo de la especificación del paquete para que estén en secuencia alfabética.
- Edite la especificación del paquete y reorganice los subprogramas de forma alfabética. Haga clic en Run Script para volver a crear la especificación del paquete. Compile la especificación del paquete. ¿Qué sucede?
  - Edite el cuerpo del paquete y reorganice todos los subprogramas de forma alfabética. Haga clic en Run Script para volver a crear la especificación del paquete. Recompile la especificación del paquete. ¿Qué sucede?
  - Corrija el error de compilación utilizando una declaración anticipada en el cuerpo para la referencia de subprograma adecuada. Haga clic en Run Script para volver a crear el paquete y, a continuación, recompile el paquete. ¿Qué sucede?

## Soluciones a la Práctica 4-1: Trabajar con Paquetes

En esta práctica, modificará el código del paquete EMP\_PKG que ha creado anteriormente y, a continuación, sobrecargará el procedimiento ADD\_EMPLOYEE. A continuación, creará dos funciones sobrecargadas denominadas GET\_EMPLOYEE en el paquete EMP\_PKG. También agregará un procedimiento público a EMP\_PKG para llenar una tabla PL/SQL privada de identificadores de departamento válidos, además de modificar la función VALID\_DEPTID para utilizar el contenido de la tabla PL/SQL privada con el fin de validar los valores de identificador válidos. También cambiará la función de procesamiento de validación VALID\_DEPTID para utilizar la tabla PL/SQL privada de identificadores de departamento. Por último, reorganizará los subprogramas en el cuerpo y la especificación del paquete para que estén en secuencia alfabética.

- 1) Modifique el código del paquete EMP\_PKG que ha creado en la Práctica 4, paso 2 y sobrecargue el procedimiento ADD\_EMPLOYEE.
  - a) En la especificación del paquete, agregue un nuevo procedimiento denominado ADD\_EMPLOYEE, que acepte los tres parámetros siguientes:
    - i) First name
    - ii) Last name
    - iii) Department ID

Abra el archivo /home/oracle/labs/plpu/solns/sol\_04\_01\_a.sql. El código se muestra de la siguiente forma:

```

CREATE OR REPLACE PACKAGE emp_pkg IS
 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30);

 /* New overloaded add_employee */

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE);

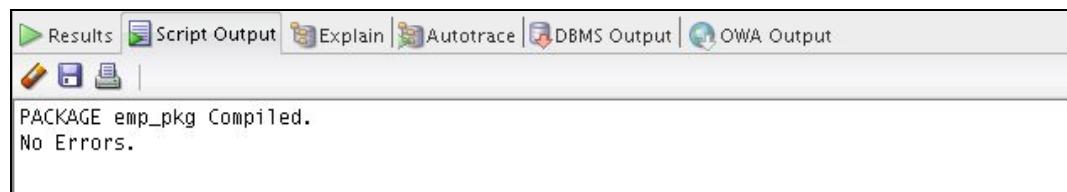
 PROCEDURE get_employee(
 p.empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE);
END emp_pkg;
/

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

SHOW ERRORS

- b) Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar el paquete.



- c) Implante el nuevo procedimiento ADD\_EMPLOYEE en el cuerpo del paquete, de la siguiente forma:

- Formatee la dirección de correo electrónico en caracteres en mayúscula, utilizando la primera letra del nombre concatenado con las siete primeras letras del apellido.
- El procedimiento debe llamar al procedimiento ADD\_EMPLOYEE existente para realizar la operación INSERT real utilizando los parámetros y el correo electrónico formateado para proporcionar los valores.
- Haga clic en Run Script para crear el paquete. Compile el paquete.

**Abra el script /home/oracle/labs/plpu/solns/sol\_04\_01\_c.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para llamar al procedimiento del paquete. El código y el resultado se muestran de la siguiente forma (el código que se acaba de agregar se resalta en negrita en el cuadro de texto siguiente):**

```

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

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

p_last_name employees.last_name%TYPE,
p_email employees.email%TYPE,
p_job employees.job_id%TYPE DEFAULT 'SA_REP',
p_mgr employees.manager_id%TYPE DEFAULT 145,
p_sal employees.salary%TYPE DEFAULT 1000,
p_comm employees.commission_pct%TYPE DEFAULT 0,
p_deptid employees.department_id%TYPE DEFAULT 30) IS

BEGIN
 IF valid_deptid(p_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, p_first_name, p_last_name,
 p_email, p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
 p_deptid);
 ELSE
 RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID. Try
 again.');
 END IF;
END add_employee;

/* New overloaded add_employee procedure */

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

/* End declaration of the overloaded add_employee procedure */

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

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

PACKAGE emp\_pkg Compiled.  
No Errors.  
PACKAGE BODY emp\_pkg Compiled.  
No Errors.

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

**Abra el script /home/oracle/labs/plpu/solns/sol\_04\_01\_d.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para llamar al procedimiento del paquete. El código y el resultado se muestran de la siguiente forma:**

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

anonymous block completed

- e) Confirme que el nuevo empleado se ha agregado a la tabla EMPLOYEES.

Abra el script /home/oracle/labs/plpu/solns/sol\_04\_01\_e.sql. Haga clic en cualquier parte de la sentencia SELECT y, a continuación, haga clic en el icono Execute Statement (F9) de la barra de herramientas de SQL Worksheet para ejecutar la consulta. El código y el resultado se muestran de la siguiente forma:

```
SELECT *
FROM employees
WHERE last_name = 'Joplin';
```

| Results  |             |            |           |         |              |           |        |        |                |            |               |
|----------|-------------|------------|-----------|---------|--------------|-----------|--------|--------|----------------|------------|---------------|
| Results: |             |            |           |         |              |           |        |        |                |            |               |
|          | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL   | PHONE_NUMBER | HIRE_DATE | JOB_ID | SALARY | COMMISSION_PCT | MANAGER_ID | DEPARTMENT_ID |
| 1        | 209         | Samuel     | Joplin    | SJOPLIN | (null)       | 17-JUN-09 | SA_REP | 1000   | 0              | 145        | 30            |

- 2) En el paquete EMP\_PKG, cree dos funciones sobrecargadas denominadas GET\_EMPLOYEE:
- En la especificación del paquete, agregue las siguientes funciones:

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

- i) La función GET\_EMPLOYEE que acepta el parámetro denominado p\_emp\_id basado en el tipo employees.employee\_id%TYPE. Esta función debe devolver EMPLOYEES%ROWTYPE.
- ii) La función GET\_EMPLOYEE que acepta el parámetro denominado p\_family\_name de tipo employees.last\_name%TYPE. Esta función debe devolver EMPLOYEES%ROWTYPE.

Abra el script /home/oracle/labs/plpu/sols/sol\_04\_02\_a.sql.

```

CREATE OR REPLACE PACKAGE emp_pkg IS
 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30);

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE);

 PROCEDURE get_employee(
 p.empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE);

/* New overloaded get_employees functions specs starts here: */
 FUNCTION get_employee(p.emp_id employees.employee_id%type)
 return employees%rowtype;

 FUNCTION get_employee(p.family_name employees.last_name%type)
 return employees%rowtype;

/* New overloaded get_employees functions specs ends here. */

END emp_pkg;
/
SHOW ERRORS

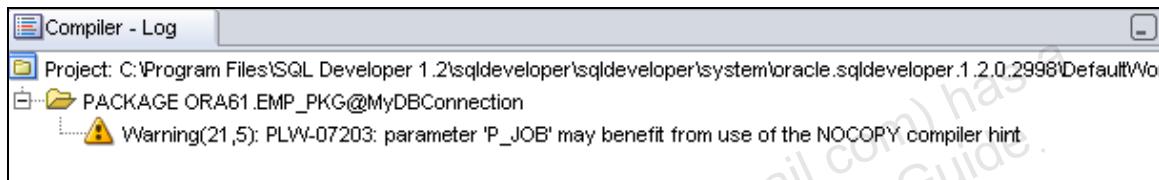
```

- b) Haga clic en Run Script para volver a crear y compilar la especificación del paquete.
- Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para volver a crear y compilar la especificación del paquete. A continuación se muestra el resultado:**

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

PACKAGE emp\_pkg Compiled.  
No Errors.

**Nota:** como se menciona anteriormente, si el código contiene un mensaje de error, puede recompilarlo mediante el siguiente procedimiento para ver los detalles del error o la advertencia en el separador Compiler – Log. Para compilar la especificación del paquete, haga clic con el botón derecho en el nombre de la especificación del paquete (o en todo el paquete) en el árbol Object Navigator y, a continuación, seleccione Compile en el menú de acceso directo. La advertencia se esperaba y es sólo para su información.



- c) En el cuerpo del paquete:
    - i) Implante la primera función GET\_EMPLOYEE para consultar un empleado mediante el identificador del mismo.
    - ii) Implante la segunda función GET\_EMPLOYEE para utilizar el operador de igualdad en el valor suministrado en el parámetro p\_family\_name.
- Abra el script /home/oracle/labs/plpu/solns/sol\_04\_02\_c.sql. Las funciones que se acaban de agregar se resaltan en el siguiente cuadro de código.**

```
CREATE OR REPLACE PACKAGE emp_pkg IS
 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30);

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE);
```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

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

/* New overloaded get_employees functions specs starts here: */

FUNCTION get_employee(p_emp_id employees.employee_id%type)
 return employees%rowtype;

FUNCTION get_employee(p_family_name employees.last_name%type)
 return employees%rowtype;

/* New overloaded get_employees functions specs ends here. */

END emp_pkg;
/
SHOW ERRORS

-- package body

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

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30) IS
 BEGIN
 IF valid_deptid(p_deptid) THEN
 INSERT INTO employees(employee_id, first_name, last_name,
 email, job_id, manager_id, hire_date, salary,
 commission_pct, department_id)

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
 p_email, p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
 p_deptid);
ELSE
 RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
 Try again.');
END IF;
END add_employee;

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

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

/* New get_employee function declaration starts here */

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

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

```

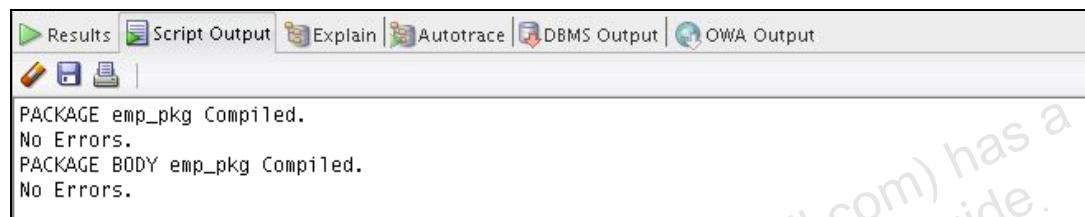
## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```
/* New overloaded get_employee function declaration ends here */

END emp_pkg;
/
SHOW ERRORS
```

- d) Haga clic en Run Script para volver a crear el paquete. Compile el paquete.

**Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para volver a crear y compilar el paquete. A continuación se muestra el resultado:**



The screenshot shows the Oracle SQL Worksheet interface with the 'Results' tab selected. The output window displays the following message:

```
PACKAGE emp_pkg Compiled.
No Errors.
PACKAGE BODY emp_pkg Compiled.
No Errors.
```

- e) Agregue un procedimiento de utilidad PRINT\_EMPLOYEE al paquete EMP\_PKG, de la siguiente forma:
- El procedimiento acepta EMPLOYEES%ROWTYPE como parámetro.
  - El procedimiento muestra lo siguiente para un empleado en una línea, mediante el paquete DBMS\_OUTPUT:

- department\_id
- employee\_id
- first\_name
- last\_name
- job\_id
- salary

**Abra el script /home/oracle/labs/plpu/solns/sol\_04\_02\_e.sql. El código que se acaba de agregar se resalta en el siguiente cuadro de código.**

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

p_deptid employees.department_id%TYPE DEFAULT 30);

PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
 p.empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p_emp_id employees.employee_id%type)
 return employees%rowtype;

FUNCTION get_employee(p_family_name
employees.last_name%type)
 return employees%rowtype;

/* New print_employee print_employee procedure spec */

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

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

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30) IS

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

BEGIN
 IF valid_deptid(p_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, p_first_name,
p_last_name, p_email,
 p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
 ELSE
 RAISE APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
 END IF;
END add_employee;

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

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

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

FUNCTION get_employee(p_family_name
employees.last_name%type)
 return employees%rowtype IS
 rec_emp employees%rowtype;

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

BEGIN
 SELECT * INTO rec_emp
 FROM employees
 WHERE last_name = p_family_name;
 RETURN rec_emp;
END;

/* New print_employees procedure declaration. */

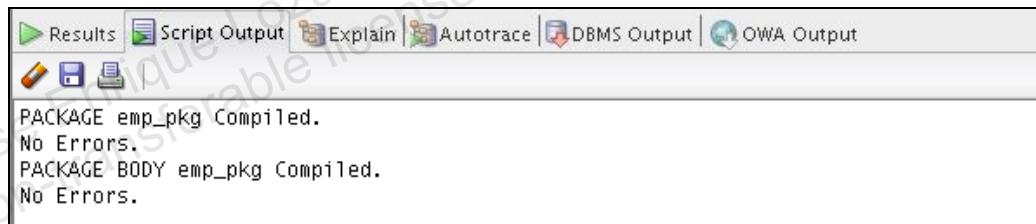
PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
 DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
 p_rec_emp.employee_id || ' ' ||
 p_rec_emp.first_name || ' ' ||
 p_rec_emp.last_name || ' ' ||
 p_rec_emp.job_id || ' ' ||
 p_rec_emp.salary);
END;

END emp_pkg;
/
SHOW ERRORS

```

- f) Haga clic en Run Script (F5) para crear y compilar el paquete.

**Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para volver a crear y compilar el paquete.**



The screenshot shows the Oracle SQL Worksheet interface with the following output:

```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
[Icons]
PACKAGE emp_pkg Compiled.
No Errors.
PACKAGE BODY emp_pkg Compiled.
No Errors.

```

- g) Utilice un bloque anónimo para llamar a la función EMP\_PKG.GET\_EMPLOYEE con un identificador de empleado 100 y con apellido 'Joplin'. Utilice el procedimiento PRINT\_EMPLOYEE para mostrar los resultados para cada fila devuelta. Asegúrese de introducir SET SERVEROUTPUT ON antes.

**Abra el script /home/oracle/labs/plpu/solns/sol\_04\_02\_g.sql.**

```

SET SERVEROUTPUT ON
BEGIN
 emp_pkg.print_employee(emp_pkg.get_employee(100));
 emp_pkg.print_employee(emp_pkg.get_employee('Joplin'));
END;
/

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
| |
anonymous block completed
90 100 Steven King AD_PRES 24000
30 209 Samuel Joplin SA_REP 1000

```

- 3) Como la compañía no cambia con frecuencia sus datos de departamento, puede mejorar el rendimiento de `EMP_PKG` agregando un procedimiento público, `INIT_DEPARTMENTS`, para llenar una tabla PL/SQL privada de identificadores de departamento válidos. Modifique la función `VALID_DEPTID` para utilizar el contenido de la tabla PL/SQL privada con el fin de validar los valores de los identificadores de departamento.

**Nota:** el script del archivo de soluciones `sol_04_03.sql` contiene el código de los pasos a, b y c.

- En la especificación del paquete, cree un procedimiento denominado `INIT_DEPARTMENTS` sin parámetros. Para ello, agregue lo siguiente a la sección de especificación del paquete antes de la especificación `PRINT_EMPLOYEES`:

```

PROCEDURE init_departments;

```

- En el cuerpo del paquete, implante el procedimiento `INIT_DEPARTMENTS` para almacenar todos los identificadores de departamento en una tabla de índice PL/SQL privada denominada `valid_departments` que contiene valores BOOLEAN.

- Declare la variable `valid_departments` y su definición de tipo `boolean_tab_type` antes que todos los procedimientos del cuerpo. Introduzca lo siguiente al comienzo del cuerpo del paquete:

```

TYPE boolean_tab_type IS TABLE OF BOOLEAN
INDEX BY BINARY_INTEGER;
valid_departments boolean_tab_type;

```

- Utilice el valor de la columna `department_id` como índice para crear la entrada en la tabla de índice para indicar su presencia y asignar a la entrada un valor de TRUE. Introduzca la declaración del procedimiento `INIT_DEPARTMENTS` al final del cuerpo del paquete (justo después del procedimiento `print_employees`), de la siguiente forma:

```

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

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

- c) En el cuerpo, cree un bloque de inicialización que llame al procedimiento INIT\_DEPARTMENTS para inicializar la tabla, de la siguiente forma:

```
BEGIN
 init_departments;
END;
```

**Abra el script /home/oracle/labs/plpu/solns/sol\_04\_03.sql. El código que se acaba de agregar se resalta en el siguiente cuadro de código.**

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30);

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE);

 PROCEDURE get_employee(
 p.empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE);

 FUNCTION get_employee(p.emp_id employees.employee_id%type)
 return employees%rowtype;

 FUNCTION get_employee(p.family_name
employees.last_name%type)
 return employees%rowtype;

/* New procedure init_departments spec */

 PROCEDURE init_departments;

 PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY
```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

CREATE OR REPLACE PACKAGE BODY emp_pkg IS

/* New type */

TYPE boolean_tab_type IS TABLE OF BOOLEAN
 INDEX BY BINARY_INTEGER;
 valid_departments boolean_tab_type;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
 v_dummy PLS_INTEGER;
BEGIN
 SELECT 1
 INTO v_dummy
 FROM departments
 WHERE department_id = p_deptid;
 RETURN TRUE;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RETURN FALSE;
END valid_deptid;

PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
 IF valid_deptid(p_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, p_first_name, p_last_name,
 p_email, p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
 p_deptid);
 ELSE
 RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
 Try again.');
 END IF;
END add_employee;

PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE) IS
 p_email employees.email%TYPE;

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

BEGIN
 p_email := UPPER(SUBSTR(p_first_name, 1,
1) || SUBSTR(p_last_name, 1, 7));
 add_employee(p_first_name, p_last_name, p_email, p_deptid
=> p_deptid);
END;

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

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

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

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

/* New init_departments procedure declaration. */

PROCEDURE init_departments IS
BEGIN

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

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

/* call the new init_departments procedure. */

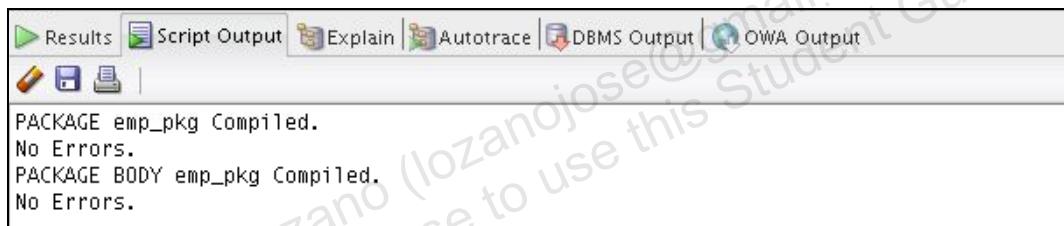
BEGIN
 init_departments;
END emp_pkg;

/
SHOW ERRORS

```

- d) Haga clic en Run Script (F5) para volver a crear y compilar el paquete.

**Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para volver a crear y compilar el paquete.**



- 4) Cambie la función de procesamiento de validación VALID\_DEPTID para utilizar la tabla PL/SQL privada de identificadores de departamento.
- Modifique la función VALID\_DEPTID para realizar la validación utilizando la tabla PL/SQL de valores de identificadores de departamento. Haga clic en Run Script (F5) para crear y compilar el paquete.

**Abra el script /home/oracle/labs/plpu/solns/sol\_04\_04\_a.sql. Haga clic en Run Script (F5) para crear y compilar el paquete. El código que se acaba de agregar se resalta en el siguiente cuadro de código.**

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

 p_deptid employees.department_id%TYPE DEFAULT 30);

PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
 p_empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p_emp_id employees.employee_id%type)
 return employees%rowtype;

FUNCTION get_employee(p_family_name
 employees.last_name%type)
 return employees%rowtype;

/* New procedure init_departments spec */

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS

TYPE boolean_tab_type IS TABLE OF BOOLEAN
 INDEX BY BINARY_INTEGER;
valid_departments boolean_tab_type;

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

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

p_email employees.email%TYPE,
p_job employees.job_id%TYPE DEFAULT 'SA_REP',
p_mgr employees.manager_id%TYPE DEFAULT 145,
p_sal employees.salary%TYPE DEFAULT 1000,
p_comm employees.commission_pct%TYPE DEFAULT 0,
p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
 IF valid_deptid(p_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, p_first_name,
 p_last_name, p_email,
 p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,p_deptid);
 ELSE
 RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
 Try again.');
 END IF;
END add_employee;

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

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

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

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

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

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

/* New init_departments procedure declaration. */

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

/* call the new init_departments procedure. */

BEGIN
 init_departments;
END emp_pkg;

/
SHOW ERRORS

```



The screenshot shows the Oracle SQL Developer interface with the 'Script Output' tab selected. The output window displays the following messages:

```

Results Script Output Explain Autotrace DBMS Output OWA Output
PACKAGE emp_pkg Compiled.
No Errors.
PACKAGE BODY emp_pkg Compiled.
No Errors.

```

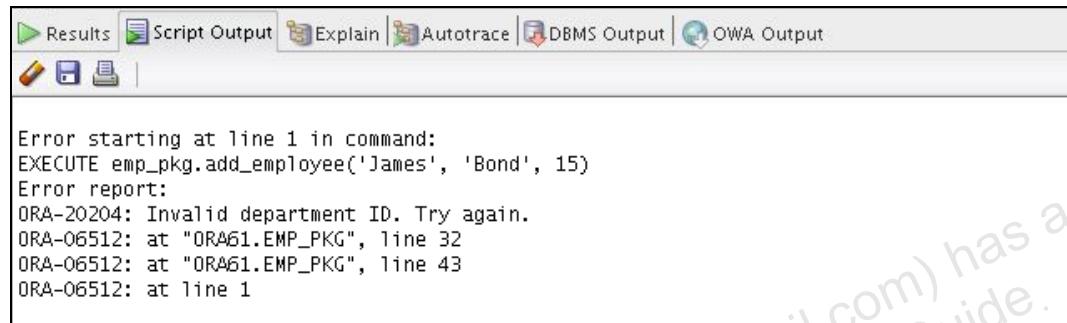
- b) Pruebe el código llamando a ADD\_EMPLOYEE con el nombre James Bond en el departamento 15. ¿Qué sucede?

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

Abra el script `/home/oracle/labs/plpu/solns/sol_04_04_b.sql`.

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

Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para probar la inserción de un nuevo empleado. La operación de inserción para agregar un empleado ha fallado con una excepción, porque el departamento 15 no existe.



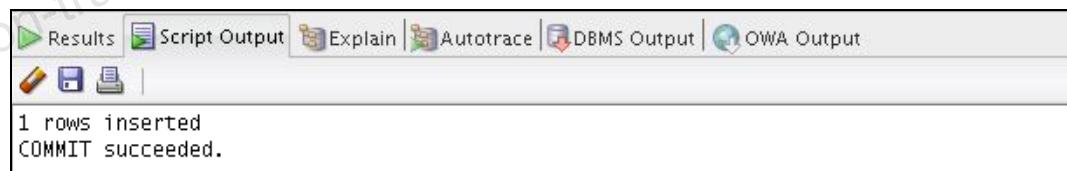
The screenshot shows the Oracle SQL Worksheet interface. The 'Script Output' tab is selected. The output window displays the following error message:

```
Error starting at line 1 in command:
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)
Error report:
ORA-2004: Invalid department ID. Try again.
ORA-06512: at "ORA61.EMP_PKG", line 32
ORA-06512: at "ORA61.EMP_PKG", line 43
ORA-06512: at line 1
```

- Inserte un nuevo departamento. Especifique 15 como identificador del departamento y 'Security' como nombre del departamento. Confirme y verifique los cambios.

Abra el script `/home/oracle/labs/plpu/solns/sol_04_04_c.sql`. El código y el resultado se muestran de la siguiente forma:

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



The screenshot shows the Oracle SQL Worksheet interface. The 'Script Output' tab is selected. The output window displays the following message:

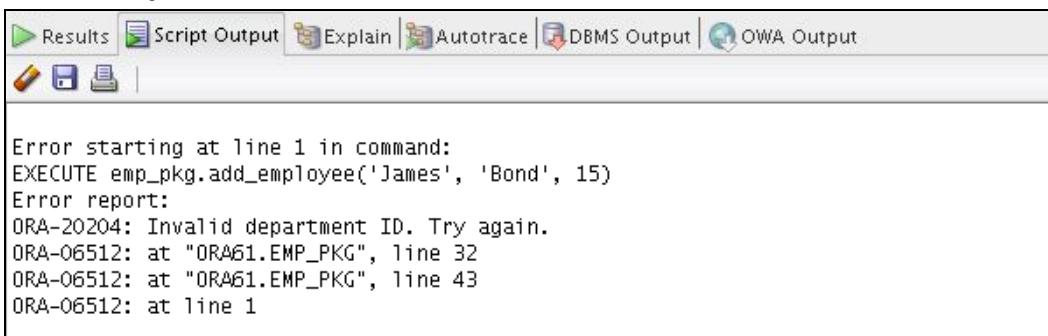
```
1 rows inserted
COMMIT succeeded.
```

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

Abra el script `/home/oracle/labs/plpu/solns/sol_04_04_d.sql`. El código y el resultado se muestran de la siguiente forma:

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

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)



```
Error starting at line 1 in command:
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)
Error report:
ORA-20204: Invalid department ID. Try again.
ORA-06512: at "ORA61.EMP_PKG", line 32
ORA-06512: at "ORA61.EMP_PKG", line 43
ORA-06512: at line 1
```

**La operación de inserción para agregar un empleado falla con una excepción. El departamento 15 no existe como entrada en la variable de estado de paquete de (la tabla de índice) de la matriz asociativa PL/SQL.**

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

**Abra el script /home/oracle/labs/plpu/solns/sol\_04\_04\_e.sql. El código y el resultado se muestran de la siguiente forma:**

```
EXECUTE EMP_PKG.INIT_DEPARTMENTS
```



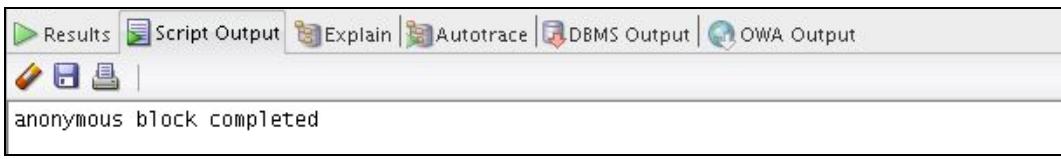
```
anonymous block completed
```

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

**Abra el script /home/oracle/labs/plpu/solns/sol\_04\_04\_f.sql. El código y el resultado se muestran de la siguiente forma.**

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

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



```
anonymous block completed
```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (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.

**Abra el script /home/oracle/labs/plpu/solns/sol\_04\_04\_g.sql. El código y el resultado se muestran de la siguiente forma.**

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

```
Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
| |
1 rows deleted
1 rows deleted
COMMIT succeeded.
anonymous block completed
```

- 5) Reorganice los subprogramas en el cuerpo y la especificación del paquete para que estén en secuencia alfabética.
- Edite la especificación del paquete y reorganice los subprogramas de forma alfabética. Haga clic en Run Script para volver a crear la especificación del paquete. Compile la especificación del paquete. ¿Qué sucede?
- Abra el script /home/oracle/labs/plpu/solns/sol\_04\_05\_a.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para volver a crear y compilar el paquete. El código y el resultado se muestran de la siguiente forma. Los subprogramas de la especificación del paquete ya están en orden alfabético.**

```
CREATE OR REPLACE PACKAGE emp_pkg IS

/* the package spec is already in an alphabetical order. */

PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30);
```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
 p.empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p.emp_id
employees.employee_id%type)
 return employees%rowtype;

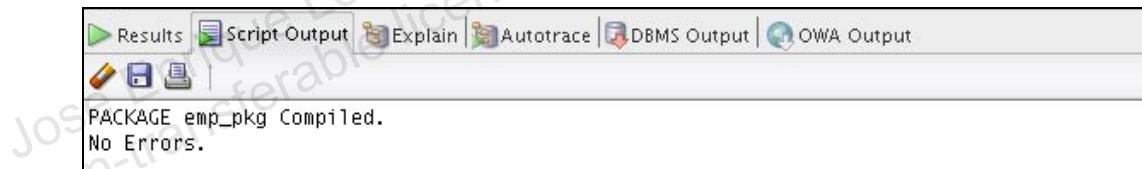
FUNCTION get_employee(p.family_name
employees.last_name%type)
 return employees%rowtype;

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

```



- b) Edite el cuerpo del paquete y reorganice todos los subprogramas de forma alfabética. Haga clic en Run Script para volver a crear la especificación del paquete. Recompile la especificación del paquete. ¿Qué sucede?

**Abra el script /home/oracle/labs/plpu/solns/sol\_04\_05\_b.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para volver a crear el paquete. El código y el resultado se muestran de la siguiente forma.**

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

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
 IF valid_deptid(p_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, p_first_name,
p_last_name, p_email,
 p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
 ELSE
 RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID. Try again.');
 END IF;
END add_employee;

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

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

FUNCTION get_employee(p.emp_id
employees.employee_id%type)
return employees%rowtype IS
 rec_emp employees%rowtype;
BEGIN

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

SELECT * INTO rec_emp
FROM employees
WHERE employee_id = p_emp_id;
RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name
employees.last_name%type)
return employees%rowtype IS
rec_emp employees%rowtype;
BEGIN
SELECT * INTO rec_emp
FROM employees
WHERE last_name = p_family_name;
RETURN rec_emp;
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(p_rec_emp employees%rowtype) IS
BEGIN
DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id ||' ' ||
p_rec_emp.employee_id||' ' ||
p_rec_emp.first_name||' ' ||
p_rec_emp.last_name||' ' ||
p_rec_emp.job_id||' ' ||
p_rec_emp.salary);
END;

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

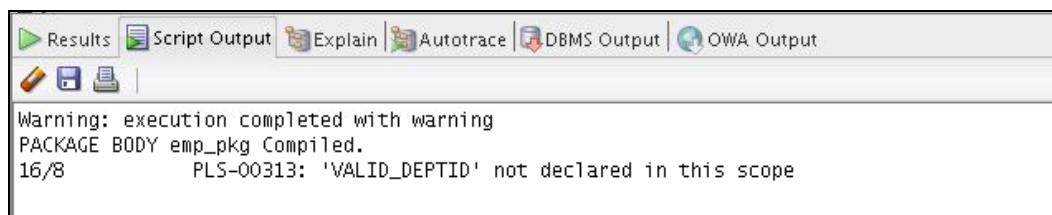
BEGIN
init_departments;
END emp_pkg;

/
SHOW ERRORS

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

El paquete no se compila correctamente porque se hace referencia a la función VALID\_DEPTID antes de que se declare.



The screenshot shows the Oracle SQL Worksheet interface with the 'Script Output' tab selected. The output window displays the following message:

```
Warning: execution completed with warning
PACKAGE BODY emp_pkg Compiled.
16/8 PLS-00313: 'VALID_DEPTID' not declared in this scope
```

- c) Corrija el error de compilación utilizando una declaración anticipada en el cuerpo para la referencia de subprograma adecuada. Haga clic en Run Script para volver a crear el paquete y, a continuación, recompile el paquete. ¿Qué sucede?

Abra el script /home/oracle/labs/plpu/solns/sol\_04\_05\_c.sql. La declaración anticipada de la función aparece resaltada en el cuadro de código siguiente. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para volver a crear y compilar el paquete. El código y el resultado se muestran de la siguiente forma.

```
-- Package BODY

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

/* forward declaration of valid_deptid */

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

PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
 IF valid_deptid(p_deptid) THEN /* valid_deptid function
referenced */
 INSERT INTO employees(employee_id, first_name,
last_name, email,
 job_id, manager_id, hire_date, salary, commission_pct,
department_id)
```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

 VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
 p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
 ELSE
 RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
 END IF;
END add_employee;

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

PROCEDURE get_employee(
 p_empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE) IS
BEGIN
 SELECT salary, job_id
 INTO p_sal, p_job
 FROM employees
 WHERE employee_id = p_empid;
END get_employee;

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

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

```

## Soluciones a la Práctica 4-1: Trabajar con Paquetes (continuación)

```

/* New alphabetical location of function init_departments. */

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

/* New alphabetical location of function valid_deptid. */

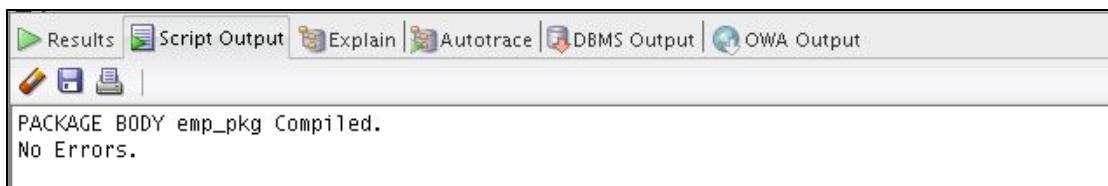
FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
 v_dummy PLS_INTEGER;
BEGIN
 RETURN valid_departments.exists(p_deptid);
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RETURN FALSE;
END valid_deptid;

BEGIN
 init_departments;
END emp_pkg;

/
SHOW ERRORS

```

**Una declaración anticipada de la función VALID\_DEPTID permite al cuerpo del paquete compilarse correctamente, como se muestra a continuación:**



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

- Toolbar icons: Results, Script Output, Explain, Autotrace, DBMS Output, OWA Output.
- Message area: PACKAGE BODY emp\_pkg Compiled. No Errors.

## Prácticas y Soluciones de la Lección 5

En esta práctica, utilizará el paquete UTL\_FILE para generar un informe de archivo de texto de los empleados de cada departamento.

**Nota:** si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

## Práctica 5-1: Uso del Paquete UTL\_FILE

En esta práctica, utilizará el paquete UTL\_FILE para generar un informe de archivo de texto de los empleados de cada departamento. En primer lugar, creará y ejecutará 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. Por último, verá el archivo de texto de la salida generada.

- 1) Cree un procedimiento denominado EMPLOYEE\_REPORT, que genera un informe de empleados en un archivo del sistema operativo utilizando el paquete UTL\_FILE. Este informe generará una lista de los empleados que han excedido el salario medio de su departamento.
  - a) El programa debe aceptar dos parámetros. El primero es el directorio de salida. El segundo es el nombre del archivo de texto escrito.

**Nota:** utilice el valor de la ubicación del directorio UTL\_FILE. Agregue una sección de manejo de excepciones para manejar los errores que se pueden encontrar al utilizar el paquete UTL\_FILE.
  - b) Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar el procedimiento.
- 2) Llame al procedimiento con los dos argumentos siguientes:
  - a) Utilice REPORTS\_DIR como alias para el objeto de directorio como primer parámetro.
  - b) Utilice sal\_rpt61.txt como segundo parámetro.
- 3) Vea el archivo de texto de la salida generada, de la siguiente forma:
  - a) Haga clic dos veces en el ícono **Terminal** del escritorio. Se mostrará la ventana **Terminal**.
  - b) En la petición de datos \$, cambie a la carpeta **/home/oracle/labs/p1pu/reports** que contenga el archivo de salida generado, sal\_rpt61.txt mediante el comando cd.

**Nota:** puede utilizar el comando pwd para enumerar el directorio de trabajo actual.

## Soluciones a la Práctica 5-1: Uso del Paquete UTL\_FILE

En esta práctica, utilizará el paquete UTL\_FILE para generar un informe de archivo de texto de los empleados de cada departamento. En primer lugar, creará y ejecutará 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. Por último, verá el archivo de texto de la salida generada.

- 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.

**Abra el archivo en el script /home/oracle/labs/plpu/solns/sol\_05\_01.sql.**

```
-- Verify with your instructor that the database initSID.ora
-- file has the directory path you are going to use with this
-- procedure.
-- For example, there should be an entry such as:
-- UTL_FILE_DIR = /home1/teachX/UTL_FILE in your initSID.ora
-- (or the SPFILE)
-- HOWEVER: The course has a directory alias provided called
-- "REPORTS_DIR" that is associated with an appropriate
-- directory. Use the directory alias name in quotes for the
-- first parameter to create a file in the appropriate
-- directory.

CREATE OR REPLACE PROCEDURE employee_report(
 p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
 f UTL_FILE.FILE_TYPE;
 CURSOR cur_avg 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(p_dir, p_filename, 'W');

```

## Soluciones a la Práctica 5-1: Uso del Paquete UTL\_FILE (continuación)

```

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 cur_avg
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;
/

```

- b) Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar el procedimiento.



- 2) Llame al procedimiento con los argumentos siguientes:
- Utilice REPORTS\_DIR como alias para el objeto de directorio como primer parámetro.
  - Utilice sal\_rpt61.txt como segundo parámetro.
- Abra el script /home/oracle/labs/plpu/solns/sol\_05\_02.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el procedimiento. A continuación se muestra el resultado. Asegúrese de que el archivo externo y la base de datos están en la misma computadora.

```
-- For example, if you are student ora61, use 61 as a prefix
EXECUTE employee_report('REPORTS_DIR','sal_rpt61.txt')
```

## **Soluciones a la Práctica 5-1: Uso del Paquete UTL\_FILE (continuación)**

- 3) Vea el archivo de texto de la salida generada, de la siguiente forma:
- Haga clic dos veces en el ícono **Terminal** del escritorio. Se mostrará la ventana **Terminal**.
  - En la petición de datos \$, cambie a la carpeta **/home/oracle/labs/plpu/reports** que contenga el archivo de salida generado, **sal\_rpt61.txt** mediante el comando **cd**, de la siguiente forma:

```
[oracle@EDRSR13P1 ~]$cd labs/plpu/reports
[oracle@EDRSR13P1 reports]$pwd
/home/oracle/labs/plpu/reports
[oracle@EDRSR13P1 reports]$
```

**Nota:** puede utilizar el comando **pwd** para enumerar el directorio de trabajo actual en la captura de pantalla anterior.

- Enumere el contenido del directorio actual mediante el comando **ls**, de la siguiente forma:

```
[oracle@EDRSR13P1 ~]$cd labs/plpu/reports
[oracle@EDRSR13P1 reports]$pwd
/home/oracle/labs/plpu/reports
[oracle@EDRSR13P1 reports]$ls
salary_report.sql sal_rpt61.txt
[oracle@EDRSR13P1 reports]$
```

**Observe el archivo de salida generado, **sal\_rpt61.txt**.**

- Abra el archivo transferido **sal\_rpt61.txt**, mediante **gedit** o mediante el editor que desee. El informe se muestra de la siguiente forma:

```
[oracle@EDRSR13P1 ~]$cd labs/plpu/reports
[oracle@EDRSR13P1 reports]$pwd
/home/oracle/labs/plpu/reports
[oracle@EDRSR13P1 reports]$ls
salary_report.sql sal_rpt61.txt
[oracle@EDRSR13P1 reports]$gedit sal_rpt61.txt
```

## **Soluciones a la Práctica 5-1: Uso del Paquete UTL\_FILE (continuación)**

| Employees who earn more than average salary:<br>REPORT GENERATED ON 18-JUN-09 |     |             |
|-------------------------------------------------------------------------------|-----|-------------|
| Hartstein                                                                     | 20  | \$13,000.00 |
| Raphaely                                                                      | 30  | \$11,000.00 |
| Mavris                                                                        | 40  | \$6,500.00  |
| Weiss                                                                         | 50  | \$8,000.00  |
| Fripp                                                                         | 50  | \$8,200.00  |
| Kaufling                                                                      | 50  | \$7,900.00  |
| Vollman                                                                       | 50  | \$6,500.00  |
| Hunold                                                                        | 60  | \$9,000.00  |
| Baer                                                                          | 70  | \$10,000.00 |
| Cambrault                                                                     | 80  | \$11,000.00 |
| Zlotkey                                                                       | 80  | \$10,500.00 |
| Tucker                                                                        | 80  | \$10,000.00 |
| Bernstein                                                                     | 80  | \$9,500.00  |
| Hall                                                                          | 80  | \$9,000.00  |
| Olsen                                                                         | 80  | \$8,000.00  |
| Cambrault                                                                     | 80  | \$7,500.00  |
| Tuvault                                                                       | 80  | \$7,000.00  |
| King                                                                          | 80  | \$10,000.00 |
| Sully                                                                         | 80  | \$9,500.00  |
| McEwen                                                                        | 80  | \$9,000.00  |
| Smith                                                                         | 80  | \$8,000.00  |
| Doran                                                                         | 80  | \$7,500.00  |
| Sewall                                                                        | 80  | \$7,000.00  |
| Vishney                                                                       | 80  | \$10,500.00 |
| Greene                                                                        | 80  | \$9,500.00  |
| Marvins                                                                       | 80  | \$7,200.00  |
| Lee                                                                           | 80  | \$6,800.00  |
| <b>...</b>                                                                    |     |             |
| Kochhar                                                                       | 90  | \$17,000.00 |
| Urman                                                                         | 100 | \$7,800.00  |
| Sciarra                                                                       | 100 | \$7,700.00  |
| Chen                                                                          | 100 | \$8,200.00  |
| Faviet                                                                        | 100 | \$9,000.00  |
| Greenberg                                                                     | 100 | \$12,000.00 |
| Popp                                                                          | 100 | \$6,900.00  |
| Higgins                                                                       | 110 | \$12,000.00 |
| Gietz                                                                         | 110 | \$8,300.00  |
| Grant                                                                         | -   | \$7,000.00  |
| <b>*** END OF REPORT ***</b>                                                  |     |             |

## Prácticas y Soluciones de la Lección 6

En esta práctica, creará un paquete que utiliza SQL dinámico nativo para crear o borrar una tabla y para llenar, modificar y suprimir filas de una tabla. Además, creará un paquete que compile código PL/SQL en el esquema, ya sea todo el código PL/SQL o aquel que tenga un estado `INVALID` en la tabla `USER_OBJECTS`.

**Nota:** si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

## Práctica 6-1: Uso de SQL Dinámico Nativo

En esta práctica, creará un paquete que utiliza SQL dinámico nativo para crear o borrar una tabla y para llenar, modificar y suprimir filas de una tabla. Además, creará un paquete que compile código PL/SQL en el esquema, ya sea todo el código PL/SQL o aquel que tenga un estado INVALID en la tabla USER\_OBJECTS.

- 1) Cree un paquete denominado TABLE\_PKG que utilice SQL dinámico nativo para crear o borrar una tabla y para llenar, modificar y suprimir filas de la tabla. Los subprogramas deben gestionar los parámetros por defecto opcionales con valores NULL.

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

```
PROCEDURE make(p_table_name VARCHAR2, p_col_specs VARCHAR2)
PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
 VARCHAR2, p_cols VARCHAR2 := NULL)
PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
 VARCHAR2, p_conditions VARCHAR2 := NULL)
PROCEDURE del_row(p_table_name VARCHAR2,
 p_conditions VARCHAR2 := NULL);
PROCEDURE remove(p_table_name VARCHAR2)
```

- 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 para el procedimiento remove, que se debe escribir con el paquete DBMS\_SQL.
- c) Ejecute el procedimiento empaquetado MAKE para crear una tabla, de la siguiente forma:

```
make('my_contacts', 'id number(4), name
varchar2(40)');
```

- d) Describa la estructura de la tabla MY\_CONTACTS.
- e) Ejecute el procedimiento empaquetado ADD\_ROW para agregar las filas siguientes. Active SERVEROUTPUT.

```
add_row('my_contacts','1,''Lauran Serhal''','id, name');
add_row('my_contacts','2,''Nancy''','id, name');
add_row('my_contacts','3,''Sunitha Patel''','id, name');
add_row('my_contacts','4,''Valli Pataballa''','id, name');
```

- f) Consulte el contenido de la tabla MY\_CONTACTS para verificar las adiciones.
- g) Ejecute el procedimiento empaquetado DEL\_ROW para suprimir un contacto con un valor de identificador 3.
- h) Ejecute el procedimiento UPD\_ROW con los siguientes datos de fila:

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

## **Práctica 6-1: Uso de SQL Dinámico Nativo (continuación)**

- i) Consulte el contenido de la tabla MY\_CONTACTS para verificar los cambios.
- j) Borre la tabla utilizando el procedimiento remove y describa la tabla MY\_CONTACTS.
- 2) Cree un paquete COMPILE\_PKG que compile el código PL/SQL en el esquema.
- a) En la especificación, cree un procedimiento empaquetado denominado MAKE, que acepte el nombre de la unidad de programa PL/SQL que se va a compilar.
- b) En el cuerpo del paquete, incluya lo siguiente:
- i) El procedimiento EXECUTE utilizado en el procedimiento TABLE\_PKG del paso 1 de esta práctica.
  - ii) Una función privada denominada GET\_TYPE para determinar el tipo de objeto PL/SQL del diccionario de datos.
    - La función devuelve el nombre de tipo (utilice PACKAGE para un paquete con un cuerpo) si existe el objeto; en caso contrario, debe devolver NULL.
    - En la condición de la cláusula WHERE, agregue lo siguiente a la condición para garantizar que sólo se devuelva una fila si el nombre representa un PACKAGE, que también puede tener un PACKAGE\_BODY. En este caso, sólo puede compilar el paquete completo, pero no la especificación ni el cuerpo como componentes independientes:

```
rownum = 1
```
  - iii) Cree el procedimiento MAKE mediante la siguiente información:
    - El procedimiento MAKE acepta un argumento, name, que representa el nombre del objeto.
    - El procedimiento MAKE debe llamar a la función GET\_TYPE. Si el objeto existe, MAKE lo compila dinámicamente con la sentencia ALTER.
- c) Utilice el procedimiento COMPILE\_PKG.MAKE para compilar lo siguiente:
- i) El procedimiento EMPLOYEE\_REPORT
  - ii) El paquete EMP\_PKG
  - iii) Un objeto no existente denominado EMP\_DATA

## Soluciones a la Práctica 6-1: Uso de SQL Dinámico Nativo

En esta práctica, creará un paquete que utiliza SQL dinámico nativo para crear o borrar una tabla y para llenar, modificar y suprimir filas de una tabla. Además, creará un paquete que compile código PL/SQL en el esquema, ya sea todo el código PL/SQL o aquel que tenga un estado INVALID en la tabla USER\_OBJECTS.

- 1) Cree un paquete denominado TABLE\_PKG que utilice SQL dinámico nativo para crear o borrar una tabla y para llenar, modificar y suprimir filas de la tabla. Los subprogramas deben gestionar los parámetros por defecto opcionales con valores NULL.

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

```

PROCEDURE make(p_table_name VARCHAR2, p_col_specs VARCHAR2)
PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
 VARCHAR2, p_cols VARCHAR2 := NULL)
PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
 VARCHAR2, p_conditions VARCHAR2 := NULL)
PROCEDURE del_row(p_table_name VARCHAR2,
 p_conditions VARCHAR2 := NULL);
PROCEDURE remove(p_table_name VARCHAR2)

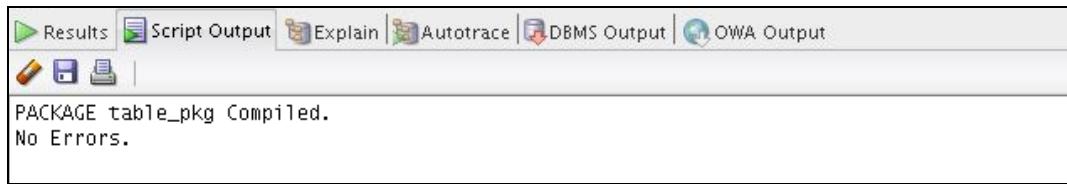
```

**Abra el script /home/oracle/labs/plpu/solns/sol\_06\_01\_a.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar la especificación del paquete. El código y el resultado se muestran de la siguiente forma:**

```

CREATE OR REPLACE PACKAGE table_pkg IS
 PROCEDURE make(p_table_name VARCHAR2, p_col_specs
 VARCHAR2);
 PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
 VARCHAR2, p_cols VARCHAR2 := NULL);
 PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
 VARCHAR2, p_conditions VARCHAR2 := NULL);
 PROCEDURE del_row(p_table_name VARCHAR2, p_conditions
 VARCHAR2 := NULL);
 PROCEDURE remove(p_table_name VARCHAR2);
END table_pkg;
/
SHOW ERRORS

```



## Soluciones a la Práctica 6-1: Uso de SQL Dinámico Nativo (continuación)

- 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 para el procedimiento remove, que se debe escribir con el paquete DBMS\_SQL.

Abra el script /home/oracle/labs/plpu/solns/sol\_06\_01\_b.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar la especificación del paquete. El código y el resultado se muestran a continuación.

```

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

 PROCEDURE make(p_table_name VARCHAR2, p_col_specs VARCHAR2)
 IS
 v_stmt VARCHAR2(200) := 'CREATE TABLE '|| p_table_name ||
 ' (' || p_col_specs || ')';
 BEGIN
 execute(v_stmt);
 END;

 PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
 VARCHAR2, p_cols VARCHAR2 := NULL) IS
 v_stmt VARCHAR2(200) := 'INSERT INTO '|| p_table_name;
 BEGIN
 IF p_cols IS NOT NULL THEN
 v_stmt := v_stmt || ' (' || p_cols || ')';
 END IF;
 v_stmt := v_stmt || ' VALUES (' || p_col_values || ')';
 execute(v_stmt);
 END;

 PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
 VARCHAR2, p_conditions VARCHAR2 := NULL) IS

 v_stmt VARCHAR2(200) := 'UPDATE '|| p_table_name || ' SET ' ||
 p_set_values;
 BEGIN
 IF p_conditions IS NOT NULL THEN
 v_stmt := v_stmt || ' WHERE ' || p_conditions;
 END IF;
 execute(v_stmt);
 END;

```

## Soluciones a la Práctica 6-1: Uso de SQL Dinámico Nativo (continuación)

```

PROCEDURE del_row(p_table_name VARCHAR2, p_conditions
 VARCHAR2 := NULL) IS
BEGIN
 v_stmt VARCHAR2(200) := 'DELETE FROM '|| p_table_name;
 IF p_conditions IS NOT NULL THEN
 v_stmt := v_stmt || ' WHERE ' || p_conditions;
 END IF;
 execute(v_stmt);
END;

PROCEDURE remove(p_table_name VARCHAR2) IS
 cur_id INTEGER;
 v_stmt VARCHAR2(100) := 'DROP TABLE '||p_table_name;
BEGIN
 cur_id := DBMS_SQL.OPEN_CURSOR;
 DBMS_OUTPUT.PUT_LINE(v_stmt);
 DBMS_SQLPARSE(cur_id, v_stmt, DBMS_SQL.NATIVE);
 -- Parse executes DDL statements, no EXECUTE is required.
 DBMS_SQLCLOSE_CURSOR(cur_id);
END;

END table_pkg;
/
SHOW ERRORS

```

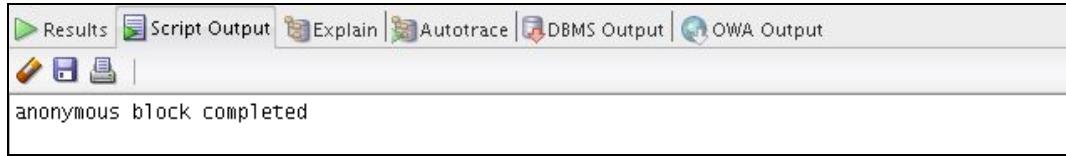


- c) Ejecute el procedimiento empaquetado MAKE para crear una tabla, de la siguiente forma:

```
make('my_contacts', 'id number(4), name
 varchar2(40)');
```

**Abra el script /home/oracle/labs/plpu/sols/sol\_06\_01\_c.sql.**  
**Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear la especificación del paquete. El código y el resultado se muestran de la siguiente forma:**

```
EXECUTE table_pkg.make('my_contacts', 'id number(4), name
 varchar2(40)')
```



**Soluciones a la Práctica 6-1: Uso de SQL Dinámico Nativo (continuación)**

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

**El código y el resultado se muestran de la siguiente forma:**

```
MyDBConnection | 1.21718299 seconds | MyD
| DESCRIBE my_contacts
|
| Results Script Output Explain Autotrace DBMS Output OWA Output
|
| DESCRIBE my_contacts
Name Null Type
ID NUMBER(4)
NAME VARCHAR2(40)
2 rows selected
```

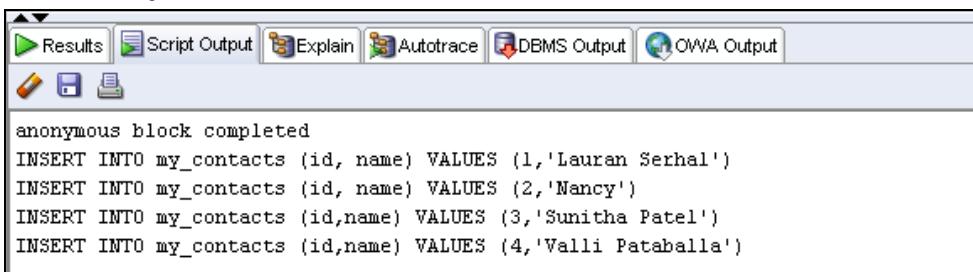
- e) Ejecute el procedimiento empaquetado ADD\_ROW para agregar las filas siguientes. Active SERVEROUTPUT.

```
add_row('my_contacts','1','Lauran Serhal','','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');
```

Abra el script `/home/oracle/labs/plpu/solns/sol_06_01_e.sql`.  
Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script.

```
1 SET SERVEROUTPUT ON
2
3 BEGIN
4 table_pkg.add_row('my_contacts','1','Lauran Serhal','');
5 table_pkg.add_row('my_contacts','2','Nancy','');
6 table_pkg.add_row('my_contacts','3','Sunitha Patel','');
7 table_pkg.add_row('my_contacts','4','Valli Pataballa','');
8 END;
9 /
10
```

## Soluciones a la Práctica 6-1: Uso de SQL Dinámico Nativo (continuación)



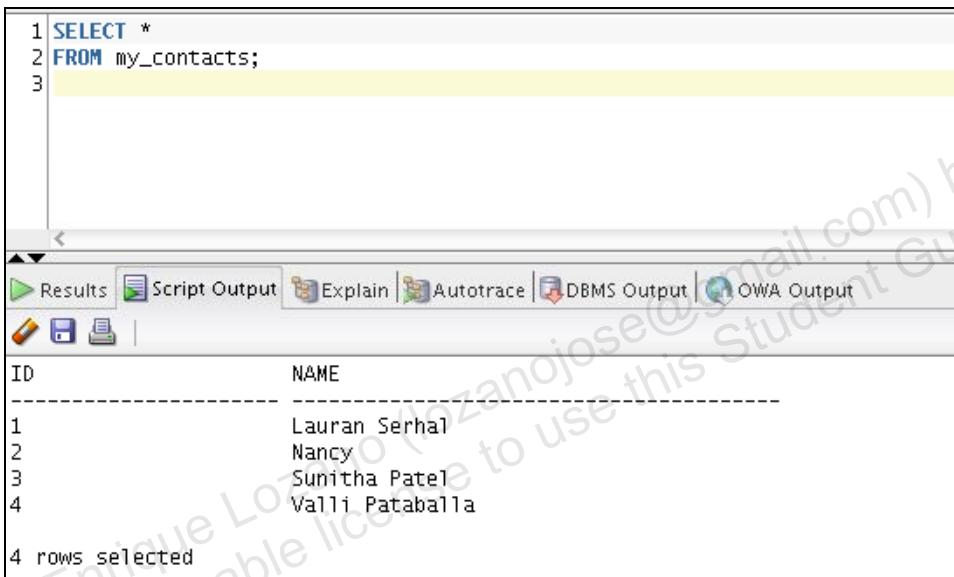
```

anonymous block completed
INSERT INTO my_contacts (id, name) VALUES (1,'Lauran Serhal')
INSERT INTO my_contacts (id, name) VALUES (2,'Nancy')
INSERT INTO my_contacts (id,name) VALUES (3,'Sunitha Patel')
INSERT INTO my_contacts (id,name) VALUES (4,'Valli Pataballa')

```

- f) Consulte el contenido de la tabla MY\_CONTACTS para verificar las adiciones.

**El código y el resultado se muestran de la siguiente forma:**



```

1 SELECT *
2 FROM my_contacts;
3

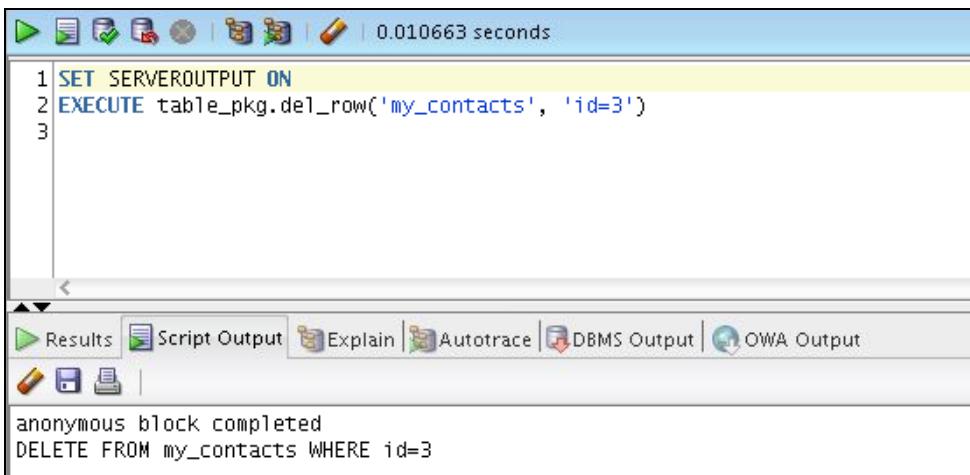
```

| ID | NAME            |
|----|-----------------|
| 1  | Lauran Serhal   |
| 2  | Nancy           |
| 3  | Sunitha Patel   |
| 4  | Valli Pataballa |

4 rows selected

- g) Ejecute el procedimiento empaquetado DEL\_ROW para suprimir un contacto con un valor de identificador 3.

**El código y el resultado se muestran de la siguiente forma:**



```

1 SET SERVEROUTPUT ON
2 EXECUTE table_pkg.del_row('my_contacts', 'id=3')
3

```

```

anonymous block completed
DELETE FROM my_contacts WHERE id=3

```

## Soluciones a la Práctica 6-1: Uso de SQL Dinámico Nativo (continuación)

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

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

**El código y el resultado se muestran de la siguiente forma:**

The screenshot shows an Oracle SQL Worksheet interface. The code entered is:

```
1 SET SERVEROUTPUT ON
2 EXEC table_pkg.upd_row('my_contacts', 'name='''Nancy Greenberg'''', 'id=2')
3
```

The output pane shows the results of the execution:

```
anonymous block completed
UPDATE my_contacts SET name='Nancy Greenberg' WHERE id=2
```

- i) Consulte el contenido de la tabla MY\_CONTACTS para verificar los cambios.

**El código y el resultado se muestran de la siguiente forma:**

The screenshot shows an Oracle SQL Worksheet interface. The code entered is:

```
1 SELECT *
2 FROM my_contacts;
3
```

The output pane shows the results of the query:

| ID | NAME            |
|----|-----------------|
| 1  | Lauran Serhal   |
| 2  | Nancy Greenberg |
| 4  | Valli Pataballa |

3 rows selected

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

## Soluciones a la Práctica 6-1: Uso de SQL Dinámico Nativo (continuación)

El código y el resultado se muestran de la siguiente forma:

```

1 EXECUTE table_pkg.remove('my_contacts')
2 DESCRIBE my_contacts
3

anonymous block completed
DESCRIBE my_contacts
ERROR:

ERROR: object MY_CONTACTS does not exist

1 rows selected

```

- 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 la unidad de programa PL/SQL que se va a compilar.  
Abra el script /home/oracle/labs/plpu/solns/sol\_06\_02\_a.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar la especificación del paquete. El código y el resultado se muestran a continuación.

```

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

```

## Soluciones a la Práctica 6-1: Uso de SQL Dinámico Nativo (continuación)

The screenshot shows the results tab with the message: PACKAGE compile\_pkg Compiled. No Errors.

- b) En el cuerpo del paquete, incluya lo siguiente:
- El procedimiento EXECUTE utilizado en el procedimiento TABLE\_PKG del paso 1 de esta práctica.
  - Una función privada denominada GET\_TYPE para determinar el tipo de objeto PL/SQL del diccionario de datos.
    - La función devuelve el nombre de tipo (utilice PACKAGE para un paquete con un cuerpo) si existe el objeto; en caso contrario, debe devolver NULL.
    - En la condición de la cláusula WHERE, agregue lo siguiente a la condición para garantizar que sólo se devuelva una fila si el nombre representa un PACKAGE, que también puede tener un PACKAGE BODY. En este caso, sólo puede compilar el paquete completo, pero no la especificación ni el cuerpo como componentes independientes:
 

```
rownum = 1
```
  - Cree el procedimiento MAKE mediante la siguiente información:
    - El procedimiento MAKE acepta un argumento, name, que representa el nombre del objeto.
    - El procedimiento MAKE debe llamar a la función GET\_TYPE. Si el objeto existe, MAKE lo compila dinámicamente con la sentencia ALTER.

**Abra el script /home/oracle/labs/plpu/solns/sol\_06\_02\_b.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar el cuerpo del paquete. El código y el resultado se muestran de la siguiente forma:**

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

## Soluciones a la Práctica 6-1: Uso de SQL Dinámico Nativo (continuación)

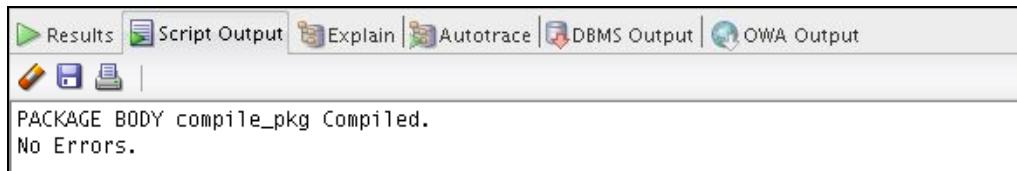
```

END;

FUNCTION get_type(p_name VARCHAR2) RETURN VARCHAR2 IS
 v_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 v_proc_type
 FROM user_objects
 WHERE object_name = UPPER(p_name)
 AND ROWNUM = 1;
 RETURN v_proc_type;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RETURN NULL;
END;

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

```



- c) Utilice el procedimiento COMPILE\_PKG.MAKE para compilar lo siguiente:
- El procedimiento EMPLOYEE\_REPORT
  - El paquete EMP\_PKG

## Soluciones a la Práctica 6-1: Uso de SQL Dinámico Nativo (continuación)

iii) Un objeto no existente denominado EMP\_DATA

Abra el archivo del script /home/oracle/labs/plpu/solns/sol\_06\_02\_c.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el procedimiento del paquete. El código y el resultado se muestran a continuación.

```
SET SERVEROUTPUT ON

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

The screenshot shows the Oracle SQL Worksheet interface. The script area contains the following code:

```
1 SET SERVEROUTPUT ON
2
3 EXECUTE compile_pkg.make('employee_report')
4 EXECUTE compile_pkg.make('emp_pkg')
5 EXECUTE compile_pkg.make('emp_data')
6
```

The output pane shows the results of the execution:

```
anonymous block completed
ALTER PROCEDURE employee_report COMPILE

anonymous block completed
ALTER PACKAGE emp_pkg COMPILE

Error starting at line 5 in command:
EXECUTE compile_pkg.make('emp_data')
Error report:
ORA-20001: Subprogram 'emp_data' does not exist
ORA-06512: at "ORA61.COMPILE_PKG", line 39
ORA-06512: at line 1
```

## Prácticas y Soluciones de la Lección 7

En esta práctica, creará un paquete que realice una recuperación en bloque de empleados de un departamento concreto. Los datos se almacenarán en una tabla PL/SQL del paquete. También proporcionará un procedimiento para visualizar el contenido de la tabla. Además, creará el procedimiento `add_employee`, que inserta nuevos empleados. El procedimiento utilizará un subprograma autónomo local para escribir un registro log cada vez que se llame al procedimiento `add_employee`, tanto si agrega correctamente un registro como si no.

**Nota:** si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

## Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas

En esta práctica, creará un paquete que realice una recuperación en bloque de empleados de un departamento concreto. Los datos se almacenarán en una tabla PL/SQL del paquete. También proporcionará un procedimiento para visualizar el contenido de la tabla. Además, creará el procedimiento `add_employee`, que inserta nuevos empleados. El procedimiento utilizará un subprograma autónomo local para escribir un registro log cada vez que se llame al procedimiento `add_employee`, tanto si agrega correctamente un registro como si no.

- 1) Actualice el paquete `EMP_PKG` con un nuevo procedimiento para consultar los empleados de un departamento especificado.
  - a) En la especificación del paquete:
    - i) Declare un procedimiento `get_employees` con un parámetro denominado `dept_id`, que se basa en el tipo de columna `employees.department_id`
    - ii) Defina un tipo de índice PL/SQL como `TABLE OF EMPLOYEES%ROWTYPE`
  - b) En el cuerpo del paquete:
    - i) Defina una variable privada denominada `emp_table`, basada en el tipo definido en la especificación para almacenar los registros de los empleados
    - ii) Implante el procedimiento `get_employees` para recuperar en bloque los datos en la tabla
  - c) Cree un nuevo procedimiento en la especificación y el cuerpo denominado `show_employees`, que no toma argumentos. El procedimiento muestra el contenido de la variable de tabla PL/SQL privada (si existen datos). Utilice el procedimiento `print_employee` que ha creado en una práctica anterior. Para ver los resultados, haga clic en el ícono Enable DBMS Output del separador DBMS Output en SQL Developer, en caso de que no lo haya hecho aún.
  - d) Active SERVEROUTPUT. Llame al procedimiento `emp_pkg.get_employees` para el departamento 30 y, a continuación, llame a `emp_pkg.show_employees`. Repítalo para el departamento 60.
- 2) Su superior 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 script `/home/oracle/labs/plpu/solns/sol_07_02_a.sql` para crear una tabla de log denominada `LOG_NEWEMP` y una secuencia denominada `log_newemp_seq`.

## Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

- b) En el cuerpo del paquete EMP\_PKG, modifique el procedimiento add\_employee, que realiza la operación real INSERT. Agregue un procedimiento local denominado audit\_newemp, de la siguiente forma:
- El procedimiento audit\_newemp debe utilizar una transacción autónoma para insertar un registro log en la tabla LOG\_NEWEMP.
  - Almacene el USER, la hora actual y el nombre del nuevo empleado en la fila de la tabla de log.
  - Utilice log\_newemp\_seq para definir la columna entry\_id.

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

- c) Modifique el procedimiento add\_employee para llamar a audit\_emp antes de que realice la operación de inserción.
- d) Llame al procedimiento add\_employee para los siguientes nuevos empleados: Max Smart del departamento 20 y Clark Kent del departamento 10. ¿Qué sucede?
- c) Consulte los dos registros EMPLOYEES que se han agregado y los registros de la tabla LOG\_NEWEMP. ¿Cuántos registros log están presentes?
- d) Ejecute una sentencia ROLLBACK para deshacer las operaciones de inserción que no se han confirmado. Utilice las mismas consultas desde el paso 2 e., como se muestra a continuación:
- Utilice la primera consulta para comprobar si las filas de empleados de Smart y Kent se han eliminado.
  - Utilice la segunda consulta para comprobar los registros log de la tabla LOG\_NEWEMP. ¿Cuántos registros log están presentes? ¿Por qué?

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas

En esta práctica, creará un paquete que realice una recuperación en bloque de empleados de un departamento concreto. Los datos se almacenarán en una tabla PL/SQL del paquete. También proporcionará un procedimiento para visualizar el contenido de la tabla. Además, creará el procedimiento `add_employee`, que inserta nuevos empleados. El procedimiento utilizará un subprograma autónomo local para escribir un registro log cada vez que se llame al procedimiento `add_employee`, tanto si agrega correctamente un registro como si no.

- 1) Actualice el paquete `EMP_PKG` con un nuevo procedimiento para consultar los empleados de un departamento especificado.
  - a) En la especificación del paquete:
    - i) Declare un procedimiento `get_employees` con un parámetro denominado `dept_id`, que se basa en el tipo de columna `employees.department_id`
    - ii) Defina un tipo de índice PL/SQL como `TABLE OF EMPLOYEES%ROWTYPE`

Abra el script `/home/oracle/labs/plpu/solns/sol_07_01_a.sql`.

Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar la especificación. El código y el resultado se muestran de la siguiente forma. El código que se acaba de agregar se resalta en negrita en el cuadro de código siguiente.

```

CREATE OR REPLACE PACKAGE emp_pkg IS

 TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30);

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE);

 PROCEDURE get_employee(
 p_empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE);

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

FUNCTION get_employee(p_emp_id employees.employee_id%type)
 return employees%rowtype;

FUNCTION get_employee(p_family_name
employees.last_name%type)
 return employees%rowtype;

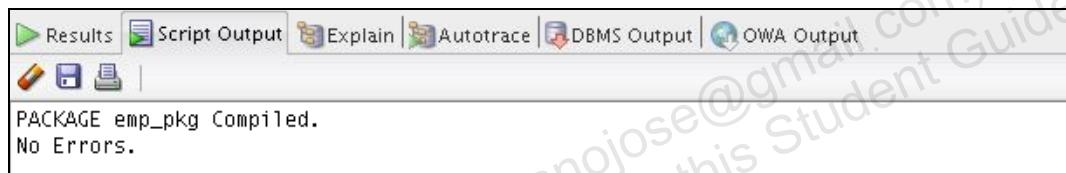
PROCEDURE get_employees(p_dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW 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
  - Implante el procedimiento `get_employees` para recuperar en bloque los datos en la tabla

Abra el script `/home/oracle/labs/plpu/solns/sol_07_01_b.sql`. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar el cuerpo del paquete. El código y el resultado se muestran a continuación. El código que se acaba de agregar se resalta en negrita en el cuadro de código siguiente.

```

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

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA REP',

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

p_mgr employees.manager_id%TYPE DEFAULT 145,
p_sal employees.salary%TYPE DEFAULT 1000,
p_comm employees.commission_pct%TYPE DEFAULT 0,
p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
 IF valid_deptid(p_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, p_first_name,
p_last_name, p_email,
 p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
 ELSE
 RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
 END IF;
END add_employee;

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

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

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

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

FUNCTION get_employee(p_family_name
employees.last_name%type)
 return employees%rowtype IS

rec_emp employees%rowtype;
BEGIN
 SELECT * INTO rec_emp
 FROM employees
 WHERE last_name = p_family_name;
 RETURN rec_emp;
END;

/* New get_employees procedure. */

PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
BEGIN
 SELECT * BULK COLLECT INTO emp_table
 FROM EMPLOYEES
 WHERE department_id = p_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(p_rec_emp employees%rowtype) IS
BEGIN
 DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
 p_rec_emp.employee_id|| ' ' ||
 p_rec_emp.first_name|| ' ' ||
 p_rec_emp.last_name|| ' ' ||
 p_rec_emp.job_id|| ' ' ||
 p_rec_emp.salary);
END;

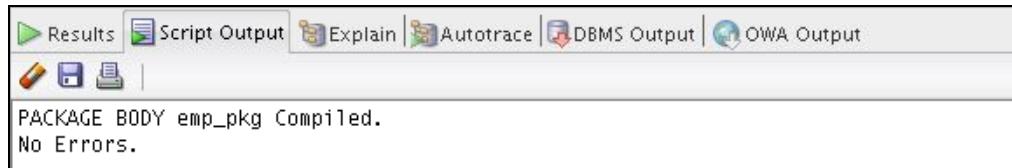
FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
 v_dummy PLS_INTEGER;
BEGIN
 RETURN valid_departments.exists(p_deptid);
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RETURN FALSE;
END valid_deptid;

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```
BEGIN
 init_departments;

END emp_pkg;
/
SHOW ERRORS
```



- c) Cree un nuevo procedimiento en la especificación y el cuerpo denominado show\_employees, que no toma argumentos. El procedimiento muestra el contenido de la variable de tabla PL/SQL privada (si existen datos). Utilice el procedimiento print\_employee que ha creado en una práctica anterior. Para ver los resultados, haga clic en el ícono Enable DBMS Output del separador DBMS Output en SQL Developer, en caso de que no lo haya hecho aún.

**Abra el script /home/oracle/labs/plpu/solns/sol\_07\_01\_c.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para volver a crear y compilar el paquete con el nuevo procedimiento. El código y el resultado se muestran a continuación.**

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
 TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30);

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE);

 PROCEDURE get_employee(
 p_empid IN employees.employee_id%TYPE,
```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p_emp_id employees.employee_id%type)
 return employees%rowtype;

FUNCTION get_employee(p_family_name
employees.last_name%type)
 return employees%rowtype;

PROCEDURE get_employees(p_dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

PROCEDURE show_employees;

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
 TYPE boolean_tab_type IS TABLE OF BOOLEAN
 INDEX BY BINARY_INTEGER;

 valid_departments boolean_tab_type;
 emp_table emp_tab_type;
 FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
 RETURN BOOLEAN;

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30) IS
 BEGIN
 IF valid_deptid(p_deptid) THEN
 INSERT INTO employees(employee_id, first_name,
last_name, email,
 job_id, manager_id, hire_date, salary, commission_pct,
department_id)

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

 VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
 p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
 ELSE
 RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
 END IF;
 END add_employee;

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

PROCEDURE get_employee(
 p_empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE) IS
BEGIN
 SELECT salary, job_id
 INTO p_sal, p_job
 FROM employees
 WHERE employee_id = p_empid;
END get_employee;

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

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

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

 RETURN rec_emp;
 END;

 PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
 BEGIN
 SELECT * BULK COLLECT INTO emp_table
 FROM EMPLOYEES
 WHERE department_id = p_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(p_rec_emp employees%rowtype) IS
 BEGIN
 DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id||' | '
 p_rec_emp.employee_id||' | '
 p_rec_emp.first_name||' | '
 p_rec_emp.last_name||' | '
 p_rec_emp.job_id||' | '
 p_rec_emp.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(p_deptid IN
departments.department_id%TYPE)
 RETURN BOOLEAN IS
 v_dummy PLS_INTEGER;
 BEGIN
 RETURN valid_departments.exists(p_deptid);
 EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RETURN FALSE;
 END valid_deptid;

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```
BEGIN
 init_departments;
END emp_pkg;

/
SHOW ERRORS
```

The screenshot shows the Oracle SQL Worksheet interface with several tabs at the top: Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. Below the tabs, there are three rows of text output:

- PACKAGE emp\_pkg Compiled.
- No Errors.
- PACKAGE BODY emp\_pkg Compiled.
- No Errors.

- d) Active SERVEROUTPUT. 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.

**Abra el script /home/oracle/labs/plpu/solns/sol\_07\_01\_d.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para llamar a los procedimientos del paquete. El código y el resultado se muestran a continuación:**

```
SET SERVEROUTPUT ON

EXECUTE emp_pkg.get_employees(30)
EXECUTE emp_pkg.show_employees

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

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

Results Script Output Explain Autotrace DBMS Output OWA Output
anonymous block 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

anonymous block completed
anonymous block 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

```

- 2) Su superior 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`.
- En primer lugar, cargue y ejecute el script `/home/oracle/labs/plpu/solns/sol_07_02_a.sql` para crear una tabla de log denominada `LOG_NEWEMP` y una secuencia denominada `log_newemp_seq`. Abra el script `/home/oracle/labs/plpu/solns/sol_07_02_a.sql`. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.

```

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;

```

```

Results Script Output Explain Autotrace DBMS Output OWA Output
anonymous block 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

anonymous block completed
anonymous block 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

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

- b) En el cuerpo del paquete EMP\_PKG, modifique el procedimiento add\_employee, que realiza la operación real INSERT. Agregue un procedimiento local denominado audit\_newemp, de la siguiente forma:
- 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.

**Abra el script /home/oracle/labs/plpu/solns/sol\_07\_02\_b.sql. El código que se acaba de agregar se resalta en negrita en el cuadro de código siguiente. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran de la siguiente forma:**

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS

 TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30);

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE);

 PROCEDURE get_employee(
 p.empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE);

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

FUNCTION get_employee(p_emp_id
employees.employee_id%type)
 return employees%rowtype;

FUNCTION get_employee(p_family_name
employees.last_name%type)
 return employees%rowtype;

PROCEDURE get_employees(p_dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

PROCEDURE show_employees;

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
 TYPE boolean_tab_type IS TABLE OF BOOLEAN
 INDEX BY BINARY_INTEGER;

 valid_departments boolean_tab_type;
 emp_table emp_tab_type;

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

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
 p_deptid employees.department_id%TYPE DEFAULT 30) IS

-- New local procedure

 PROCEDURE audit_newemp IS
 PRAGMA AUTONOMOUS_TRANSACTION;
 user_id VARCHAR2(30) := USER;
 BEGIN
 INSERT INTO log_newemp (entry_id, user_id, log_time,

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

 name)
VALUES (log_newemp_seq.NEXTVAL, user_id,
 sysdate,p_first_name||' '||p_last_name);
 COMMIT;
END audit_newemp;

BEGIN -- add_employee
 IF valid_deptid(p_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, p_first_name,
p_last_name, p_email,
 p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
 ELSE
 RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID. Try again.');
 END IF;
END add_employee;

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

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

FUNCTION get_employee(p.emp_id
employees.employee_id%type)
 return employees%rowtype IS
 rec_emp employees%rowtype;
BEGIN
 SELECT * INTO rec_emp

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

 FROM employees
 WHERE employee_id = p_emp_id;
 RETURN rec_emp;
 END;

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

/* New get_employees procedure. */

 PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
 BEGIN
 SELECT * BULK COLLECT INTO emp_table
 FROM EMPLOYEES
 WHERE department_id = p_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(p_rec_emp employees%rowtype) IS
 BEGIN
 DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
 p_rec_emp.employee_id|| ' ' ||
 p_rec_emp.first_name|| ' ' ||
 p_rec_emp.last_name|| ' ' ||
 p_rec_emp.job_id|| ' ' ||
 p_rec_emp.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;

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

END show_employees;

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

BEGIN
 init_departments;
END emp_pkg;
/
SHOW ERRORS

```

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

- Toolbar buttons: Results, Script Output, Explain, Autotrace, DBMS Output, OWA Output.
- Script Editor area: Displays the package compilation results.
- Output area: Shows the following messages:
  - PACKAGE emp\_pkg Compiled.
  - No Errors.
  - PACKAGE BODY emp\_pkg Compiled.
  - No Errors.

- c) Modifique el procedimiento add\_employee para llamar a audit\_emp antes de que realice la operación de inserción.

**Abra el script /home/oracle/labs/plpu/solns/sol\_07\_02\_c.sql. El código que se acaba de agregar se resalta en negrita en el cuadro de código siguiente. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```

-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS

 TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

p_sal employees.salary%TYPE DEFAULT 1000,
p_comm employees.commission_pct%TYPE DEFAULT 0,
p_deptid employees.department_id%TYPE DEFAULT 30);

PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
 p.empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p.emp_id employees.employee_id%type)
 return employees%rowtype;

FUNCTION get_employee(p.family_name
employees.last_name%type)
 return employees%rowtype;

PROCEDURE get_employees(p_dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

PROCEDURE show_employees;

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
 TYPE boolean_tab_type IS TABLE OF BOOLEAN
 INDEX BY BINARY_INTEGER;

 valid_departments boolean_tab_type;
 emp_table emp_tab_type;

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

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

p_job employees.job_id%TYPE DEFAULT 'SA_REP',
p_mgr employees.manager_id%TYPE DEFAULT 145,
p_sal employees.salary%TYPE DEFAULT 1000,
p_comm employees.commission_pct%TYPE DEFAULT 0,
p_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,p_first_name||' '||p_last_name);
 COMMIT;
END audit_newemp;

BEGIN -- add_employee
 IF valid_deptid(p_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, p_first_name,
p_last_name, p_email,
 p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
 ELSE
 RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
 END IF;
 END add_employee;

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

PROCEDURE get_employee(
 p.empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE) IS
BEGIN

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

SELECT salary, job_id
INTO p_sal, p_job
FROM employees
WHERE employee_id = p.empid;
END get_employee;

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

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

PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
BEGIN
 SELECT * BULK COLLECT INTO emp_table
 FROM EMPLOYEES
 WHERE department_id = p_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(p_rec_emp employees%rowtype) IS
BEGIN
 DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
 p_rec_emp.employee_id|| ' ' ||
 p_rec_emp.first_name|| ' ' ||
 p_rec_emp.last_name|| ' ')

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

```

 p_rec_emp.job_id || ' ' ||
 p_rec_emp.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(p_deptid IN
departments.department_id%TYPE)
RETURN BOOLEAN IS
 v_dummy PLS_INTEGER;
BEGIN
 RETURN valid_departments.exists(p_deptid);
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RETURN FALSE;
END valid_deptid;
BEGIN
 init_departments;
END emp_pkg;
/
SHOW ERRORS

```

The screenshot shows the Oracle SQL Worksheet interface. At the top, there are tabs for Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. Below the tabs, there are icons for Run, Stop, and Save. The main area displays the following text:

PACKAGE emp\_pkg Compiled.

No Errors.

PACKAGE BODY emp\_pkg Compiled.

No Errors.

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

Abra el script /home/oracle/labs/plpu/solns/sol\_07\_02\_d.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado son los siguientes.

```

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

```

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

The screenshot shows the Oracle SQL Worksheet interface. The toolbar at the top includes 'Results', 'Script Output', 'Explain', 'Autotrace', 'DBMS Output', and 'OWA Output'. Below the toolbar, there are three small icons: a red pencil, a blue floppy disk, and a green printer. The main area displays the message 'anonymous block completed' twice.

**Ambas sentencias insert se han terminado correctamente. 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?

Abra el script /home/oracle/labs/plpu/solns/sol\_07\_02\_e.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran de la siguiente forma:

```
select department_id, employee_id, last_name, first_name
from employees
where last_name in ('Kent', 'Smart');

select * from log_newemp;
```

The screenshot shows the Oracle SQL Worksheet interface with the results of the executed script. The results are displayed in two tables. The first table shows employees with last names Kent and Smart. The second table shows log entries for those employees.

| DEPARTMENT_ID | EMPLOYEE_ID | LAST_NAME | FIRST_NAME |
|---------------|-------------|-----------|------------|
| 10            | 212         | Kent      | Clark      |
| 20            | 211         | Smart     | Max        |

2 rows selected

| ENTRY_ID | USER_ID | LOG_TIME  | NAME       |
|----------|---------|-----------|------------|
| 1        | ORA61   | 19-AUG-09 | Max Smart  |
| 2        | ORA61   | 19-AUG-09 | Clark Kent |

2 rows selected

**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 desde el paso 2 e., como se muestra a continuación:
- Utilice la primera consulta para comprobar si las filas de empleados de Smart y Kent se han eliminado.
  - Utilice la segunda consulta para comprobar los registros log de la tabla LOG\_NEWEMP. ¿Cuántos registros log están presentes? ¿Por qué?

ROLLBACK;

## Soluciones a la Práctica 7-1: Uso del Enlace en Bloque y de las Transacciones Autónomas (continuación)

The screenshot shows two sessions in Oracle SQL Developer. The top session shows a successful rollback:

```
ROLLBACK succeeded.
```

The bottom session shows the execution of two queries:

```
1 select department_id, employee_id, last_name, first_name
2 from employees
3 where last_name in ('Kent', 'Smart');
4
5 select * from log_newemp;
6
```

The results of the first query show no rows selected:

| DEPARTMENT_ID   | EMPLOYEE_ID | LAST_NAME | FIRST_NAME |
|-----------------|-------------|-----------|------------|
| -----           |             |           |            |
| 0 rows selected |             |           |            |

The results of the second query show two entries in the log table:

| ENTRY_ID | USER_ID | LOG_TIME  | NAME       |
|----------|---------|-----------|------------|
| 1        | ORA61   | 18-JUN-09 | Max Smart  |
| 2        | ORA61   | 18-JUN-09 | Clark Kent |

A watermark across the bottom of the screen reads: "Enrique Lozano (lozanojose@gmail.com) has a variable license to use this Student Guide."

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.

## Prácticas y Soluciones de la Lección 8

En esta práctica, creará disparadores de sentencia y de fila. También creará procedimientos que se llamarán desde los disparadores.

**Nota:** si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

## Práctica 8-1: Creación de Disparadores de Sentencia y de Fila

En esta práctica, creará disparadores de sentencia y de fila. También creará procedimientos que se llamarán desde los disparadores.

- 1) Las filas de la tabla JOBS almacenan los salarios mínimos y máximos permitidos para los distintos valores de JOB\_ID. Le piden que escriba un código para garantizar que el salario de los empleados esté dentro del rango permitido por su tipo de trabajo, para operaciones de inserción y actualización.
  - a) Cree un procedimiento denominado CHECK\_SALARY, de la siguiente forma:
    - i) El procedimiento acepta dos parámetros, uno para la cadena del identificador de trabajo del empleado y el otro para el salario.
    - ii) El procedimiento utiliza el identificador de trabajo para determinar el salario mínimo y máximo para el trabajo especificado.
    - iii) 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 llenados con consultas. Guarde el archivo.
  - b) Cree un disparador denominado CHECK\_SALARY\_TRG en la tabla EMPLOYEES que arranque ante una operación INSERT o UPDATE en cada fila:
    - i) El disparador debe llamar al procedimiento CHECK\_SALARY para ejecutar la lógica de negocio.
    - ii) El disparador debe transferir el nuevo identificador de trabajo y salario a los parámetros de procedimiento.
- 2) Pruebe el disparador CHECK\_SAL\_TRG utilizando los siguientes casos:
  - a) Utilice el procedimiento EMP\_PKG.ADD\_EMPLOYEE para agregar a la empleada Eleanor Beh al departamento 30. ¿Qué sucede? ¿Por qué?
  - b) Actualice el salario del empleado 115 a 2.000 dólares. En otra operación de actualización, cambie el identificador de trabajo del empleado a HRREP. ¿Qué sucede en cada caso?
  - c) Actualice el salario del empleado 115 a 2.800 dólares. ¿Qué sucede?
- 3) Actualice el disparador CHECK\_SALARY\_TRG para que arranque sólo cuando los valores del identificador de trabajo o el salario hayan cambiado en realidad.
  - a) Implante la regla de negocio utilizando una cláusula WHEN para comprobar si los valores JOB\_ID o SALARY han cambiado.

## Práctica 8-1: Creación de Disparadores de Sentencia y de Fila (continuación)

**Nota:** asegúrese de que la condición maneja NULL en los valores de OLD.column\_name si se realiza una operación INSERT; si no es así, la operación de inserción fallará.

- b) Compruebe el disparador ejecutando el procedimiento EMP\_PKG.ADD\_EMPLOYEE con los siguientes valores de parámetros:
  - p\_first\_name: 'Eleanor'
  - p\_last\_name: 'Beh'
  - p\_Email: 'EBEH'
  - p\_Job: 'IT\_PROG'
  - p\_Sal: 5000
- c) Actualice a los empleados con un trabajo IT\_PROG incrementando su salario en 2.000 dólares. ¿Qué sucede?
- d) Actualice a 9.000 dólares el salario de Eleanor Beh.

**Indicación:** utilice una sentencia UPDATE con una subconsulta en la cláusula WHERE. ¿Qué sucede?

- e) Cambie el trabajo de Eleanor Beh a ST\_MAN utilizando otra sentencia UPDATE con una subconsulta. ¿Qué sucede?
- 4) Se le pide que evite que se suprima a los empleados durante las horas laborables.
  - a) Escriba un disparador de sentencia denominado DELETE\_EMP\_TRG en la tabla EMPLOYEES para evitar que las filas se supriman durante horas laborables entre semana, es decir, de las 9:00 a.m. a las 6:00 p.m.
  - b) Intente suprimir los empleados con JOB\_ID SA REP que no estén asignados a un departamento.

**Indicación:** empleado Grant con identificador 178.

## Soluciones a la Práctica 8-1: Creación de Disparadores de Sentencia y de Fila

En esta práctica, creará disparadores de sentencia y de fila. También creará procedimientos que se llamarán desde los disparadores.

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

**Abra el script /home/oracle/labs/plpu/solns/sol\_08\_01\_a.sql.**

**Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```

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

```

## Soluciones a la Práctica 8-1: Creación de Disparadores de Sentencia y de Fila (continuación)

```
PROCEDURE check_salary Compiled.
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 debe transferir el nuevo identificador de trabajo y salario a los parámetros de procedimiento.

**Abra el script /home/oracle/labs/plpu/solns/sol\_08\_01\_b.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```
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
```

```
TRIGGER check_salary_trg Compiled.
No Errors.
```

- 2) Pruebe el disparador CHECK\_SAL\_TRG utilizando los siguientes casos:
- Utilice el procedimiento EMP\_PKG.ADD\_EMPLOYEE para agregar a la empleada Eleanor Beh al departamento 30. ¿Qué sucede? ¿Por qué?
- Abra el script /home/oracle/labs/plpu/solns/sol\_08\_02\_a.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

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

## Soluciones a la Práctica 8-1: Creación de Disparadores de Sentencia y de Fila (continuación)

```

Results Script Output Explain Autotrace DBMS Output OWA Output
| Edit | Save | Print |
Error starting at line 1 in command:
EXECUTE emp_pkg.add_employee('Eleanor', 'Beh', 30)
Error report:
ORA-20100: Invalid salary $1000. Salaries for job SA_REP must be between $6000 and $12000
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'
ORA-06512: at "ORA61.EMP_PKG", line 35
ORA-06512: at "ORA61.EMP_PKG", line 51
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?
- Abra el script `/home/oracle/labs/plpu/solns/sol_08_02_b.sql`. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.

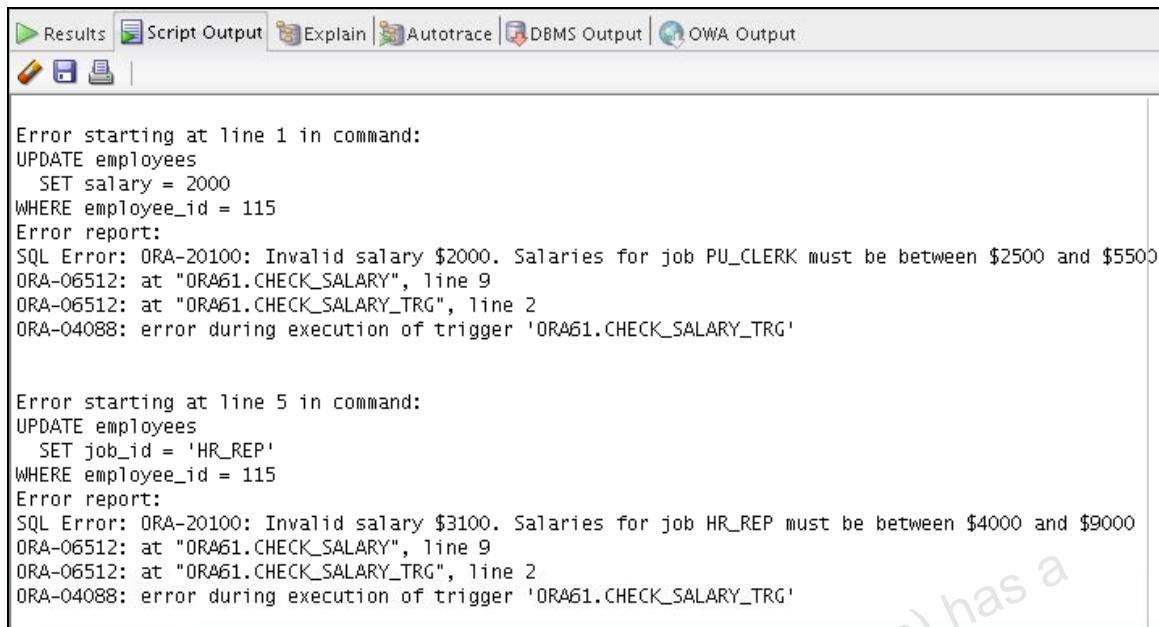
```

UPDATE employees
 SET salary = 2000
 WHERE employee_id = 115;

UPDATE employees
 SET job_id = 'HR REP'
 WHERE employee_id = 115;

```

## Soluciones a la Práctica 8-1: Creación de Disparadores de Sentencia y de Fila (continuación)



The screenshot shows the Oracle SQL Worksheet interface with the 'Results' tab selected. Two error messages are displayed:

```
Error starting at line 1 in command:
UPDATE employees
 SET salary = 2000
WHERE employee_id = 115
Error report:
SQL Error: ORA-20100: Invalid salary $2000. Salaries for job PU_CLERK must be between $2500 and $5500
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'

Error starting at line 5 in command:
UPDATE employees
 SET job_id = 'HR REP'
WHERE employee_id = 115
Error report:
SQL Error: ORA-20100: Invalid salary $3100. Salaries for job HR REP must be between $4000 and $9000
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'
```

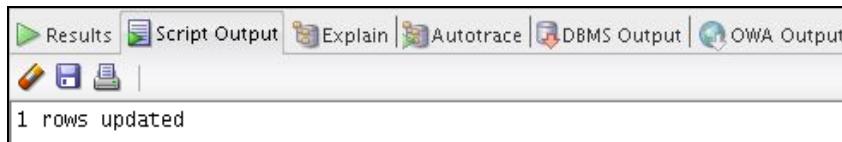
**La primera sentencia de actualización no consigue definir el salario en 2.000 dólares. La regla del disparador de comprobación del salario no consigue realizar la operación de actualización, porque el nuevo salario para el empleado 115 es menor que el mínimo que permite el identificador de trabajo 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 identificador de trabajo HR REP.**

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

Abra el script /home/oracle/labs/plpu/solns/sol\_08\_02\_c.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.

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



The screenshot shows the Oracle SQL Worksheet interface with the 'Results' tab selected. The output shows:

```
1 rows updated
```

## Soluciones a la Práctica 8-1: Creación de Disparadores de Sentencia y de Fila (continuación)

**La operación de actualización se ha realizado correctamente, porque el nuevo salario está dentro del rango aceptable 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) Implante 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.

**Abra el script /home/oracle/labs/plpu/solns/sol\_08\_03\_a.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```
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;
/
SHOW ERRORS
```



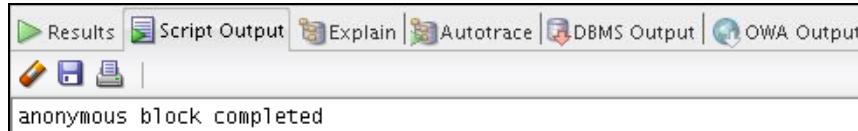
- b) Compruebe el disparador ejecutando el procedimiento EMP\_PKG.ADD\_EMPLOYEE con los siguientes valores de parámetros:
  - p\_first\_name: 'Eleanor'
  - p\_last\_name: 'Beh'
  - p\_Email: 'EBEH'
  - p\_Job: 'IT\_PROG'
  - p\_Sal: 5000

## Soluciones a la Práctica 8-1: Creación de Disparadores de Sentencia y de Fila (continuación)

Abra el script `/home/oracle/labs/plpu/solns/sol_08_03_b.sql`.

Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.

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



- c) Actualice a los empleados con un trabajo `IT_PROG` incrementando su salario en 2.000 dólares. ¿Qué sucede?

Abra el script `/home/oracle/labs/plpu/solns/sol_08_03_c.sql`.

Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.

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

```
Error starting at line 1 in command:
UPDATE employees
SET salary = salary + 2000
WHERE job_id = 'IT_PROG'
Error report:
SQL Error: ORA-20100: Invalid salary $11000. Salaries for job IT_PROG must be between $4000 and $10000
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.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`.**

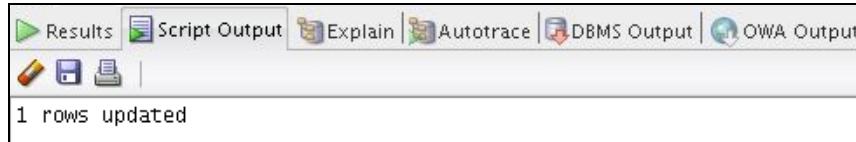
- d) Actualice a 9.000 dólares el salario de Eleanor Beh.

Abra el script `/home/oracle/labs/plpu/solns/sol_08_03_d.sql`.

Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.

## Soluciones a la Práctica 8-1: Creación de Disparadores de Sentencia y de Fila (continuación)

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

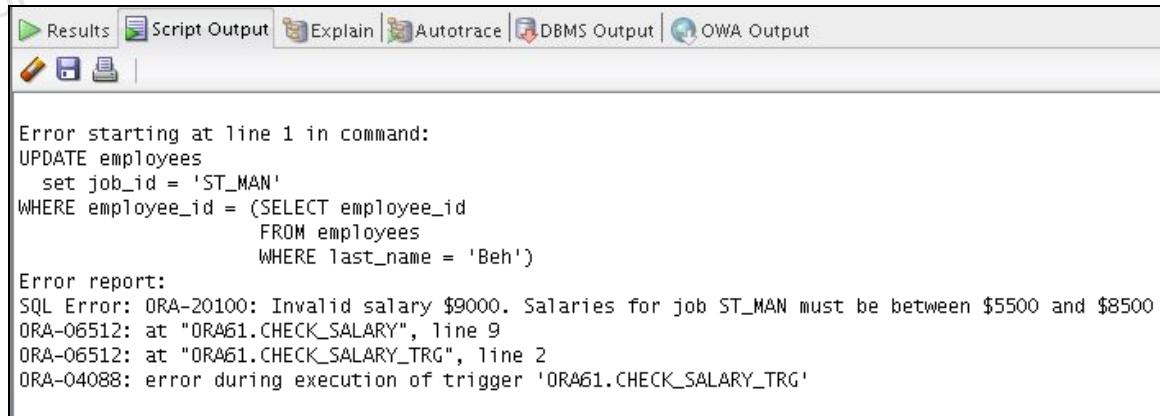


**Indicación:** utilice una sentencia UPDATE con una subconsulta en la cláusula WHERE. ¿Qué sucede?

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

Abra el script /home/oracle/labs/plpu/solns/sol\_08\_03\_e.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.

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



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

## Soluciones a la Práctica 8-1: Creación de Disparadores de Sentencia y de Fila (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.

**Abra el script /home/oracle/labs/plpu/solns/sol\_08\_04\_a.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```
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
business
hours of 9AM and 6PM');
 END IF;
END;
/
SHOW ERRORS
```

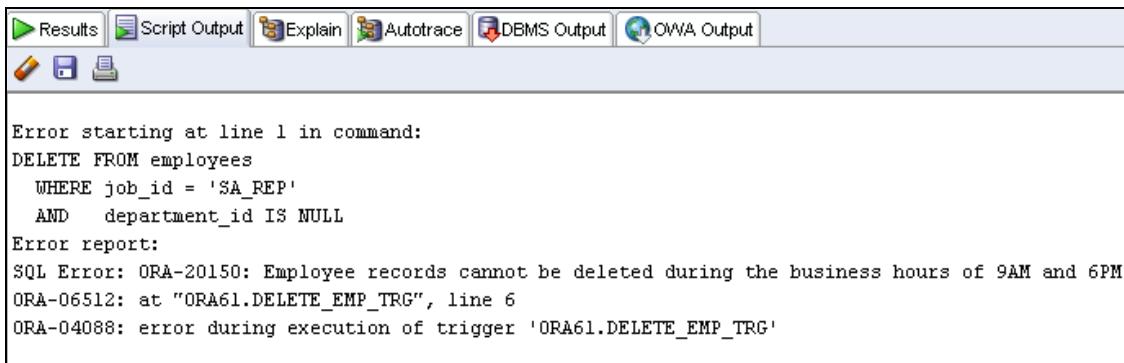
- b) Intente suprimir los empleados con JOB\_ID SA REP que no estén asignados a un departamento.

**Indicación:** empleado Grant con identificador 178.

**Abra el script /home/oracle/labs/plpu/solns/sol\_08\_04\_b.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

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

## Soluciones a la Práctica 8-1: Creación de Disparadores de Sentencia y de Fila (continuación)



The screenshot shows the Oracle SQL Developer interface with the 'Results' tab selected. The results window displays the following error message:

```
Error starting at line 1 in command:
DELETE FROM employees
 WHERE job_id = 'SA_REP'
 AND department_id IS NULL
Error report:
SQL Error: ORA-20150: Employee records cannot be deleted during the business hours of 9AM and 6PM
ORA-06512: at "ORA61.DELETE_EMP_TRG", line 6
ORA-04088: error during execution of trigger 'ORA61.DELETE_EMP_TRG'
```

Nota: en función de la hora actual en la máquina host del aula, es posible que pueda o que no pueda realizar operaciones de supresión. Por ejemplo, en la captura de pantalla anterior, falla la operación de supresión, ya que se realiza fuera de las horas laborables permitidas (según la hora de la máquina host).

## Prácticas y Soluciones de la Lección 9

En esta práctica, implantará una regla de negocio sencilla para garantizar la integridad de los datos de los salarios de los empleados con respecto al rango de salarios válidos para sus trabajos. Cree un disparador para esta regla. Durante este proceso, los nuevos disparadores darán lugar a un efecto en cascada con disparadores creados en la sección práctica de la lección anterior. El efecto en cascada originará una excepción de tabla mutante en la tabla JOBS. A continuación, cree un paquete PL/SQL y disparadores adicionales para resolver el problema de la tabla mutante.

**Nota:** si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

## Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes

En esta práctica, implantará una regla de negocio sencilla para garantizar la integridad de los datos de los salarios de los empleados con respecto al rango de salarios válidos para sus trabajos. Cree un disparador para esta regla. Durante este proceso, los nuevos disparadores darán lugar a un efecto en cascada con disparadores creados en la sección práctica de la lección anterior. El efecto en cascada originará una excepción de tabla mutante en la tabla JOBS. A continuación, cree un paquete PL/SQL y disparadores adicionales para resolver el problema de la tabla mutante.

- 1) Los empleados reciben un aumento de sueldo automáticamente si el salario mínimo de un trabajo se aumenta a un valor superior que sus salarios actuales. Implante 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 los salarios 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 (que actualizó por última vez en la práctica 8), de la siguiente forma:
    - i. Agregue un procedimiento denominado SET\_SALARY, que actualice los salarios de los empleados.
    - ii. El procedimiento SET\_SALARY acepta los dos parámetros siguientes: el identificador de trabajo de aquellos salarios que puede que haya que actualizar y el nuevo salario mínimo para el identificador del trabajo
  - b. Cree un disparador de fila denominado UPD\_MINSALARY\_TRG en la tabla JOBS, que llame al procedimiento EMP\_PKG.SET\_SALARY cuando el salario mínimo de la tabla JOBS se actualice para un identificador de trabajo especificado.
  - c. Escriba una consulta para mostrar el identificador de empleado, el apellido, el identificador de trabajo, el salario actual y el salario mínimo para los empleados que sean programadores, es decir, su JOB\_ID es 'IT\_PROG'. A continuación, actualice el salario mínimo en la tabla JOBS para aumentarlo en 1.000 dólares. ¿Qué sucede?
- 2) Para resolver el problema de la tabla mutante, cree JOBS\_PKG para mantener en memoria una copia de las filas de la tabla JOBS. A continuación, modifique 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, 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 arranque el disparador de fila CHECK\_SALARY.
  - a. Cree un nuevo paquete denominado JOBS\_PKG con la siguiente especificación:

## Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)

```

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

```

- b. Implante el cuerpo de JOBS\_PKG, como se detalla a continuación:
- Declare una tabla de índice PL/SQL privada denominada jobs\_tab\_type indexada por un tipo de cadena basada en JOBS.JOB\_ID%TYPE.
  - Declare una variable privada denominada jobstab basada en jobs\_tab\_type.
  - 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 p\_jobid como índice para jobstab y devuelve min\_salary para dicho elemento.
  - La función GET\_MAXSALARY utiliza un parámetro p\_jobid como índice para jobstab y devuelve max\_salary para dicho elemento.
  - El procedimiento SET\_MINSALARY utiliza su p\_jobid como índice para jobstab con el fin de definir el campo min\_salary de su elemento en el valor del parámetro min\_salary.
  - El procedimiento SET\_MAXSALARY utiliza su p\_jobid como índice para jobstab con el fin de definir el campo max\_salary de su elemento en el valor del parámetro max\_salary.
- 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 adecuadas. Este paso debe eliminar la excepción de disparador mutante.
- d. Implante un disparador de sentencia BEFORE INSERT OR UPDATE denominado INIT\_JOBPKG\_TRG que utilice la sintaxis CALL para llamar al procedimiento JOBS\_PKG.INITIALIZE, con el fin de garantizar que el estado del paquete sea actual antes de que se realicen las operaciones DML.
- e. Pruebe los cambios de código ejecutando la consulta para mostrar los empleados que son programadores y, a continuación, emita una sentencia de actualización para aumentar el salario mínimo del tipo de trabajo IT\_PROG en 1.000 en la tabla JOBS. Después de esto, realice una consulta de los empleados con el tipo de trabajo IT\_PROG para comprobar los cambios resultantes. ¿Los salarios de qué empleados se han definido en el mínimo para sus trabajos?

## **Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)**

- 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.
- Pruébelo agregando un nuevo empleado mediante EMP\_PKG.ADD\_EMPLOYEE con los siguientes parámetros: ('Steve', 'Morse', 'SMORSE', and sal => 6500). ¿Qué sucede?
  - 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.
  - Pruebe el disparador agregando el empleado Steve Morse de nuevo. Confirme el registro insertado en la tabla EMPLOYEES mostrando el identificador de empleado, el nombre y el apellido, el salario, el identificador de trabajo y el identificador de departamento.

## Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes

En esta práctica, implantará una regla de negocio sencilla para garantizar la integridad de los datos de los salarios de los empleados con respecto al rango de salarios válidos para sus trabajos. Cree un disparador para esta regla. Durante este proceso, los nuevos disparadores darán lugar a un efecto en cascada con disparadores creados en la sección práctica de la lección anterior. El efecto en cascada originará una excepción de tabla mutante en la tabla JOBS. A continuación, cree un paquete PL/SQL y disparadores adicionales para resolver el problema de la tabla mutante.

- 1) Los empleados reciben un aumento de sueldo automáticamente si el salario mínimo de un trabajo se aumenta a un valor superior a sus salarios actuales. Implante 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 los salarios 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 (que actualizó por última vez en la práctica 8), de la siguiente forma:
    - i. Agregue un procedimiento denominado SET\_SALARY, que actualice los salarios de los empleados.
    - ii. El procedimiento SET\_SALARY acepta los dos parámetros siguientes: el identificador de trabajo de aquellos salarios que puede que haya que actualizar y el nuevo salario mínimo para el identificador del trabajo.

**Abra el script /home/oracle/labs/plpu/solns/sol\_09\_01\_a.sql.**

**Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran de la siguiente forma. El código que se acaba de agregar se resalta en negrita en el cuadro de código siguiente.**

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS

 TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_email employees.email%TYPE,
 p_job employees.job_id%TYPE DEFAULT 'SA_REP',
 p_mgr employees.manager_id%TYPE DEFAULT 145,
 p_sal employees.salary%TYPE DEFAULT 1000,
 p_comm employees.commission_pct%TYPE DEFAULT 0,
```

## Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)

```

p_deptid employees.department_id%TYPE DEFAULT 30);

PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,
 p_last_name employees.last_name%TYPE,
 p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
 p.empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,
 p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p_emp_id employees.employee_id%type)
 return employees%rowtype;

FUNCTION get_employee(p_family_name
employees.last_name%type)
 return employees%rowtype;

PROCEDURE get_employees(p_dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

PROCEDURE show_employees;

/* New set_salary procedure */

PROCEDURE set_salary(p_jobid VARCHAR2, p_min_salary NUMBER);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

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

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

 PROCEDURE add_employee(
 p_first_name employees.first_name%TYPE,

```

## Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)

```

p_last_name employees.last_name%TYPE,
p_email employees.email%TYPE,
p_job employees.job_id%TYPE DEFAULT 'SA_REP',
p_mgr employees.manager_id%TYPE DEFAULT 145,
p_sal employees.salary%TYPE DEFAULT 1000,
p_comm employees.commission_pct%TYPE DEFAULT 0,
p_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,p_first_name||' '||p_last_name);
 COMMIT;
END audit_newemp;

BEGIN -- add_employee
 IF valid_deptid(p_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, p_first_name,
p_last_name, p_email,
 p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
 ELSE
 RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
 END IF;
 END add_employee;

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

PROCEDURE get_employee(
 p.empid IN employees.employee_id%TYPE,
 p_sal OUT employees.salary%TYPE,

```

## Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)

```

 p_job OUT employees.job_id%TYPE) IS
BEGIN
 SELECT salary, job_id
 INTO p_sal, p_job
 FROM employees
 WHERE employee_id = p_empid;
END get_employee;

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

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

PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
BEGIN
 SELECT * BULK COLLECT INTO emp_table
 FROM EMPLOYEES
 WHERE department_id = p_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(p_rec_emp employees%rowtype) IS
BEGIN
 DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
 p_rec_emp.employee_id|| ' ' ||
 p_rec_emp.first_name|| ' ' ||
 p_rec_emp.last_name|| ' ')

```

## Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)

```

 p_rec_emp.job_id || ' ' ||
 p_rec_emp.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(p_deptid IN
departments.department_id%TYPE)
RETURN BOOLEAN IS
 v_dummy PLS_INTEGER;
BEGIN
 RETURN valid_departments.exists(p_deptid);
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RETURN FALSE;
END valid_deptid;

/* New set_salary procedure */

PROCEDURE set_salary(p_jobid VARCHAR2, p_min_salary NUMBER) IS
CURSOR cur_emp IS
 SELECT employee_id
 FROM employees
 WHERE job_id = p_jobid AND salary < p_min_salary;
BEGIN
 FOR rec_emp IN cur_emp
 LOOP
 UPDATE employees
 SET salary = p_min_salary
 WHERE employee_id = rec_emp.employee_id;
 END LOOP;
END set_salary;

BEGIN
 init_departments;
END emp_pkg;

/
SHOW ERRORS

```

## Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)

Results    Script Output    Explain    Autotrace    DBMS Output    OWA Output

PACKAGE emp\_pkg Compiled.  
No Errors.  
PACKAGE BODY emp\_pkg Compiled.  
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.

**Abra el script /home/oracle/labs/plpu/solns/sol\_09\_01\_b.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```
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
```

Results    Script Output    Explain    Autotrace    DBMS Output    OWA Output

TRIGGER upd\_minsalary\_trg Compiled.  
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?

**Abra el script /home/oracle/labs/plpu/solns/sol\_09\_01\_c.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

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

## Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)

```
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   |
| 214         | Beh       | 9000   |

6 rows selected

Error starting at line 5 in command:  
 UPDATE jobs  
 SET min\_salary = min\_salary + 1000  
 WHERE job\_id = 'IT\_PROG'  
 Error report:  
 SQL Error: ORA-04091: table ORA61.JOBS is mutating, trigger/function may not see it  
 ORA-06512: at "ORA61.CHECK\_SALARY", line 5  
 ORA-06512: at "ORA61.CHECK\_SALARY\_TRG", line 2  
 ORA-04088: error during execution of trigger 'ORA61.CHECK\_SALARY\_TRG'  
 ORA-06512: at "ORA61.EMP\_PKG", line 143  
 ORA-06512: at "ORA61.UPD\_MINSALARY\_TRG", line 2  
 ORA-04088: error during execution of trigger 'ORA61.UPD\_MINSALARY\_TRG'  
 04091. 00000 - "table %s.%s is mutating, trigger/function may not see it"  
 \*Cause: A trigger (or a user defined plsql function that is referenced in  
 this statement) attempted to look at (or modify) a table that was  
 in the middle of being modified by the statement which fired it.  
 \*Action: Rewrite the trigger (or function) so it does not read that table.

**La actualización de la columna `min_salary` del 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`. Mientras se lee la tabla `JOBS`, el procedimiento `CHECK_SALARY` detecta la excepción de tabla mutante.**

- 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, modifique 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, 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 arranque el disparador de fila `CHECK_SALARY`.

## Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)

- 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);

```

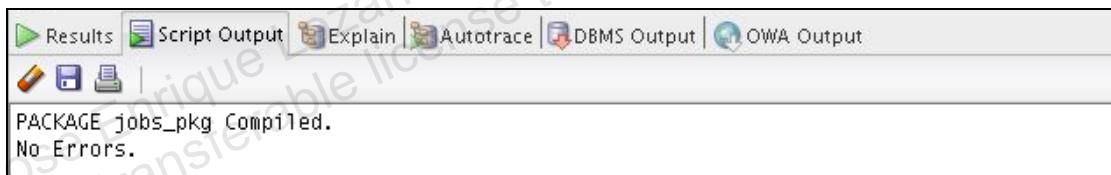
Abra el archivo **sol\_09\_02\_a.sql** de la carpeta

**/home/oracle/labs/plpu/solns** o copie y pegue el código siguiente en el área SQL Worksheet. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.

```

CREATE OR REPLACE PACKAGE jobs_pkg IS
 PROCEDURE initialize;
 FUNCTION get_minsalary(p_jobid VARCHAR2) RETURN NUMBER;
 FUNCTION get_maxsalary(p_jobid VARCHAR2) RETURN NUMBER;
 PROCEDURE set_minsalary(p_jobid VARCHAR2, p_min_salary
NUMBER);
 PROCEDURE set_maxsalary(p_jobid VARCHAR2, p_max_salary
NUMBER);
END jobs_pkg;
/
SHOW ERRORS

```



- b) Implante el cuerpo de JOBS\_PKG, como se detalla a continuación:

- i. Declare una tabla de índice PL/SQL privada denominada `jobs_tab_type` indexada por un tipo de cadena basada en `JOBS.JOB_ID%TYPE`.
- ii. Declare una variable privada denominada `jobstab` basada en `jobs_tab_type`.
- iii. 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.
- iv. La función `GET_MINSalary` utiliza un parámetro `p_jobid` como índice para `jobstab` y devuelve `min_salary` para dicho elemento.
- v. La función `GET_MAXSalary` utiliza un parámetro `p_jobid` como índice para `jobstab` y devuelve `max_salary` para dicho elemento.

## **Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)**

- vi. El procedimiento SET\_MINSalary utiliza su p\_jobid como índice para jobstab con el fin de definir el campo min\_salary de su elemento en el valor del parámetro min\_salary.
- vii. El procedimiento SET\_MAXSalary utiliza su p\_jobid como índice para jobstab con el fin de definir el campo max\_salary de su elemento en el valor del parámetro max\_salary.

**Abra el script /home/oracle/labs/plpu/solns/sol\_09\_02\_b.sql.**

**Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación. Para compilar el cuerpo del paquete, haga clic con el botón derecho en el nombre o el cuerpo del paquete en el árbol Object Navigator y, a continuación, seleccione Compile.**

```

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

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

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

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

 PROCEDURE set_minsalary(p_jobid VARCHAR2, p_min_salary
NUMBER) IS
 BEGIN
 jobstab(p_jobid).max_salary := p_min_salary;
 END set_minsalary;

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

```

## Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)

```
END jobs_pkg;
/
SHOW ERRORS
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

PACKAGE BODY jobs\_pkg Compiled.  
No Errors.

- c) Copie el procedimiento CHECK\_SALARY de la práctica 8, 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 adecuadas. Este paso debe eliminar la excepción de disparador mutante.

**Abra el script /home/oracle/labs/plpu/solns/sol\_09\_02\_c.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```
CREATE OR REPLACE PROCEDURE check_salary (p_the_job
VARCHAR2, p_the_salary NUMBER) IS
 v_minsal jobs.min_salary%type;
 v_maxsal jobs.max_salary%type;
BEGIN
 /*
 ** Commented out to avoid mutating trigger exception on
 the JOBS table
 SELECT min_salary, max_salary INTO v_minsal, v_maxsal
 FROM jobs
 WHERE job_id = UPPER(p_the_job);
 */

 v_minsal := jobs_pkg.get_minsalary(UPPER(p_the_job));
 v_maxsal := jobs_pkg.get_maxsalary(UPPER(p_the_job));
 IF p_the_salary NOT BETWEEN v_minsal AND v_maxsal THEN
 RAISE_APPLICATION_ERROR(-20100,
 'Invalid salary-'.$' || p_the_salary || '. ' ||
 'Salaries for job '|| p_the_job ||
 ' must be between $'|| v_minsal || ' and $' || v_maxsal);
 END IF;
 END;
/
SHOW ERRORS
```

## Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)

```
PROCEDURE check_salary Compiled.
No Errors.
```

- d) Implante 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.

**Abra el script /home/oracle/labs/plpu/solns/sol\_09\_02\_d.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

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

```
TRIGGER init_jobpkg_trg Compiled.
No Errors.
```

- e) Pruebe los cambios de código ejecutando la consulta para mostrar los empleados que son programadores y, a continuación, emita una sentencia de actualización para aumentar el salario mínimo del tipo de trabajo IT\_PROG en 1.000 en la tabla JOBS. Después de esto, realice una consulta de los empleados con el tipo de trabajo IT\_PROG para comprobar los cambios resultantes. ¿Los salarios de qué empleados se han definido en el mínimo para sus trabajos?

**Abra el script /home/oracle/labs/plpu/solns/sol\_09\_02\_e.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```
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';
```

## Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)

The screenshot shows the following SQL session:

```

Results Script Output Explain Autotrace DBMS Output OWA Output
| | | | | |
EMPLOYEE_ID LAST_NAME SALARY
-----|-----|-----
103 Hunold 9000
104 Ernst 6000
105 Austin 4800
106 Pataballa 4800
107 Lorentz 4200
214 Beh 9000
6 rows selected

1 rows updated
EMPLOYEE_ID LAST_NAME SALARY
-----|-----|-----
103 Hunold 9000
104 Ernst 6000
105 Austin 5000
106 Pataballa 5000
107 Lorentz 5000
214 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 implantado 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) Pruébelo agregando un nuevo empleado mediante EMP\_PKG.ADD\_EMPLOYEE con los siguientes parámetros: ('Steve', 'Morse', 'SMORSE', and sal => 6500). ¿Qué sucede?

**Abra el script /home/oracle/labs/plpu/solns/sol\_09\_03\_a.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

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

## Soluciones a la Práctica 9-1: Gestión de Reglas de Integridad de Datos y de Excepciones de Tablas Mutantes (continuación)

```
Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
| |
anonymous block completed
```

- 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.

**Abra el script /home/oracle/labs/plpu/solns/sol\_09\_03\_b.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

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

```
Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
| |
TRIGGER employee_initjobs_trg Compiled.
```

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

**Abra el script /home/oracle/labs/plpu/solns/sol\_09\_03\_c.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```
Error starting at line 1 in command:
EXECUTE emp_pkg.add_employee('Steve', 'Morse', 'SMORSE', p_sal => 6500)
Error report:
ORA-00001: unique constraint (ORA61.EMP_EMAIL_UK) violated
ORA-06512: at "ORA61.EMP_PKG", line 35
ORA-06512: at line 1
00001. 00000 - "unique constraint (%s,%s) violated"
*Cause: An UPDATE or INSERT statement attempted to insert a duplicate key.
For Trusted Oracle configured in DBMS MAC mode, you may see
this message if a duplicate entry exists at a different level.
*Action: Either remove the unique restriction or do not insert the key.
```

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | SALARY | JOB_ID | DEPARTMENT_ID |
|-------------|------------|-----------|--------|--------|---------------|
| 215         | Steve      | Morse     | 6500   | SA_REP | 30            |

1 rows selected

## Prácticas y Soluciones de la Lección 10

En esta práctica, visualizará los parámetros de inicialización del compilador. A continuación, activará la compilación nativa para la sesión y compilará un procedimiento. Después, suprimirá todas las categorías de advertencia del compilador para, a continuación, restaurar la configuración de advertencia de sesión original. Por último, identificará las categorías de algunos números de mensaje de advertencia del compilador.

**Nota:** si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

## Práctica 10-1: Uso de Parámetros y Advertencias del Compilador PL/SQL

En esta práctica, visualizará los parámetros de inicialización del compilador. A continuación, activará la compilación nativa para la sesión y compilará un procedimiento. Después, suprimirá todas las categorías de advertencia del compilador para, a continuación, restaurar la configuración de advertencia de sesión original. Por último, identificará las categorías de algunos números de mensaje de advertencia del compilador.

- 1) Cree y ejecute un script `lab_10_01` para mostrar la siguiente información sobre los parámetros de inicialización del compilador mediante la vista del diccionario de datos `USER_PLSQL_OBJECT_SETTINGS`. Observe la configuración del objeto `ADD_JOB_HISTORY`.
 

**Nota:** utilice el icono Execute Statement (F9) para mostrar los resultados en el separador Results.

  - a) Nombre de objeto
  - b) Tipo de objeto
  - c) Modo de compilación del objeto
  - d) Nivel de optimización de la compilación
- 2) Modifique el parámetro `PLSQL_CODE_TYPE` para activar la compilación nativa de la sesión y compile `ADD_JOB_HISTORY`.
  - a) Ejecute el comando `ALTER SESSION` para activar la compilación nativa de la sesión.
  - b) Compile el procedimiento `ADD_JOB_HISTORY`.
  - c) Vuelva a ejecutar el script `sol_10_01`. Observe el parámetro `PLSQL_CODE_TYPE`.
  - d) Cambie la compilación para utilizar el modo de compilación interpretada, de la siguiente forma:
- 3) Utilice la región Tools > Preferences > PL/SQL Compiler Options para desactivar todas las categorías de advertencias del compilador.
- 4) Edite, examine y ejecute el script `lab_10_04.sql` para crear el procedimiento `UNREACHABLE_CODE`. Haga clic en el icono Run Script (F5) para crear el procedimiento. Utilice el nombre del procedimiento en el árbol Navigation para compilar el procedimiento.
- 5) ¿Qué advertencias de compilador aparecen en el separador Compiler – Log, en caso de que aparezca alguna?
- 6) Active todos los mensajes de advertencia del compilador para esta sesión mediante la ventana Preferences.
- 7) Recompile el procedimiento `UNREACHABLE_CODE` mediante el árbol Object Navigation. ¿Qué advertencias del compilador (si hay alguna) se muestran?
- 8) Utilice la vista del diccionario de datos `USER_ERRORS` para mostrar los detalles de los mensajes de advertencia del compilador, de la siguiente forma.
- 9) Cree un script denominado `warning_msgs` que utilice los paquetes `EXECUTE_DBMS_OUTPUT` y `DBMS_WARNING` para identificar las categorías para los siguientes números de mensajes de advertencia del compilador: 5050, 6075 y 7100. Active `SERVERROUTPUT` antes de ejecutar el script.

## Soluciones a la Práctica 10-1: Uso de Parámetros y Advertencias del Compilador PL/SQL

En esta práctica, visualizará los parámetros de inicialización del compilador. A continuación, activará la compilación nativa para la sesión y compilará un procedimiento. Después, suprimirá todas las categorías de advertencia del compilador para, a continuación, restaurar la configuración de advertencia de sesión original. Por último, identificará las categorías de algunos números de mensaje de advertencia del compilador.

- 1) Cree y ejecute un script lab\_10\_01 para mostrar la siguiente información sobre los parámetros de inicialización del compilador mediante la vista del diccionario de datos USER\_PLSQL\_OBJECT\_SETTINGS. Observe la configuración del objeto ADD\_JOB\_HISTORY.

**Nota:** utilice el icono Execute Statement (F9) para mostrar los resultados en el separador Results.

- a) Nombre de objeto
- b) Tipo de objeto
- c) Modo de compilación del objeto
- d) Nivel de optimización de la compilación

Abra el script /home/oracle/labs/plpu/solns/sol\_10\_01.sql. Haga clic en el icono Execute Statement (F9) de la barra de herramientas de SQL Worksheet para ejecutar la consulta. El código y un ejemplo del resultado se muestran a continuación.

```
SELECT name, type, plsql_code_type as code_type,
 plsql_optimize_level as opt_lvl
 FROM user_plsql_object_settings;
```

| NAME                   | TYPE             | CODE_TYPE          |
|------------------------|------------------|--------------------|
| ADD_EMPLOYEE           | PROCEDURE        | INTERPRETED        |
| ADD_JOB                | PROCEDURE        | INTERPRETED        |
| <b>ADD_JOB_HISTORY</b> | <b>PROCEDURE</b> | <b>INTERPRETED</b> |
| CHECK_SALARY           | PROCEDURE        | INTERPRETED        |
| CHECK_SALARY_TRG       | TRIGGER          | INTERPRETED        |
| COMPILE_PKG            | PACKAGE          | INTERPRETED        |
| COMPILE_PKG            | PACKAGE BODY     | INTERPRETED        |
| DELETE_EMP_TRG         | TRIGGER          | INTERPRETED        |
| DEL_JOB                | PROCEDURE        | INTERPRETED        |
| EMPLOYEE_INITJOBS_TRG  | TRIGGER          | INTERPRETED        |
| EMPLOYEE_REPORT        | PROCEDURE        | INTERPRETED        |
| EMP_LIST               | PROCEDURE        | INTERPRETED        |
| EMP_PKG                | PACKAGE          | INTERPRETED        |
| EMP_PKG                | PACKAGE BODY     | INTERPRETED        |
| GET_ANNUAL_COMP        | FUNCTION         | INTERPRETED        |
| GET_EMPLOYEE           | PROCEDURE        | INTERPRETED        |
| GET_JOB                | FUNCTION         | INTERPRETED        |
| GET_LOCATION           | FUNCTION         | INTERPRETED        |
| INIT_JOBPKG_TRG        | TRIGGER          | INTERPRETED        |
| JOB_PKG                | PACKAGE          | INTERPRETED        |
| JOB_PKG                | PACKAGE BODY     | INTERPRETED        |
| JOB_PKG                | PACKAGE          | INTERPRETED        |

...

## Soluciones a la Práctica 10-1: Uso de Parámetros y Advertencias del Compilador PL/SQL (continuación)

- 2) Modifique el parámetro PLSQL\_CODE\_TYPE para activar la compilación nativa de la sesión y compile ADD\_JOB\_HISTORY.

- a) Ejecute el comando ALTER SESSION para activar la compilación nativa de la sesión.

**Abra el script /home/oracle/labs/plpu/solns/sol\_10\_02\_a.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar la consulta. El código y el resultado se muestran a continuación.**

```
ALTER SESSION SET PLSQL_CODE_TYPE = 'NATIVE';
```

ALTER SESSION SET succeeded.

- b) Compile el procedimiento ADD\_JOB\_HISTORY.

**Abra el script /home/oracle/labs/plpu/solns/sol\_10\_02\_b.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar la consulta. El código y el resultado se muestran a continuación.**

```
ALTER PROCEDURE add_job_history COMPILE;
```

ALTER PROCEDURE add\_job\_history succeeded.

- c) Vuelva a ejecutar el script sol\_10\_01.sql. Observe el parámetro PLSQL\_CODE\_TYPE.

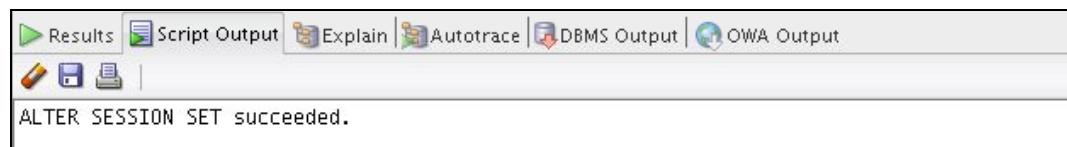
```
SELECT name, type, plsql_code_type as code_type,
plsql_optimize_level as opt_lvl
FROM user_plsql_object_settings;
```

| NAME             | TYPE      | CODE_TYPE   |
|------------------|-----------|-------------|
| ADD_EMPLOYEE     | PROCEDURE | INTERPRETED |
| ADD_JOB          | PROCEDURE | INTERPRETED |
| ADD_JOB_HISTORY  | PROCEDURE | NATIVE      |
| CHECK_SALARY     | PROCEDURE | INTERPRETED |
| CHECK_SALARY TRG | TRIGGER   | INTERPRETED |

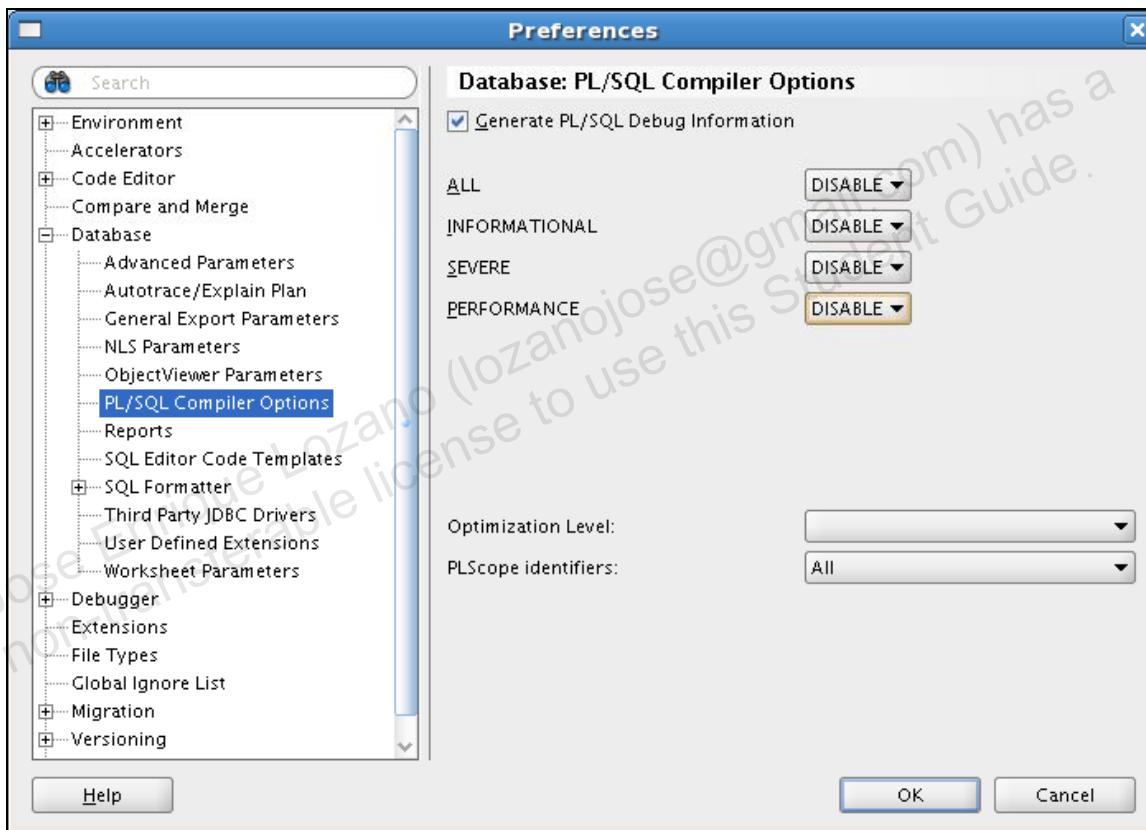
## Soluciones a la Práctica 10-1: Uso de Parámetros y Advertencias del Compilador PL/SQL (continuación)

- d) Cambie la compilación para utilizar el modo de compilación interpretada, de la siguiente forma:

```
ALTER SESSION SET PLSQL_CODE_TYPE = 'INTERPRETED';
```



- 3) Utilice la región Tools > Preferences > PL/SQL Compiler Options para desactivar todas las categorías de advertencias del compilador.



**Seleccione DISABLE para las cuatro categorías de advertencias del compilador PL/SQL y, a continuación, haga clic en OK.**

- 4) Edite, examine y ejecute el script lab\_10\_04.sql para crear el procedimiento UNREACHABLE\_CODE. Haga clic en el ícono Run Script (F5) para crear y compilar el procedimiento.

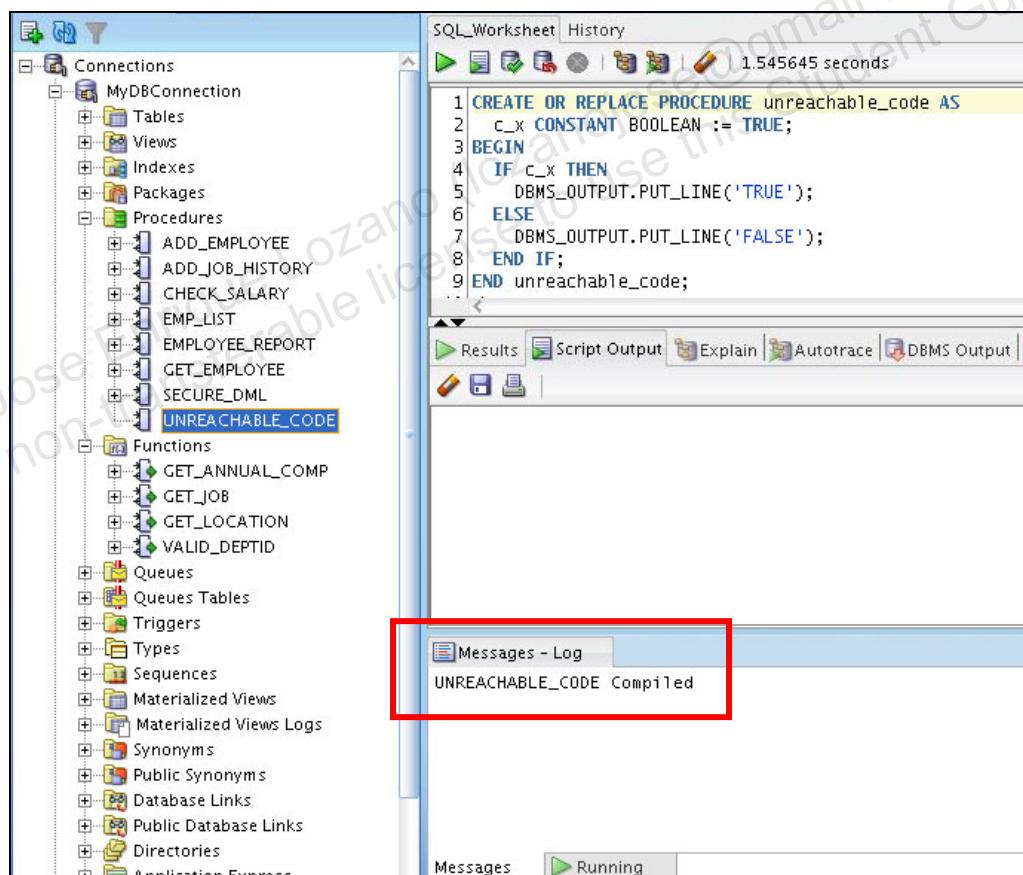
**Abra el script /home/oracle/labs/plpu/solns/sol\_10\_04.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar la consulta. El código y el resultado se muestran a continuación.**

## Soluciones a la Práctica 10-1: Uso de Parámetros y Advertencias del Compilador PL/SQL (continuación)

```
CREATE OR REPLACE PROCEDURE unreachable_code AS
 c_x CONSTANT BOOLEAN := TRUE;
BEGIN
 IF c_x THEN
 DBMS_OUTPUT.PUT_LINE('TRUE');
 ELSE
 DBMS_OUTPUT.PUT_LINE('FALSE');
 END IF;
END unreachable_code;
/
```



Para ver los posibles errores del compilador, haga clic con el botón derecho en el nombre del procedimiento en el nodo Procedures del árbol Navigation y, a continuación, haga clic en Compile.



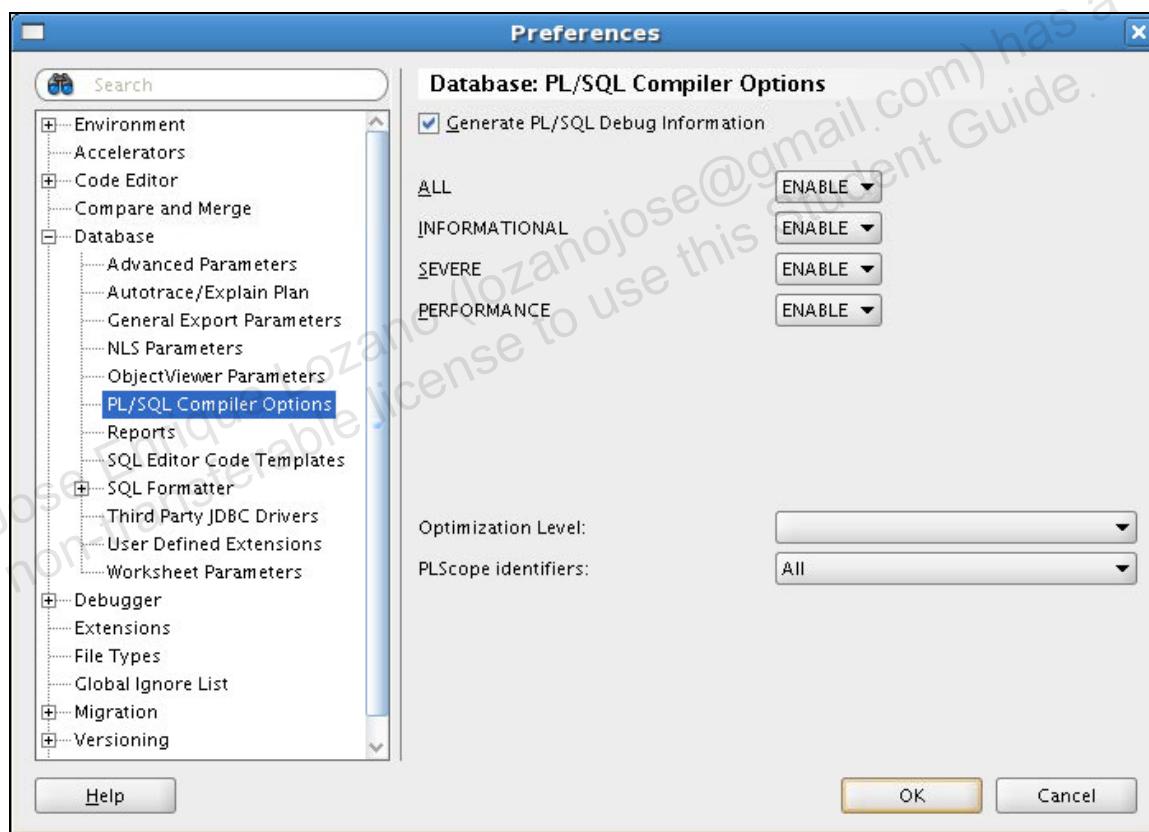
### Nota

- Si el procedimiento no aparece en el árbol Navigation, haga clic en el icono Refresh del separador Connections.

## Soluciones a la Práctica 10-1: Uso de Parámetros y Advertencias del Compilador PL/SQL (continuación)

- Asegúrese de que aparece el separador Messages – Log (seleccione View > Log en la barra de menús).
- 5) ¿Qué advertencias de compilador aparecen en el separador Compiler – Log, en caso de que aparezca alguna?
- Observe que el mensaje en el separador Messages – Log es “UNREACHABLE\_CODE Compiled” sin ningún mensaje de advertencia, porque ha desactivado las advertencias del compilador en el paso 3.**
- 6) Active todos los mensajes de advertencia del compilador para esta sesión mediante la ventana Preferences.

**Seleccione ENABLE para las cuatro advertencias del compilador PL/SQL y, a continuación, haga clic en OK.**



- 7) Recompile el procedimiento UNREACHABLE\_CODE mediante el árbol Object Navigation. ¿Qué advertencias del compilador (si hay alguna) se muestran?

**Haga clic con el botón derecho en el nombre del procedimiento en el árbol Object Navigation y, a continuación, seleccione Compile. Observe los mensajes que aparecen en los subseparadores Messages y Compiler del separador Compiler – Log.**

## Soluciones a la Práctica 10-1: Uso de Parámetros y Advertencias del Compilador PL/SQL (continuación)

The screenshot shows the 'Compiler - Log' tab in SQL Developer. It displays a project named 'Project1.jpr' containing a procedure 'PROCEDURE ORA 61.UNREACHABLE\_CODE@MyDBConnection'. Two warning messages are listed:

- Warning(2,1): PLW-05018: unit UNREACHABLE\_CODE omitted optional AUTHID clause; default value DEFINER used
- Warning(8,5): PLW-06002: Unreachable code

**Nota:** puede que aparezcan las dos advertencias siguientes en SQL Developer, pero esto es algo esperado en algunas versiones de SQL Developer. Si aparecen las siguientes advertencias, es porque la versión de SQL Developer sigue usando el parámetro en desuso PLSQL\_DEBUG de la base de datos Oracle 11g.

```
Warning (1) :PLW-06015:parameter PLSQL_DEBUG is deprecated
; use PLSQL_OPTIMIZE_LEVEL=1
```

```
Warning (1) :PLW-06013:deprecated parameter PLSQL_DEBUG
forces PLSQL_OPTIMIZE_LEVEL<=1
```

- 8) Utilice la vista del diccionario de datos USER\_ERRORS para mostrar los detalles de los mensajes de advertencia del compilador, de la siguiente forma.

```
DESCRIBE user_errors
```

```
DESCRIBE user_errors
Name Null Type

NAME NOT NULL VARCHAR2(30)
TYPE VARCHAR2(12)
SEQUENCE NOT NULL NUMBER
LINE NOT NULL NUMBER
POSITION NOT NULL NUMBER
TEXT NOT NULL VARCHAR2(4000)
ATTRIBUTE VARCHAR2(9)
MESSAGE_NUMBER NUMBER

8 rows selected
```

```
SELECT *
FROM user_errors;
```

The screenshot shows the 'Results' tab in SQL Developer. It displays the description of the 'user\_errors' view followed by the results of a query selecting all columns from the 'user\_errors' view. The results show four rows of data corresponding to the four warnings listed in the log.

| NAME                         | TYPE | SEQUENCE | LINE | POSITION | TEXT                                                                    | ATTRIBUTE | MESSAGE_NUMBER |
|------------------------------|------|----------|------|----------|-------------------------------------------------------------------------|-----------|----------------|
| 1 UNREACHABLE_CODE PROCEDURE |      | 1        | 1    | 1        | PLW-05018: unit UNREACHABLE_CODE omitted optional AUTHID clause; def... | WARNING   | 5018           |
| 2 UNREACHABLE_CODE PROCEDURE |      | 2        | 0    | 0        | PLW-06015: parameter PLSQL_DEBUG is deprecated; use PLSQL_OPTIMIZE_L... | WARNING   | 6015           |
| 3 UNREACHABLE_CODE PROCEDURE |      | 3        | 0    | 0        | PLW-06013: deprecated parameter PLSQL_DEBUG forces PLSQL_OPTIMIZE_L...  | WARNING   | 6013           |
| 4 UNREACHABLE_CODE PROCEDURE |      | 4        | 7    | 5        | PLW-06002: Unreachable code                                             | WARNING   | 6002           |

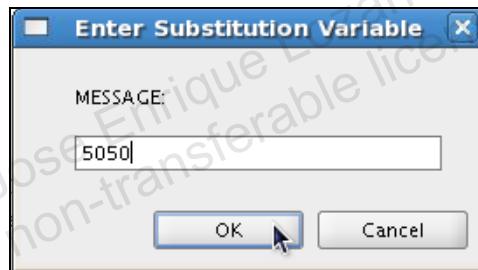
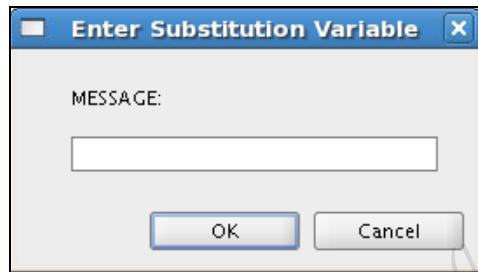
## Soluciones a la Práctica 10-1: Uso de Parámetros y Advertencias del Compilador PL/SQL (continuación)

**Nota:** la salida se mostraba en el separador Results porque hemos usado la tecla F9 para ejecutar la sentencia SELECT. El resultado de la sentencia SELECT podría variar en función de la cantidad de errores que tenga la sesión.

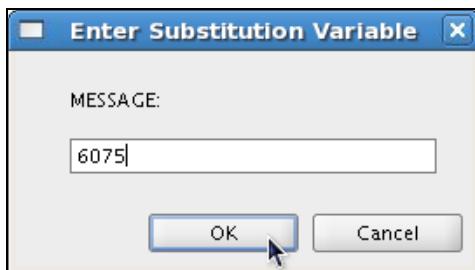
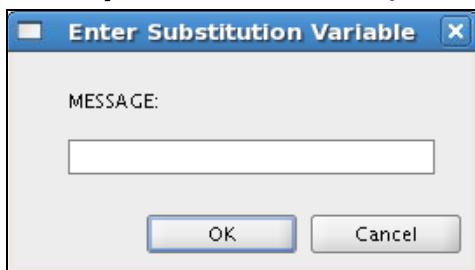
- 9) Cree un script denominado warning\_msgs que utilice los paquetes EXECUTE DBMS\_OUTPUT y DBMS\_WARNING para identificar las categorías para los siguientes números de mensajes de advertencia del compilador: 5050, 6075 y 7100. Active SERVEROUTPUT antes de ejecutar el script.

Abra el script /home/oracle/labs/plpu/solns/sol\_10\_09.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar la consulta. El código y el resultado se muestran a continuación.

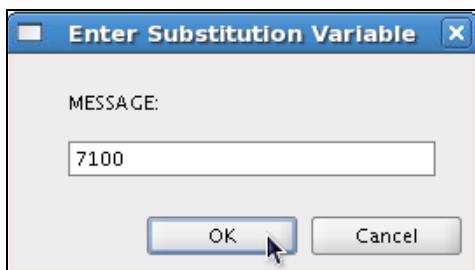
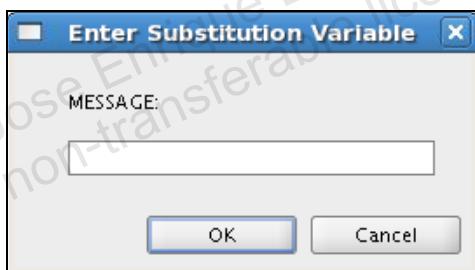
```
EXECUTE
DBMS_OUTPUT.PUT_LINE(DBMS_WARNING.GET_CATEGORY(&message));
```



```
EXECUTE
DBMS_OUTPUT.PUT_LINE(DBMS_WARNING.GET_CATEGORY(&message));
```

**Soluciones a la Práctica 10-1: Uso de Parámetros y Advertencias del Compilador PL/SQL (continuación)**

```
EXECUTE
DBMS_OUTPUT.PUT_LINE(DBMS_WARNING.GET_CATEGORY(&message));
```



**Soluciones a la Práctica 10-1: Uso de Parámetros y Advertencias del Compilador PL/SQL (continuación)**

The screenshot shows the Oracle SQL Developer interface. At the top, there is a toolbar with several buttons: 'Results' (highlighted in green), 'Script Output', 'Explain', 'Autotrace', 'DBMS Output', and 'OWA Output'. Below the toolbar is a menu bar with icons for 'File', 'Edit', 'Tools', 'Database', 'Help', and 'About'. The main area displays the output of an anonymous block. The output starts with 'anonymous block completed' followed by the word 'PERFORMANCE'.

```
anonymous block completed
PERFORMANCE
```

## Prácticas y Soluciones de la Lección 11

En esta práctica, creará un paquete y un procedimiento, en los que se utiliza la compilación condicional. Además, utilizará el paquete adecuado para recuperar el texto de origen post-procesado de la unidad PL/SQL. También ocultará parte del código PL/SQL.

**Nota:** si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

## Práctica 11-1: Uso de la Compilación Condicional

En esta práctica, creará un paquete y un procedimiento, en los que se utiliza la compilación condicional. Además, utilizará el paquete adecuado para recuperar el texto de origen post-procesado de la unidad PL/SQL. También ocultará parte del código PL/SQL.

- 1) Examine y ejecute el script `lab_11_01.sql`. En él, se definen los indicadores para mostrar el código de depuración y la información de rastreo. El script también crea el paquete `my_pkg` y el procedimiento `circle_area`.
- 2) Utilice el subprograma `DBMS_PREPROCESSOR` para recuperar el texto de origen postprocesado de la unidad PL/SQL después de procesar las directivas de compilación condicionales de `lab_11_01`. Active `SERVERTOUTPUT`.
- 3) Cree un script PL/SQL que utilice la constante `DBMS_DB_VERSION` con compilación condicional. El código debe probar la versión de la base de datos de Oracle:
  - a) Si la versión de la base de datos es inferior o igual a la 10.1, debe mostrar el siguiente mensaje de error:  
`Unsupported database release.`
  - b) Si la versión de la base de datos es la 11.1 o superior, debe mostrar el siguiente mensaje:  
`Release 11.1 is supported.`
- 4) Observe el siguiente código en el script `lab_11_04.sql`, que utiliza `CREATE_WWRAPPED` para crear y encapsular de forma dinámica una especificación de paquete y un cuerpo de paquete en una base de datos. Edite el script `lab_11_04.sql` para agregar el código necesario para ocultar el código PL/SQL. Guarde y, a continuación, ejecute el script.

```

DECLARE
 -- the package_text variable contains the text to create
 -- the package spec and body
 package_text VARCHAR2(32767);
 FUNCTION generate_spec (pkgname VARCHAR2) RETURN VARCHAR2
AS
BEGIN
 RETURN 'CREATE PACKAGE ' || pkgname || ' AS
 PROCEDURE raise_salary (emp_id NUMBER, amount NUMBER);
 PROCEDURE fire_employee (emp_id NUMBER);
 END ' || pkgname || ';';
END generate_spec;
FUNCTION generate_body (pkgname VARCHAR2) RETURN VARCHAR2
AS
BEGIN
 RETURN 'CREATE PACKAGE BODY ' || pkgname || ' AS
 PROCEDURE raise_salary (emp_id NUMBER, amount
NUMBER) IS
 BEGIN

```

**Práctica 11-1: Uso de la Compilación Condicional (continuación)**

```
UPDATE employees SET salary = salary + amount
WHERE employee_id = emp_id;
END raise_salary;

PROCEDURE fire_employee (emp_id NUMBER) IS
BEGIN
 DELETE FROM employees WHERE employee_id = emp_id;
END fire_employee;
END ' || pkgname || ';' ;
END generate_body;
```

- a) Genere la especificación del paquete mientras transfiere el parámetro emp\_actions.

- b) Cree y encapsule la especificación del paquete.

- c) Genere el cuerpo del paquete.

- d) Cree y encapsule el cuerpo del paquete.

- e) Llame al procedimiento desde el paquete encapsulado, de la siguiente forma:

```
CALL emp_actions.raise_salary(120, 100);
```

- f) Utilice la vista del diccionario de datos USER\_SOURCE para verificar que el código está oculto, de la siguiente forma:

```
SELECT text FROM USER_SOURCE WHERE name = 'EMP_ACTIONS';
```

## Soluciones a la Práctica 11-1: Uso de la Compilación Condicional

En esta práctica, creará un paquete y un procedimiento, en los que se utiliza la compilación condicional. Además, utilizará el paquete adecuado para recuperar el texto de origen post-procesado de la unidad PL/SQL. También ocultará parte del código PL/SQL.

- 1) Examine y ejecute el script `lab_11_01.sql`. En él, se definen los indicadores para mostrar el código de depuración y la información de rastreo. El script también crea el paquete `my_pkg` y el procedimiento `circle_area`.

**Abra el script `/home/oracle/labs/plpu/solns/sol_11_01.sql`. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```
ALTER SESSION SET PLSQL_CCFLAGS = 'my_debug:FALSE,
my_tracing:FALSE';

CREATE OR REPLACE PACKAGE my_pkg AS
 SUBTYPE my_real IS
 $IF DBMS_DB_VERSION.VERSION < 10 $THEN NUMBER;
-- check database version
 $ELSE BINARY_DOUBLE;
 $END
 my_pi my_real; my_e my_real;
END my_pkg;
/
CREATE OR REPLACE PACKAGE BODY my_pkg AS
BEGIN
 $IF DBMS_DB_VERSION.VERSION < 10 $THEN
 my_pi := 3.14016408289008292431940027343666863227;
 my_e := 2.71828182845904523536028747135266249775;
 $ELSE
 my_pi := 3.14016408289008292431940027343666863227d;
 my_e := 2.71828182845904523536028747135266249775d;
 $END
END my_pkg;
/

CREATE OR REPLACE PROCEDURE circle_area(radius my_pkg.my_real)
IS
 my_area my_pkg.my_real;
 my_datatype VARCHAR2(30);
BEGIN
 my_area := my_pkg.my_pi * radius * radius;
 DBMS_OUTPUT.PUT_LINE('Radius: ' || TO_CHAR(radius)
 || ' Area: ' || TO_CHAR(my_area));
 $IF $$my_debug $THEN
-- if my_debug is TRUE, run some debugging code
 END IF;
 SELECT DATA_TYPE INTO my_datatype FROM USER_ARGUMENTS;
```

## Soluciones a la Práctica 11-1: Uso de la Compilación Condicional (continuación)

```

 WHERE OBJECT_NAME = 'CIRCLE_AREA' AND ARGUMENT_NAME =
'RADIUS';
 DBMS_OUTPUT.PUT_LINE('Datatype of the RADIUS argument is:
' || my_datatype);
$END
END;
/

```



The screenshot shows the Oracle SQL Worksheet interface with the following output:

```

ALTER SESSION SET succeeded.
PACKAGE my_pkg Compiled.
PACKAGE BODY my_pkg Compiled.
PROCEDURE circle_area(radius Compiled.

```

- Utilice el subprograma DBMS\_PREPROCESSOR para recuperar el texto de origen postprocesado de la unidad PL/SQL después de procesar las directivas de compilación condicionales de lab\_11\_01. Active SERVEROUTPUT.

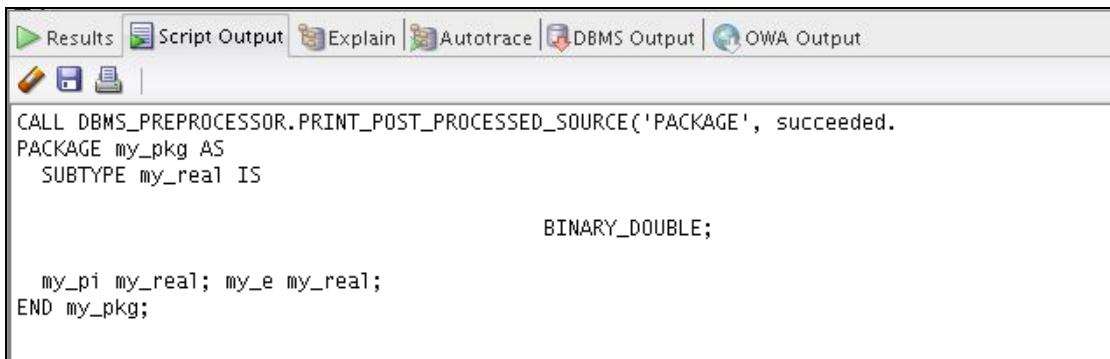
**Abra el script /home/oracle/labs/plpu/solns/sol\_11\_02.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```

-- The code example assumes you are the student with the
-- account ora70. Substitute ora70 with your account
-- information.
SET SERVEROUTPUT ON

CALL DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE('PACKAGE',
'ORA61', 'MY_PKG');

```



The screenshot shows the Oracle SQL Worksheet interface with the following output:

```

CALL DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE('PACKAGE', succeeded.
PACKAGE my_pkg AS
 SUBTYPE my_real IS
 BINARY_DOUBLE;

 my_pi my_real; my_e my_real;
END my_pkg;

```

- Cree un script PL/SQL que utilice la constante DBMS\_DB\_VERSION con compilación condicional. El código debe probar la versión de la base de datos de Oracle:

## Soluciones a la Práctica 11-1: Uso de la Compilación Condicional (continuación)

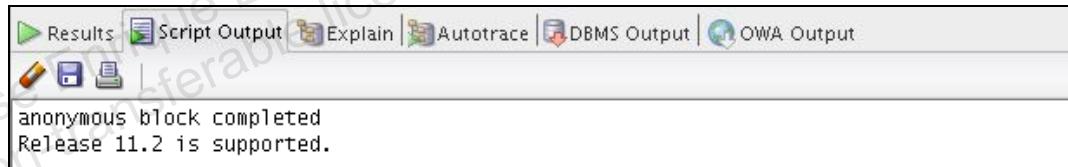
- a) Si la versión de la base de datos es inferior o igual a la 10.1, debe mostrar el siguiente mensaje de error:  
Unsupported database release.
- b) Si la versión de la base de datos es la 11.1 o superior, debe mostrar el siguiente mensaje:  
Release 11,2 is supported.

**Abra el script /home/oracle/labs/plpu/solns/sol\_11\_03.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```
BEGIN
$IF DBMS_DB_VERSION.VER_LE_10_1 $THEN
$ERROR 'unsupported database release.' $END

$ELSE
 DBMS_OUTPUT.PUT_LINE ('Release ' ||
DBMS_DB_VERSION.VERSION || '.' ||
DBMS_DB_VERSION.RELEASE || ' is
supported.');
 -- Note that this COMMIT syntax is newly supported in
10.2
 COMMIT WRITE IMMEDIATE NOWAIT;
$END
END;
/

```



- 4) Observe el siguiente código en el script lab\_11\_04.sql, que utiliza CREATE\_WWRAPPED para crear y encapsular de forma dinámica una especificación de paquete y un cuerpo de paquete en una base de datos. Edite el script lab\_11\_04.sql para agregar el código necesario para ocultar el código PL/SQL. Guarde y, a continuación, ejecute el script.

```
DECLARE
-- the package_text variable contains the text to create
-- the package spec and body
package_text VARCHAR2(32767);
FUNCTION generate_spec (pkgname VARCHAR2) RETURN VARCHAR2
AS
BEGIN
 RETURN 'CREATE PACKAGE ' || pkgname || ' AS
 PROCEDURE raise_salary (emp_id NUMBER, amount NUMBER);
 PROCEDURE fire_employee (emp_id NUMBER);
```

## Soluciones a la Práctica 11-1: Uso de la Compilación Condicional (continuación)

```

 END ' || pkgname || ';' ;
END generate_spec;
FUNCTION generate_body (pkgname VARCHAR2) RETURN VARCHAR2
AS
BEGIN
 RETURN 'CREATE PACKAGE BODY ' || pkgname || ' AS
 PROCEDURE raise_salary (emp_id NUMBER, amount
NUMBER) IS
 BEGIN
 UPDATE employees SET salary = salary + amount
WHERE employee_id = emp_id;
 END raise_salary;

 PROCEDURE fire_employee (emp_id NUMBER) IS
 BEGIN
 DELETE FROM employees WHERE employee_id = emp_id;
 END fire_employee;
 END ' || pkgname || ';' ;
END generate_body;

```

- Genere la especificación del paquete mientras transfiere el parámetro emp\_actions.
- Cree y encapsule la especificación del paquete.
- Genere el cuerpo del paquete.
- Cree y encapsule el cuerpo del paquete.
- Llame al procedimiento desde el paquete encapsulado, de la siguiente forma:  
CALL emp\_actions.raise\_salary(120, 100);
- Utilice la vista del diccionario de datos USER\_SOURCE para verificar que el código está oculto, de la siguiente forma:

```
SELECT text FROM USER_SOURCE WHERE name = 'EMP_ACTIONS';
```

**Abra el script /home/oracle/labs/plpu/sols/soln\_11\_04.sql.**

**Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```

DECLARE
-- the package_text variable contains the text to create
the package spec and body
 package_text VARCHAR2(32767);
 FUNCTION generate_spec (pkgname VARCHAR2) RETURN VARCHAR2
AS
BEGIN
 RETURN 'CREATE PACKAGE ' || pkgname || ' AS
 PROCEDURE raise_salary (emp_id NUMBER, amount
NUMBER);

```

## Soluciones a la Práctica 11-1: Uso de la Compilación Condicional (continuación)

```

 PROCEDURE fire_employee (emp_id NUMBER);
 END ' || pkgname || ';';
END generate_spec;
FUNCTION generate_body (pkgname VARCHAR2) RETURN VARCHAR2
AS
BEGIN
 RETURN 'CREATE PACKAGE BODY ' || pkgname || ' AS
 PROCEDURE raise_salary (emp_id NUMBER, amount
NUMBER) IS
 BEGIN
UPDATE employees SET salary = salary + amount WHERE
employee_id = emp_id;
 END raise_salary;
 PROCEDURE fire_employee (emp_id NUMBER) IS
 BEGIN
 DELETE FROM employees WHERE employee_id = emp_id;
 END fire_employee;
 END ' || pkgname || ';';
END generate_body;

BEGIN

-- generate package spec
 package_text := generate_spec('emp_actions');

-- create and wrap the package spec
 SYS.DBMS_DDL.CREATE_WWRAPPED(package_text);

-- generate package body
 package_text := generate_body('emp_actions');

-- create and wrap the package body
 SYS.DBMS_DDL.CREATE_WWRAPPED(package_text);
END;
/

-- call a procedure from the wrapped package
CALL emp_actions.raise_salary(120, 100);

-- Use the USER_SOURCE data dictionary view to verify that
-- the code is hidden as follows:

SELECT text FROM USER_SOURCE WHERE name = 'EMP_ACTIONS';

```

## **Soluciones a la Práctica 11-1: Uso de la Compilación Condicional (continuación)**

## **Soluciones a la Práctica 11-1: Uso de la Compilación Condicional (continuación)**

## Prácticas y Soluciones de la Lección 12

En esta práctica, utilice el procedimiento DEPTREE\_FILL y la vista IDEPTREE para investigar las dependencias del esquema. Además, recompile vistas, paquetes, funciones y procedimientos no válidos.

**Nota:** si no ha realizado un paso de una práctica, ejecute el script de soluciones adecuado de ese paso de la práctica antes de continuar con el siguiente paso o con la siguiente práctica.

## Práctica 12-1: Gestión de Dependencias en el Esquema

En esta práctica, utilizará el procedimiento DEPTREE\_FILL y la vista IDEPTREE para investigar las dependencias del esquema. Además, volverá a compilar vistas, paquetes, funciones y procedimientos no válidos.

- 1) 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 han creado en la lección titulada “Creación de Funciones”. Puede ejecutar los scripts de soluciones de la práctica 3 si tiene que crear el procedimiento y la función.

- a) Cargue y ejecute el script utldtree.sql, que está en la carpeta /home/oracle/labs/plpu/labs.
- b) Ejecute el procedimiento deptree\_fill para el procedimiento add\_employee.
- c) Consulte la tabla IDEPTREE para ver los resultados.
- d) Ejecute el procedimiento deptree\_fill para la función valid\_deptid.
- e) Consulte la tabla IDEPTREE para ver los resultados.

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

- 2) Valide de forma dinámica los objetos no válidos.
  - a) Realice una copia de la tabla EMPLOYEES, denominada EMPS.
  - b) Modifique la tabla EMPLOYEES y agregue la columna TOTSAL con el tipo de dato NUMBER (9, 2).
  - c) Cree y guarde una consulta para mostrar el nombre, el tipo y el estado de todos los objetos no válidos.
  - d) En compile\_pkg (creado en la práctica 7 de la lección titulada “Uso de SQL Dinámico”), agregue un procedimiento denominado recompile que recompile todos los procedimientos, las funciones y los paquetes no válidos del esquema. Utilice SQL dinámico nativo para modificar el tipo de objeto no válido y compilarlo.
  - e) Ejecute el procedimiento compile\_pkg.recompile.
  - f) Ejecute el archivo de script que ha creado en el paso 3 c. para comprobar el valor de la columna STATUS. ¿Aún tiene objetos con estado INVALID?

## Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema

En esta práctica, utilizará el procedimiento DEPTREE\_FILL y la vista IDEPTREE para investigar las dependencias del esquema. Además, volverá a compilar vistas, paquetes, funciones y procedimientos no válidos.

- 1) 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 han creado en la lección titulada “Creación de Funciones”. Puede ejecutar los scripts de soluciones de la práctica 3 si tiene que crear el procedimiento y la función.

- a) Cargue y ejecute el script utldtree.sql, que está en la carpeta /home/oracle/labs/plpu/labs.

**Abra el script /home/oracle/labs/plpu/sols/utldtree.sql.**

**Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```
Rem
Rem $Header: utldtree.sql,v 1.2 1992/10/26 16:24:44 RKOOI Stab
$
Rem
Rem Copyright (c) 1991 by Oracle Corporation
Rem NAME
Rem deptree.sql - Show objects recursively dependent on
Rem given object
Rem DESCRIPTION
Rem This procedure, view and temp table will allow you to
see Rem all objects that are (recursively) dependent on the
given Rem object.
Rem Note: you will only see objects for which you have
Rem permission.
Rem Examples:
Rem execute deptree_fill('procedure', 'scott', 'billing');
Rem select * from deptree order by seq#;
Rem
Rem execute deptree_fill('table', 'scott', 'emp');
Rem select * from deptree order by seq#;
Rem

Rem execute deptree_fill('package body', 'scott',
'Rem 'accts_payable');
Rem select * from deptree order by seq#;
Rem
Rem A prettier way to display this information than
```

## Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema (continuación)

```

Rem select * from deptree order by seq#;
Rem is
Rem select * from ideptree;
Rem This shows the dependency relationship via indenting.
Rem Notice that no order by clause is needed with ideptree.
Rem RETURNS
Rem
Rem NOTES
Rem Run this script once for each schema that needs this
Rem utility.
Rem MODIFIED (MM/DD/YY)
Rem rkooi 10/26/92 - owner -> schema for SQL2
Rem glumpkin 10/20/92 - Renamed from DEPTREE.SQL
Rem rkooi 09/02/92 - change ORU errors
Rem rkooi 06/10/92 - add rae errors
Rem rkooi 01/13/92 - update for sys vs. regular user
Rem rkooi 01/10/92 - fix ideptree
Rem rkooi 01/10/92 - Better formatting, add ideptree
view
Rem rkooi 12/02/91 - deal with cursors
Rem rkooi 10/19/91 - Creation

DROP SEQUENCE deptree_seq
/
CREATE SEQUENCE deptree_seq cache 200
/* cache 200 to make sequence faster */

/
DROP TABLE deptree_temptab
/
CREATE TABLE deptree_temptab
(
 object_id number,
 referenced_object_id number,
 nest_level number,
 seq# number
)
/
CREATE OR REPLACE PROCEDURE deptree_fill (type char, schema
char, name char) IS
 obj_id number;
BEGIN
 DELETE FROM deptree_temptab;
 COMMIT;
 SELECT object_id INTO obj_id FROM all_objects
 WHERE owner = upper(deptree_fill.schema)

 AND object_name = upper(deptree_fill.name)
 AND object_type = upper(deptree_fill.type);
 INSERT INTO deptree_temptab

```

## Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema (continuación)

```

VALUES(obj_id, 0, 0, 0);
INSERT INTO deptree_temptab
 SELECT object_id, referenced_object_id,
 level, deptree_seq.nextval
 FROM public_dependency
 CONNECT BY PRIOR object_id = referenced_object_id
 START WITH referenced_object_id = deptree_fill.obj_id;
EXCEPTION
 WHEN no_data_found then
 raise_application_error(-20000, 'ORU-10013: ' ||
 type || ' ' || schema || '.' || name || ' was not
found.');
END;
/
DROP VIEW deptree
/
SET ECHO ON

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

SET ECHO OFF
CREATE VIEW sys.deptree
 (nested_level, type, schema, name, seq#)
AS
 SELECT d.nest_level, o.object_type, o.owner, o.object_name,
d.seq#
 FROM deptree_temptab d, dba_objects o
 WHERE d.object_id = o.object_id (+)
UNION ALL
 SELECT d.nest_level+1, 'CURSOR', '<shared>',
"'||c.kglnaobj||''', d.seq#+.5
 FROM deptree_temptab d, x$kgldp k, x$kglob g, obj$ o, user$u,
x$kglob c,
 x$kglxss a
 WHERE d.object_id = o.obj#
 AND o.name = g.kglnaobj
 AND o.owner# = u.user#
 AND u.name = g.kglnaown
 AND g.kglhdadr = k.kglrfhdl
 AND k.kglhdadr = a.kglhdadr /* make sure it is not a
transitive */
 AND k.kgldepno = a.kglxsdep /* reference, but a direct
one */
 AND k.kglhdadr = c.kglhdadr

```

## Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema (continuación)

```

 AND c.kglhdnsp = 0 /* a cursor */
/

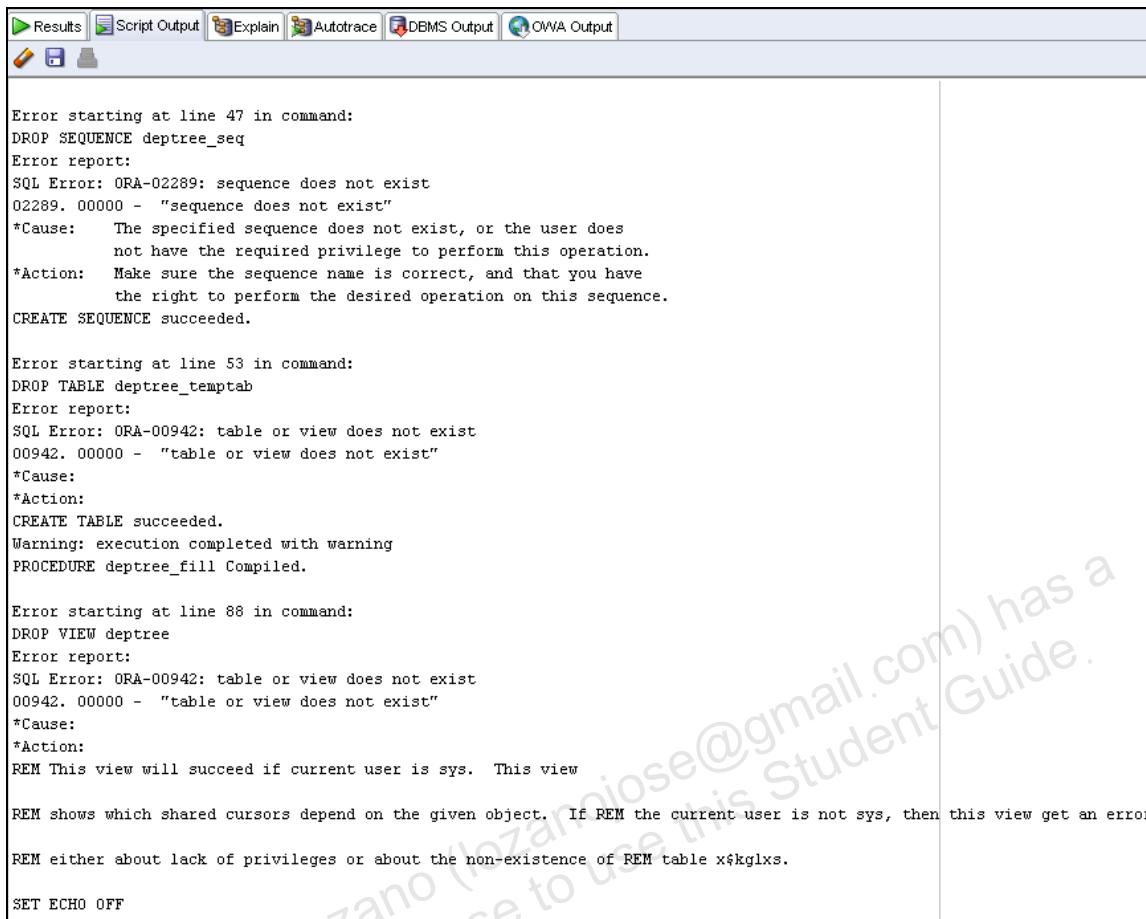
SET ECHO ON

REM This view will succeed if current user is not sys. This
view
REM does *not* show which shared cursors depend on the given
REM 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
REM create will have succeeded).

SET ECHO OFF
CREATE VIEW deptree
 (nested_level, type, schema, name, seq#)
AS
 select d.nest_level, o.object_type, o.owner, o.object_name,
d.seq#
 FROM deptree temptab d, all_objects o
 WHERE d.object_id = o.object_id (+)
/
DROP VIEW ideptree
/
CREATE VIEW ideptree (dependencies)
AS
 SELECT lpad(' ',3*(max(nested_level))) || max(nvl(type, '<no
permission>'))
 || ' ' || schema || decode(type, NULL, '', '.') || name)
 FROM deptree
 GROUP BY seq# /* So user can omit sort-by when selecting
from ideptree */
/

```

## Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema (continuación)



The screenshot shows the Oracle SQL Developer interface with several tabs at the top: Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. Below the tabs, there are two toolbar icons: a pencil and a trash can.

```

Error starting at line 47 in command:
DROP SEQUENCE deptree_seq
Error report:
SQL Error: ORA-02289: sequence does not exist
02289. 00000 - "sequence does not exist"
*Cause: The specified sequence does not exist, or the user does
 not have the required privilege to perform this operation.
*Action: Make sure the sequence name is correct, and that you have
 the right to perform the desired operation on this sequence.
CREATE SEQUENCE succeeded.

Error starting at line 53 in command:
DROP TABLE deptree temptab
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:
CREATE TABLE succeeded.
Warning: execution completed with warning
PROCEDURE deptree_fill Compiled.

Error starting at line 88 in command:
DROP VIEW deptree
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:
REM This view will succeed if current user is sys. This view
REM shows which shared cursors depend on the given object. If REM the current user is not sys, then this view get an error
REM either about lack of privileges or about the non-existence of REM table x$kglixs.

SET ECHO OFF

```

## Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema (continuación)

```
Error starting at line 98 in command:
CREATE VIEW sys.deptree
 (nested_level, type, schema, name, seq#)
AS
 SELECT d.nest_level, o.object_type, o.owner, o.object_name, d.seq#
 FROM deptree_temptab d, dba_objects o
 WHERE d.object_id = o.object_id (+)
UNION ALL
 SELECT d.nest_level+1, 'CURSOR', '<shared>', '"'||c.kglnaobj||"', d.seq#+.5
 FROM deptree_temptab d, x$kgldp k, x$kglob g, obj$ o, user$ u, x$kglob c,
 x$kglx$ a
 WHERE d.object_id = o.obj#
 AND o.name = g.kglnaobj
 AND o.owner# = u.user#
 AND u.name = g.kglnaown
 AND g.kglhdadr = k.kglrfhdl
 AND k.kglhdadr = a.kglhdadr /* make sure it is not a transitive */
 AND k.kgldepno = a.kglxsdep /* reference, but a direct one */
 AND k.kglhdadr = c.kglhdadr
 AND c.kglhdnsp = 0 /* a cursor */
Error at Command Line:102 Column:7
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:
REM This view will succeed if current user is not sys. This view

REM does *not* show which shared cursors depend on the given

REM 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
```

```
REM create will have succeeded).

SET ECHO OFF

CREATE VIEW succeeded.

Error starting at line 136 in command:
DROP VIEW ideptree
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:
CREATE VIEW succeeded.
```

## Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema (continuación)

- b) Ejecute el procedimiento deptree\_fill para el procedimiento add\_employee.

**Abra el script /home/oracle/labs/plpu/solns/sol\_12\_01\_b.sql.**

**Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

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

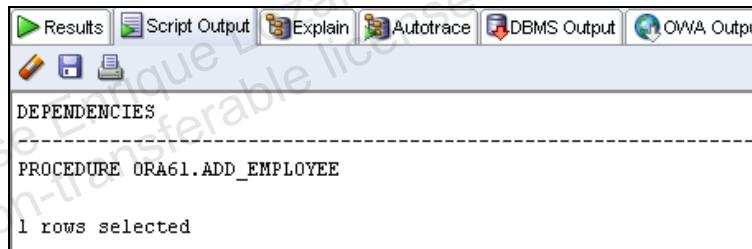


- c) Consulte la tabla IDEPTREE para ver los resultados.

**Abra el script /home/oracle/labs/plpu/solns/sol\_12\_01\_c.sql.**

**Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```
SELECT * FROM IDEPTREE;
```



- d) Ejecute el procedimiento deptree\_fill para la función valid\_deptid.

**Abra el script /home/oracle/labs/plpu/solns/sol\_12\_01\_d.sql.**

**Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

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

## Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema (continuación)

The screenshot shows the Oracle SQL Worksheet interface. The toolbar at the top includes icons for Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. Below the toolbar, there are three small icons: a pencil, a floppy disk, and a printer. The main area displays the message "anonymous block completed".

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

**Abra el script /home/oracle/labs/plpu/solns/sol\_12\_01\_e.sql. Haga clic en el icono Execute Statement (F9) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

```
SELECT * FROM IDEPTREE;
```

The screenshot shows the Oracle SQL Worksheet interface. The toolbar at the top includes icons for Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. Below the toolbar, there are three small icons: a pencil, a floppy disk, and a printer. The main area displays the message "DEPENDENCIES" followed by a dashed line. Underneath the dashed line, it shows "PROCEDURE ORA61.ADD\_EMPLOYEE" and "FUNCTION ORA61.VALID\_DEPTID". At the bottom, it says "2 rows selected".

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

- 2) Valide de forma dinámica los objetos no válidos.

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

**Abra el script /home/oracle/labs/plpu/solns/sol\_12\_02\_a.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

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

The screenshot shows the Oracle SQL Worksheet interface. The toolbar at the top includes icons for Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. Below the toolbar, there are three small icons: a pencil, a floppy disk, and a printer. The main area displays the message "CREATE TABLE succeeded."

## Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema (continuación)

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

Abra el script /home/oracle/labs/plpu/solns/sol\_12\_02\_b.sql.

Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.

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

ALTER TABLE employees succeeded.

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

Abra el script /home/oracle/labs/plpu/solns/sol\_12\_02\_c.sql.

Haga clic en el icono Execute Statement (F9) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.

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

| OBJECT_NAME          | OBJECT_TYPE  | STATUS  |
|----------------------|--------------|---------|
| 1 EMP_ACTIONS        | PACKAGE BODY | INVALID |
| 2 UPD_MINSALARY_TRG  | TRIGGER      | INVALID |
| 3 DELETE_EMP_TRG     | TRIGGER      | INVALID |
| 4 EMP_PKG            | PACKAGE BODY | INVALID |
| 5 EMP_PKG            | PACKAGE      | INVALID |
| 6 GET_EMPLOYEE       | PROCEDURE    | INVALID |
| 7 UPDATE_JOB_HISTORY | TRIGGER      | INVALID |
| 8 SECURE_EMPLOYEES   | TRIGGER      | INVALID |

## Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema (continuación)

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

**Abra el script /home/oracle/labs/plpu/solns/sol\_12\_02\_d.sql. Haga clic en el ícono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación. El código que se acaba de agregar se resalta en negrita en el cuadro de código siguiente.**

```

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;

 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

```

## Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema (continuación)

```

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;
/
SHOW ERRORS

```

The screenshot shows the Oracle SQL Developer interface with the results of a script execution. The top menu bar includes 'Results', 'Script Output', 'Explain', 'Autotrace', 'DBMS Output', and 'OWA Output'. Below the menu is a toolbar with icons for edit, run, and save. The main pane displays the following text:

```

PACKAGE compile_pkg Compiled.
No Errors.
PACKAGE BODY compile_pkg Compiled.
No Errors.

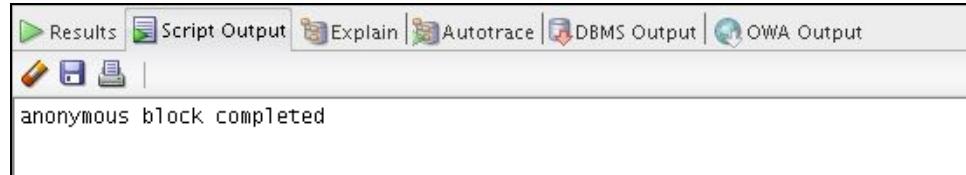
```

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

Abra el script `/home/oracle/labs/plpu/solns/sol_12_02_e.sql`. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.

## Soluciones a la Práctica 12-1: Gestión de Dependencias en el Esquema (continuación)

```
EXECUTE compile_pkg.recompile
```



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

**Abra el script /home/oracle/labs/plpu/solns/sol\_12\_02\_f.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para ejecutar el script. El código y el resultado se muestran a continuación.**

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

| Results:      |              |         |
|---------------|--------------|---------|
| OBJECT_NAME   | OBJECT_TYPE  | STATUS  |
| 1 EMP_ACTIONS | PACKAGE BODY | INVALID |

---

## **Apéndice AP**

### **Adicional: Prácticas y Soluciones**

---

Jose Enrique Lozano (lozanojose@gmail.com) has a  
non-transferable license to use this Student Guide.

## Tabla de Contenido

|                                                                                                                                           |    |
|-------------------------------------------------------------------------------------------------------------------------------------------|----|
| Práctica 1.....                                                                                                                           | 3  |
| Práctica 1-1: Creación de una Nueva Conexión de Base de Datos de SQL Developer.....                                                       | 4  |
| Práctica 1-2: Adición de un Nuevo Trabajo a la Tabla JOBS .....                                                                           | 6  |
| Práctica 1-3: Adición de una Nueva Fila a la Tabla JOB_HISTORY .....                                                                      | 7  |
| Práctica 1-4: Actualización de los Salarios Mínimos y Máximos de un Trabajo .....                                                         | 8  |
| Práctica 1-5: Supervisión de los Salarios de los Empleados.....                                                                           | 9  |
| Práctica 1-6: Recuperación del Número Total de Años de Servicio de un Empleado .....                                                      | 10 |
| Práctica 1-7: Recuperación del Número Total de Trabajos Diferentes de un Empleado .....                                                   | 11 |
| Práctica 1-8: Creación de un Nuevo Paquete que Contenga los Procedimientos y las Funciones Recién Creados .....                           | 12 |
| Práctica 1-9: Creación de un Disparador para Garantizar que los Salarios de los Empleados Estén en un Rango Aceptable .....               | 13 |
| Soluciones a la Práctica: 1-1: Creación de una Nueva Conexión de Base de Datos de SQL Developer .....                                     | 14 |
| Soluciones a la Práctica 1-2: Adición de un Nuevo Trabajo a la Tabla JOBS .....                                                           | 16 |
| Soluciones a la Práctica 1-3: Adición de una Nueva Fila a la Tabla JOB_HISTORY .....                                                      | 19 |
| Solución a la Práctica 1-4: Actualización de los Salarios Mínimos y Máximos de un Trabajo .....                                           | 23 |
| Soluciones a la Práctica 1-5: Supervisión de los Salarios de los Empleados .....                                                          | 28 |
| Soluciones a la Práctica 1-6: Recuperación del Número Total de Años de Servicio de un Empleado .....                                      | 33 |
| Soluciones a la Práctica 1-7: Recuperación del Número Total de Trabajos Diferentes de un Empleado .....                                   | 37 |
| Soluciones a la Práctica 1-8: Creación de un Nuevo Paquete que Contenga los Procedimientos y las Funciones Recién Creados.....            | 40 |
| Solución a la Práctica 1-9: Creación de un Disparador para Garantizar que los Salarios de los Empleados Estén en un Rango Aceptable ..... | 47 |
| Práctica 2.....                                                                                                                           | 51 |
| Práctica 2-1: Creación del Paquete VIDEO_PKG.....                                                                                         | 52 |
| Soluciones a la Práctica 2-1: Creación del Paquete VIDEO_PKG .....                                                                        | 54 |

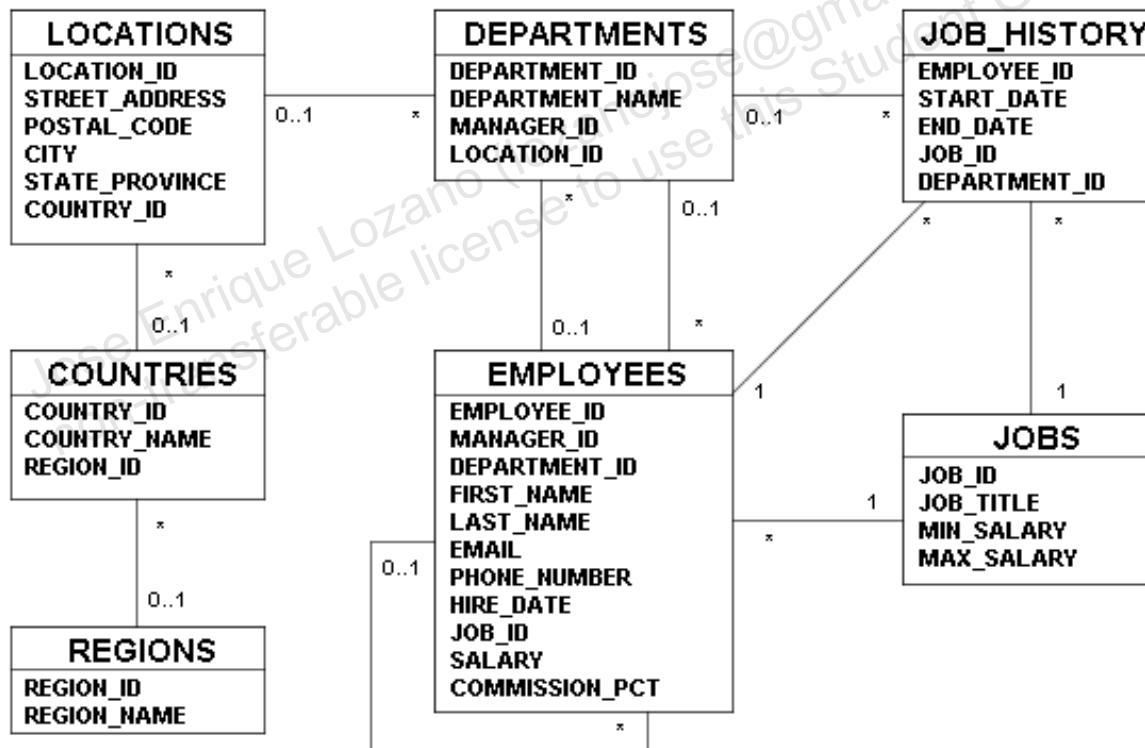
## Práctica 1

Las prácticas adicionales se proporcionan como complemento del curso *Oracle Database: Desarrollo de Unidades de Programa en PL/SQL*. En estas prácticas se aplican los conceptos que ha aprendido a lo largo del curso. Las prácticas adicionales constan de dos lecciones.

La Lección 1 proporciona ejercicios complementarios para crear procedimientos almacenados, funciones, paquetes y disparadores, así como para utilizar los paquetes proporcionados por Oracle con SQL Developer o SQL\*Plus como entorno de desarrollo. Las tablas utilizadas en esta parte de las prácticas adicionales son EMPLOYEES, JOBS, JOB\_HISTORY y DEPARTMENTS.

Al principio de cada práctica se proporciona un diagrama de entidad/relación. Cada diagrama de entidad/relación muestra las entidades de tabla y sus relaciones. Para obtener definiciones más detalladas de las tablas y los datos que contienen las tablas, consulte el apéndice titulado “Prácticas Adicionales: Descripciones de las Tablas y Datos”.

### Diagrama de Entidad/Relación del Esquema Human Resources (HR)

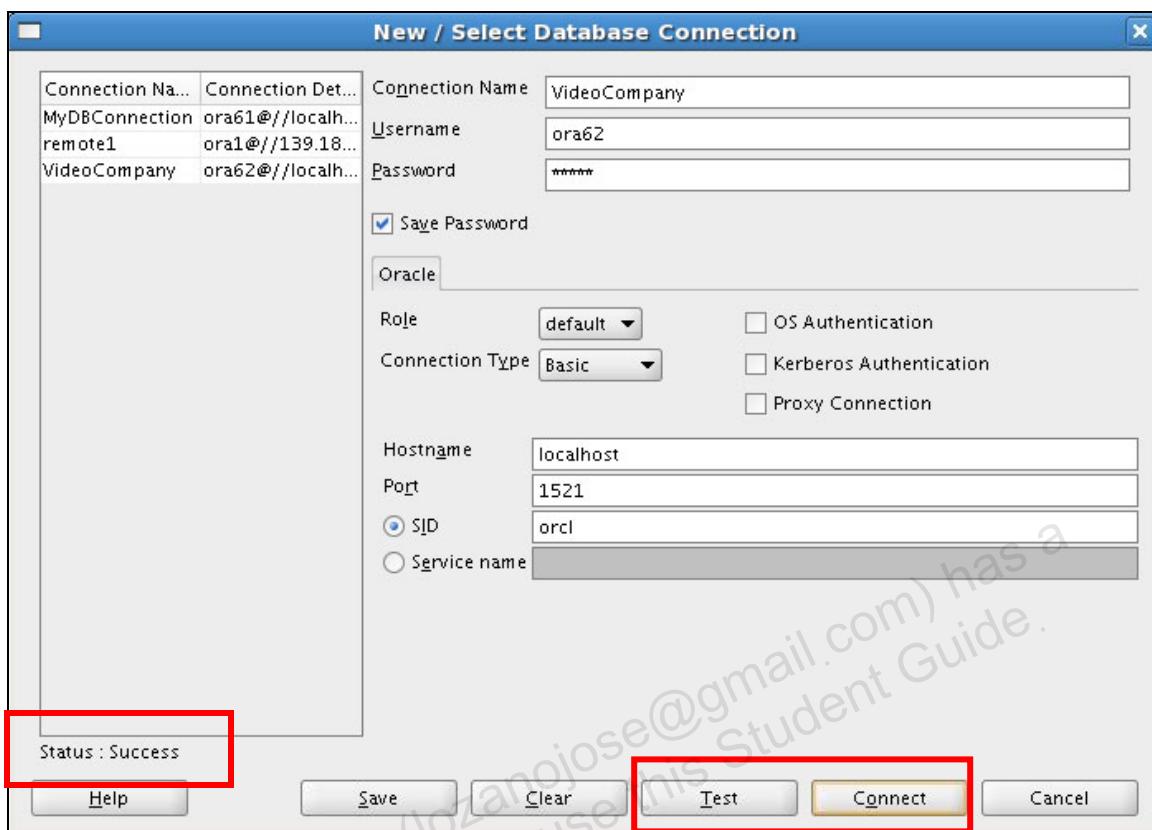


## Práctica 1-1: Creación de una Nueva Conexión de Base de Datos de SQL Developer

En esta práctica, iniciará SQL Developer mediante la información de conexión y creará una nueva conexión de base de datos.

Inicie SQL Developer mediante el identificador y la contraseña de usuario que le ha proporcionado el instructor, por ejemplo ora62.

- 1) Cree una conexión de base de datos con la siguiente información:
  - a) Connection Name: VideoCompany
  - b) Username: ora62
  - c) Password: ora62
  - d) Hostname: introduzca el nombre de host de la computadora
  - e) Port: 1521
  - f) SID: ORCL
- 2) Pruebe la nueva conexión. Si el estado es Success, conecte a la base de datos con esta nueva conexión:
  - a) Haga clic dos veces en el ícono VideoCompany de la página con separadores Connections.
  - b) Haga clic en el botón Test de la ventana New>Select Database Connection. Si el estado es Success, haga clic en el botón Connect.

**Práctica 1-1: Creación de una Nueva Conexión de Base de Datos de SQL Developer (continuación)**

## Práctica 1-2: Adición de un Nuevo Trabajo a la Tabla JOBS

En esta práctica, creará un subprograma para agregar un nuevo trabajo a la tabla JOBS.

- 1) Cree un procedimiento almacenado denominado NEW\_JOB para introducir un nuevo trabajo en la tabla JOBS. El procedimiento debe aceptar tres parámetros. El primero y el segundo proporcionan un identificador de trabajo y un cargo. El tercero proporciona el salario mínimo. Utilice el salario máximo del nuevo trabajo como el doble del salario mínimo proporcionado para el identificador de trabajo.
- 2) Active SERVEROUTPUT y, a continuación, llame al procedimiento para agregar un nuevo trabajo con el identificador 'SY\_ANAL', el cargo 'System Analyst' y un salario mínimo de 6000.
- 3) Compruebe si se ha agregado una fila y anote el identificador del nuevo trabajo para utilizarlo en el siguiente ejercicio. Confirme los cambios.

### Práctica 1-3: Adición de una Nueva Fila a la Tabla JOB\_HISTORY

En esta práctica, agregará una nueva fila a la tabla JOB\_HISTORY para un empleado existente.

- 1) Cree un procedimiento almacenado denominado ADD\_JOB\_HIST para agregar una nueva fila a la tabla JOB\_HISTORY para un empleado que está cambiando su trabajo al nuevo identificador de trabajo ('SY\_ANAL') creado en el ejercicio 1 b.
  - a) El procedimiento debe proporcionar dos parámetros, uno para el identificador del empleado que está cambiando el trabajo y el otro para el identificador del nuevo trabajo.
  - b) Lea el identificador del empleado de la tabla EMPLOYEES e insértelo en la tabla JOB\_HISTORY.
  - c) Indique la fecha de contratación de este empleado como fecha de inicio y la fecha de hoy como fecha final para esta fila en la tabla JOB\_HISTORY.
  - d) Cambie la fecha de contratación de este empleado en la tabla EMPLOYEES a la fecha de hoy.
  - e) Actualice el identificador de trabajo de este empleado según el identificador de trabajo transferido como parámetro (utilice el identificador de trabajo 'SY\_ANAL') y el salario igual al salario mínimo para ese identificador de trabajo + 500.

**Nota:** incluya el manejo de excepciones para manejar el intento de insertar un empleado no existente.

- 2) Desactive todos los disparadores de las tablas EMPLOYEES, JOBS y JOB\_HISTORY antes de llamar al procedimiento ADD\_JOB\_HIST.
- 3) Active SERVEROUTPUT y, a continuación, ejecute el procedimiento con el identificador de empleado 106 y el identificador de trabajo 'SY\_ANAL' como parámetros.
- 4) Consulte las tablas JOB\_HISTORY y EMPLOYEES para ver los cambios del empleado 106 y, a continuación, confirme los cambios.
- 5) Vuelva a activar los disparadores en las tablas EMPLOYEES, JOBS y JOB\_HISTORY.

## Práctica 1-4: Actualización de los Salarios Mínimos y Máximos de un Trabajo

En esta práctica, creará un programa para actualizar los salarios mínimos y máximos de un trabajo de la tabla JOBS.

- 1) Cree un procedimiento almacenado denominado UPD\_JOBSAL para actualizar los salarios mínimos y máximos de un identificador de trabajo concreto de la tabla JOBS. El procedimiento debe proporcionar tres parámetros: el identificador de trabajo, un nuevo salario mínimo y un nuevo salario máximo. Agregue el manejo de excepciones para justificar un identificador de trabajo no válido en la tabla JOBS. Emite una excepción si el salario máximo proporcionado es menor que el salario mínimo, y proporcione un mensaje que se mostrará si la fila de la tabla JOBS está bloqueada.  
**Indicación:** el número de error de recurso bloqueado/ocupado es -54.
- 2) Active SERVEROUTPUT y, a continuación, ejecute el procedimiento UPD\_JOBSAL mediante un identificador de trabajo 'SY\_ANAL', un salario mínimo de 7000 y un salario máximo de 140.  
**Nota:** esto debe generar un mensaje de excepción.
- 3) Desactive los disparadores de las tablas EMPLOYEES y JOBS.
- 4) Ejecute el procedimiento UPD\_JOBSAL utilizando un identificador de trabajo 'SY\_ANAL', un salario mínimo de 7000 y un salario máximo de 14000.
- 5) Consulte la tabla JOBS para ver los cambios y, a continuación, confírmelos.
- 6) Active los disparadores de las tablas EMPLOYEES y JOBS.

## Práctica 1-5: Supervisión de los Salarios de los Empleados

En esta práctica, creará un procedimiento para supervisar si los salarios de los empleados han excedido la media para el tipo de trabajo.

- 1) Desactive el disparador SECURE\_EMPLOYEES.
- 2) En la tabla EMPLOYEES, agregue una columna EXCEED\_AVGSAL para almacenar hasta tres caracteres y un valor por defecto NO. Utilice una restricción de control para permitir los valores YES o NO.
- 3) Cree un procedimiento almacenado denominado CHECK\_AVGSAL que compruebe si el salario de cada empleado excede el salario medio para JOB\_ID.
  - a) El salario medio de un trabajo se calcula a partir de la información de la tabla JOBS.
  - b) Si el salario del empleado excede la media para el trabajo, actualice la columna EXCEED\_AVGSAL de la tabla EMPLOYEES con el valor YES; de lo contrario, defina el valor en NO.
  - c) Utilice un cursor para seleccionar las filas del empleado con la opción FOR UPDATE de la consulta.
  - d) Agregue el manejo de excepciones para justificar el bloqueo de un registro.  
**Indicación:** el número de error de recurso bloqueado/ocupado es -54.
  - e) Escriba y utilice una función local denominada GET\_JOB\_AVGSAL para determinar el salario medio de un identificador de trabajo especificado como parámetro.
- 4) Ejecute el procedimiento CHECK\_AVGSAL. Para ver los resultados de las modificaciones, escriba una consulta para mostrar el identificador del empleado, el trabajo, el salario medio del trabajo, el salario del empleado y la columna del indicador exceed\_avgsal para empleados cuyos salarios exceden la media para el trabajo. Por último, confirme los cambios.

**Nota:** estos ejercicios se pueden utilizar como práctica adicional al describir cómo crear funciones.

## Práctica 1-6: Recuperación del Número Total de Años de Servicio de un Empleado

En esta práctica, creará un subprograma para recuperar el número de años de servicio de un empleado concreto.

- 1) Cree una función almacenada denominada GET\_YEARS\_SERVICE para recuperar el número total de años de servicio de un empleado concreto. La función debe aceptar el identificador de empleado como parámetro y devolver el número de años de servicio. Agregue la gestión de errores para justificar un identificador de empleado no válido.
- 2) Llame a la función GET\_YEARS\_SERVICE en una llamada a DBMS\_OUTPUT.PUT\_LINE para un empleado que tenga el identificador 999.
- 3) Muestre el número de años de servicio para el empleado 106 mediante la llamada de DBMS\_OUTPUT.PUT\_LINE a la función GET\_YEARS\_SERVICE. Asegúrese de activar SERVEROUTPUT.
- 4) Consulte los datos del empleado concreto en las tablas JOB\_HISTORY y EMPLOYEES para verificar que las modificaciones son precisas. Los valores representados en los resultados de esta página pueden variar de los valores obtenidos al ejecutar las consultas.

## Práctica 1-7: Recuperación del Número Total de Trabajos Diferentes de un Empleado

En esta práctica, creará un programa para recuperar el número de trabajos diferentes que ha tenido un empleado durante su servicio.

- 1) Cree una función almacenada denominada GET\_JOB\_COUNT para recuperar el número total de trabajos diferentes que ha tenido un empleado.
  - a) La función debe aceptar el identificador de empleado en un parámetro y devolver el número de trabajos diferentes que ha tenido un empleado hasta la fecha, incluido el presente.
  - b) Agregue el manejo de excepciones para justificar un identificador de empleado no válido.

**Indicación:** utilice los distintos identificadores de trabajo de la tabla JOB\_HISTORY y excluya el identificador de trabajo actual, si se trata de uno de los identificadores de trabajo en los que ya ha trabajado el empleado.

  - c) Una dos consultas con UNION y cuente las filas recuperadas en una tabla PL/SQL.
  - d) Utilice FETCH con BULK COLLECT INTO para obtener los trabajos únicos del empleado.
- 2) Llame a la función para el empleado con el identificador 176. Asegúrese de activar SERVEROUTPUT.

**Nota:** estos ejercicios se pueden utilizar como práctica adicional al describir cómo crear paquetes.

## Práctica 1-8: Creación de un Nuevo Paquete que Contenga los Procedimientos y las Funciones Recién Creados

En esta práctica, creará un paquete denominado `EMPJOB_PKG`, que contiene los procedimientos `NEW_JOB`, `ADD_JOB_HIST` y `UPD_JOBSAL`, así como las funciones `GET_YEARS_SERVICE` y `GET_JOB_COUNT`.

- 1) Cree la especificación del paquete con toda la construcción del subprograma pública. Mueva cualquier tipo definido localmente del subprograma a la especificación del paquete.
- 2) Cree el cuerpo del paquete con la implantación del subprograma; recuerde eliminar de las implantaciones del subprograma cualquier tipo que haya movido a la especificación del paquete.
- 3) Llame al procedimiento `EMPJOB_PKG.NEW_JOB` para crear un nuevo trabajo con el identificador `PR_MAN`, el cargo Public Relations Manager y el salario 6250. Asegúrese de activar SERVEROUTPUT.  
**Nota:** debe desactivar el disparador `UPDATE_JOB_HISTORY` antes de ejecutar el procedimiento `ADD_JOB_HIST` y volver a activar el disparador después de ejecutar el procedimiento.
- 4) Llame al procedimiento `EMPJOB_PKG.ADD_JOB_HIST` para modificar el trabajo del empleado con identificador 110 por el identificador de trabajo `PR_MAN`.  
**Nota:** estos ejercicios se pueden utilizar como práctica adicional al describir cómo crear disparadores de base de datos.
- 5) Consulte las tablas `JOBS`, `JOB_HISTORY` y `EMPLOYEES` para verificar los resultados.

## **Práctica 1-9: Creación de un Disparador para Garantizar que los Salarios de los Empleados Estén en un Rango Aceptable**

En esta práctica, creará un disparador para asegurarse de que los salarios mínimos y máximos de un trabajo nunca se modifiquen de forma que el salario de un empleado existente con ese identificador de trabajo esté fuera del nuevo rango especificado para el trabajo.

- 1) Cree un disparador denominado CHECK\_SAL\_RANGE que se dispare antes de cada fila que se actualice en las columnas MIN\_SALARY y MAX\_SALARY de la tabla JOBS.
  - a) Para cada valor de salario mínimo o máximo que se cambie, compruebe si el salario de cualquier empleado existente con ese identificador de trabajo de la tabla EMPLOYEES entra dentro del nuevo rango de salarios especificado para este identificador de trabajo.
  - b) Incluya el manejo de excepciones para cubrir un cambio de rango de salarios que afecte al registro de cualquier empleado existente.
- 2) Pruebe el disparador utilizando el trabajo SY\_ANAL y defina el nuevo salario mínimo en 5000 y el nuevo salario máximo en 7000. Antes de realizar los cambios, escriba una consulta para mostrar el rango de salarios actual para el identificador de trabajo SY\_ANAL y otra consulta para mostrar el identificador, apellido y salario del empleado para el mismo identificador de trabajo. Después de la actualización, consulte los cambios (si existen) realizados en la tabla JOBS para el identificador de trabajo especificado.
- 3) Mediante el trabajo SY\_ANAL, defina el nuevo salario mínimo en 7000 y el nuevo salario máximo en 18000. Explique los resultados.

## **Soluciones a la Práctica: 1-1: Creación de una Nueva Conexión de Base de Datos de SQL Developer**

En esta práctica, iniciará SQL Developer mediante la información de conexión y creará una nueva conexión de base de datos.

- 1) Inicie SQL Developer mediante el identificador y la contraseña de usuario que le ha proporcionado el instructor, por ejemplo ora62.

**Haga clic en el ícono SQL Developer del escritorio.**

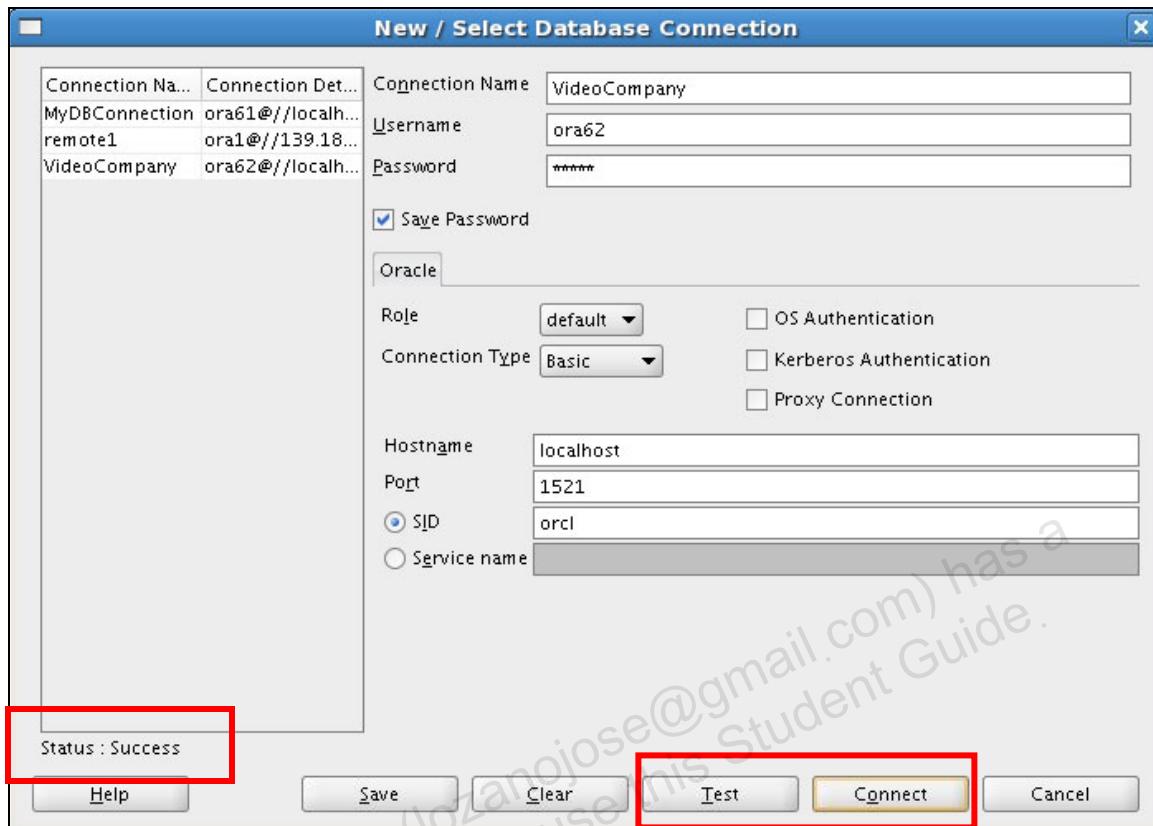


- 2) Cree una conexión de base de datos con la siguiente información:
  - a) Connection Name: VideoCompany
  - b) Username: ora62
  - c) Password: ora62
  - d) Hostname: introduzca el nombre de host de la computadora
  - e) Port: 1521
  - f) SID: ORCL

**Haga clic con el botón derecho en el ícono Connections de la página con separadores Connections y, a continuación, seleccione la opción New Connection en el menú de acceso directo. Se muestra la ventana New>Select Database Connection. Utilice la información que se ha proporcionado anteriormente para crear la nueva conexión de base de datos.**

**Nota:** para mostrar las propiedades de la conexión que se acaba de crear, haga clic con el botón derecho en el nombre de la conexión y, a continuación, seleccione Properties en el menú de acceso directo. Sustituya el nombre de usuario, la contraseña, el nombre de host y el nombre de servicio por la información correspondiente que le haya proporcionado su instructor. A continuación se muestra un ejemplo de la conexión a la base de datos que se acaba de crear para el estudiante ora62:

- 3) Pruebe la nueva conexión. Si el estado es Success, conecte a la base de datos con esta nueva conexión:
  - a) Haga clic dos veces en el ícono VideoCompany de la página con separadores Connections.
  - b) Haga clic en el botón Test de la ventana New>Select Database Connection. Si el estado es Success, haga clic en el botón Connect.

**Soluciones a la Práctica: 1-1: Creación de una Nueva Conexión de Base de Datos de SQL Developer (continuación)**

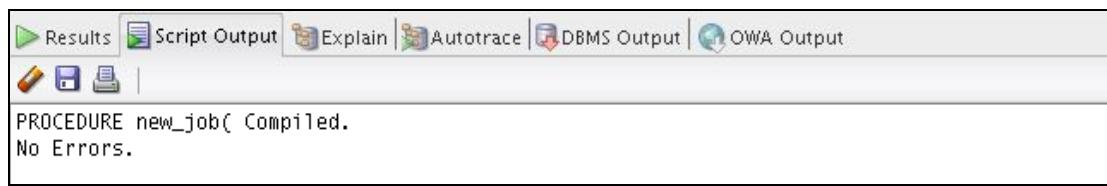
## Soluciones a la Práctica 1-2: Adición de un Nuevo Trabajo a la Tabla JOBS

En esta práctica, creará un subprograma para agregar un nuevo trabajo a la tabla JOBS.

- Cree un procedimiento almacenado denominado NEW\_JOB para introducir un nuevo trabajo en la tabla JOBS. El procedimiento debe aceptar tres parámetros. El primero y el segundo proporcionan un identificador de trabajo y un cargo. El tercero proporciona el salario mínimo. Utilice el salario máximo del nuevo trabajo como el doble del salario mínimo proporcionado para el identificador de trabajo.

**Abra el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_02\_01.sql. Haga clic en el icono Run Script (F5) de la barra de herramientas de SQL Worksheet para crear y compilar el procedimiento. Cuando se le pida que seleccione una conexión, seleccione la nueva conexión VideoCompany. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:**

```
CREATE OR REPLACE PROCEDURE new_job(
 p_jobid IN jobs.job_id%TYPE,
 p_title IN jobs.job_title%TYPE,
 v_minsal IN jobs.min_salary%TYPE) IS
 v_maxsal jobs.max_salary%TYPE := 2 * v_minsal;
BEGIN
 INSERT INTO jobs(job_id, job_title, min_salary, max_salary)
 VALUES (p_jobid, p_title, v_minsal, v_maxsal);
 DBMS_OUTPUT.PUT_LINE ('New row added to JOBS table:');
 DBMS_OUTPUT.PUT_LINE (p_jobid || ' ' || p_title || ' ' ||
 v_minsal || ' ' || v_maxsal);
END new_job;
/
SHOW ERRORS
```



## **Soluciones a la Práctica 1-2: Adición de un Nuevo Trabajo a la Tabla JOBS (continuación)**

- 2) Active SERVEROUTPUT y, a continuación, llame al procedimiento para agregar un nuevo trabajo con el identificador 'SY\_ANAL', el cargo 'System Analyst' y un salario mínimo de 6000.

Ejecute el script `/home/oracle/labs/plpu/solns/sol_ap_01_02_02.sql`. Cuando se le pida que seleccione una conexión, seleccione la nueva conexión **VideoCompany**. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
SET SERVEROUTPUT ON
EXECUTE new_job ('SY_ANAL', 'System Analyst', 6000)
```



- 3) Compruebe si se ha agregado una fila y anote el identificador del nuevo trabajo para utilizarlo en el siguiente ejercicio. Confirme los cambios.

Ejecute el script `/home/oracle/labs/plpu/solns/sol_ap_01_02_03.sql`. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
SELECT *
FROM jobs
WHERE job_id = 'SY_ANAL';
COMMIT;
```

**Soluciones a la Práctica 1-2: Adición de un Nuevo Trabajo a la Tabla JOBS (continuación)**

| JOB_ID  | JOB_TITLE      | MIN_SALARY | MAX_SALARY |
|---------|----------------|------------|------------|
| SY_ANAL | System Analyst | 6000       | 12000      |

1 rows selected  
COMMIT succeeded.

## **Soluciones a la Práctica 1-3: Adición de una Nueva Fila a la Tabla JOB\_HISTORY**

En esta práctica, agregará una nueva fila a la tabla JOB\_HISTORY para un empleado existente.

- 1) Cree un procedimiento almacenado denominado ADD\_JOB\_HIST para agregar una nueva fila a la tabla JOB\_HISTORY para un empleado que está cambiando su trabajo al nuevo identificador de trabajo ('SY\_ANAL') creado en el ejercicio 1 b.
  - a) El procedimiento debe proporcionar dos parámetros, uno para el identificador del empleado que está cambiando el trabajo y el otro para el identificador del nuevo trabajo.
  - b) Lea el identificador del empleado de la tabla EMPLOYEES e insértelo en la tabla JOB\_HISTORY.
  - c) Indique la fecha de contratación de este empleado como fecha de inicio y la fecha de hoy como fecha final para esta fila en la tabla JOB\_HISTORY.
  - d) Cambie la fecha de contratación de este empleado en la tabla EMPLOYEES a la fecha de hoy.
  - e) Actualice el identificador de trabajo de este empleado al identificador de trabajo transferido como parámetro (utilice el identificador de trabajo 'SY\_ANAL') y el salario igual al salario mínimo para ese identificador de trabajo + 500.

**Nota:** incluya el manejo de excepciones para manejar el intento de insertar un empleado no existente.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_03\_01.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```

CREATE OR REPLACE PROCEDURE add_job_hist(
 p_emp_id IN employees.employee_id%TYPE,
 p_new_jobid IN jobs.job_id%TYPE) IS
BEGIN
 INSERT INTO job_history
 SELECT employee_id, hire_date, SYSDATE, job_id,
 department_id
 FROM employees
 WHERE employee_id = p_emp_id;
 UPDATE employees
 SET hire_date = SYSDATE,
 job_id = p_new_jobid,
 salary = (SELECT min_salary + 500
 FROM jobs
 WHERE job_id = p_new_jobid)
 WHERE employee_id = p_emp_id;
 DBMS_OUTPUT.PUT_LINE ('Added employee ' || p_emp_id ||
 ' details to the JOB_HISTORY
table');

```

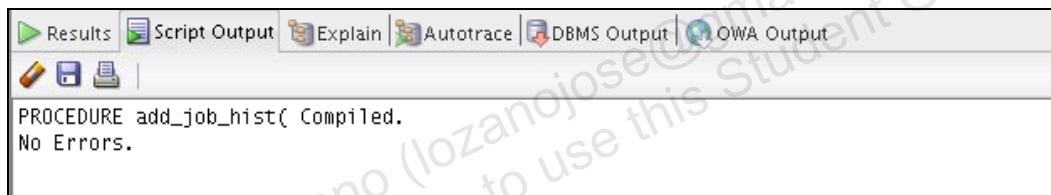
## Soluciones a la Práctica 1-3: Adición de una Nueva Fila a la Tabla JOB\_HISTORY (continuación)

```

DBMS_OUTPUT.PUT_LINE ('Updated current job of employee '
|| p_emp_id || ' to ' || p_new_jobid);

EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RAISE_APPLICATION_ERROR (-20001, 'Employee does not
exist!');
END add_job_hist;
/
SHOW ERRORS

```



- 2) Desactive todos los disparadores de las tablas EMPLOYEES, JOBS y JOB\_HISTORY antes de llamar al procedimiento ADD\_JOB\_HIST.

Ejecute el script `/home/oracle/labs/plpu/sols/sol_ap_01_03_02.sql`. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```

ALTER TABLE employees DISABLE ALL TRIGGERS;
ALTER TABLE jobs DISABLE ALL TRIGGERS;
ALTER TABLE job_history DISABLE ALL TRIGGERS;

```



## Soluciones a la Práctica 1-3: Adición de una Nueva Fila a la Tabla JOB\_HISTORY (continuación)

```

Results Script Output Explain Autotrace DBMS Output OWA Output
ALTER TABLE employees succeeded.
ALTER TABLE jobs succeeded.
ALTER TABLE job_history succeeded.

```

- 3) Active SERVEROUTPUT y, a continuación, ejecute el procedimiento con el identificador de empleado 106 y el identificador de trabajo 'SY\_ANAL' como parámetros.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_03\_03.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```

SET SERVEROUTPUT ON

EXECUTE add_job_hist(106, 'SY_ANAL')

```



```

Results Script Output Explain Autotrace DBMS Output OWA Output
anonymous block completed
Added employee 106 details to the JOB_HISTORY table
Updated current job of employee 106 to SY_ANAL

```

- 4) Consulte las tablas JOB\_HISTORY y EMPLOYEES para ver los cambios del empleado 106 y, a continuación, confirme los cambios.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_03\_04.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```

SELECT * FROM job_history
WHERE employee_id = 106;

SELECT job_id, salary FROM employees
WHERE employee_id = 106;

COMMIT;

```

## Soluciones a la Práctica 1-3: Adición de una Nueva Fila a la Tabla JOB\_HISTORY (continuación)

The screenshot shows the Oracle SQL Developer interface with the 'Results' tab selected. The output displays the following data:

| EMPLOYEE_ID | START_DATE | END_DATE  | JOB_ID  | DEPARTMENT_ID |
|-------------|------------|-----------|---------|---------------|
| 106         | 05-FEB-98  | 26-JUN-09 | IT_PROG | 60            |

1 rows selected.

| JOB_ID  | SALARY |
|---------|--------|
| SY_ANAL | 6500   |

1 rows selected.

COMMIT succeeded.

- 5) Vuelva a activar los disparadores en las tablas EMPLOYEES, JOBS y JOB\_HISTORY.

Ejecute el script `/home/oracle/labs/plpu/solns/sol_ap_01_03_05.sql`. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
ALTER TABLE employees ENABLE ALL TRIGGERS;
ALTER TABLE jobs ENABLE ALL TRIGGERS;
ALTER TABLE job_history ENABLE ALL TRIGGERS;
```



The screenshot shows the Oracle SQL Developer interface with the 'Results' tab selected. The output displays the following messages:

```
ALTER TABLE employees succeeded.
ALTER TABLE jobs succeeded.
ALTER TABLE job_history succeeded.
```

## Solución a la Práctica 1-4: Actualización de los Salarios Mínimos y Máximos de un Trabajo

En esta práctica, creará un programa para actualizar los salarios mínimos y máximos de un trabajo de la tabla JOBS.

- 1) Cree un procedimiento almacenado denominado UPD\_JOBSAL para actualizar los salarios mínimos y máximos de un identificador de trabajo concreto de la tabla JOBS. El procedimiento debe proporcionar tres parámetros: el identificador de trabajo, un nuevo salario mínimo y un nuevo salario máximo. Agregue el manejo de excepciones para justificar un identificador de trabajo no válido en la tabla JOBS. Emite una excepción si el salario máximo proporcionado es menor que el salario mínimo, y proporcione un mensaje que se mostrará si la fila de la tabla JOBS está bloqueada.  
**Indicación:** el número de error de recurso bloqueado/ocupado es -54.

Ejecute el script `/home/oracle/labs/plpu/solns/sol_ap_01_04_01.sql`. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

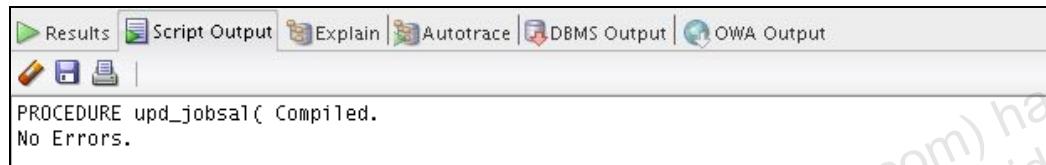
```

CREATE OR REPLACE PROCEDURE upd_jobsal(
 p_jobid IN jobs.job_id%type,
 p_new_minsal IN jobs.min_salary%type,
 p_new_maxsal IN jobs.max_salary%type) IS
 v_dummy PLS_INTEGER;
 e_resource_busy EXCEPTION;
 e_sal_error EXCEPTION;
 PRAGMA EXCEPTION_INIT (e_resource_busy , -54);
BEGIN
 IF (p_new_maxsal < p_new_minsal) THEN
 RAISE e_sal_error;
 END IF;
 SELECT 1 INTO v_dummy
 FROM jobs
 WHERE job_id = p_jobid
 FOR UPDATE OF min_salary NOWAIT;
 UPDATE jobs
 SET min_salary = p_new_minsal,
 max_salary = p_new_maxsal
 WHERE job_id = p_jobid;
EXCEPTION
 WHEN e_resource_busy THEN
 RAISE_APPLICATION_ERROR (-20001,
 'Job information is currently locked, try later.');
 WHEN NO_DATA_FOUND THEN
 RAISE_APPLICATION_ERROR(-20001, 'This job ID does not
exist');
 WHEN e_sal_error THEN
 RAISE_APPLICATION_ERROR(-20001,
 'The new maximum salary must be greater than or equal to
the new minimum salary');
END;

```

## Solución a la Práctica 1-4: Actualización de los Salarios Mínimos y Máximos de un Trabajo (continuación)

```
'Data error: Max salary should be more than min salary');
END upd_jobsal;
/
SHOW ERRORS
```



- 2) Active SERVEROUTPUT y, a continuación, ejecute el procedimiento UPD\_JOBSAL mediante un identificador de trabajo 'SY\_ANAL', un salario mínimo de 7000 y un salario máximo de 140.

**Nota:** esto debe generar un mensaje de excepción.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_04\_02.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
SET SERVEROUTPUT ON
EXECUTE upd_jobsal('SY_ANAL', 7000, 140)
```



## **Solución a la Práctica 1-4: Actualización de los Salarios Mínimos y Máximos de un Trabajo (continuación)**

```

Line 1: SQLPLUS Command Skipped: SET SEREVEROUTPUT ON

Error starting at line 3 in command:
EXECUTE upd_jobsal('SY_ANAL', 7000, 140)
Error report:
ORA-20001: Data error: Max salary should be more than min salary
ORA-06512: at "ORA62.UPD_JOBSAL", line 28
ORA-06512: at line 1

```

- 3) Desactive los disparadores de las tablas EMPLOYEES y JOBS.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_04\_03.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```

ALTER TABLE employees DISABLE ALL TRIGGERS;
ALTER TABLE jobs DISABLE ALL TRIGGERS;

```



```

ALTER TABLE employees succeeded.
ALTER TABLE jobs succeeded.

```

- 4) Ejecute el procedimiento UPD\_JOBSAL utilizando un identificador de trabajo 'SY\_ANAL', un salario mínimo de 7000 y un salario máximo de 14000.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_04\_04.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```

EXECUTE upd_jobsal('SY_ANAL', 7000, 14000)

```

## Solución a la Práctica 1-4: Actualización de los Salarios Mínimos y Máximos de un Trabajo (continuación)

The screenshot shows the Oracle SQL Developer interface. At the top, there is a 'Select Connection' dialog box with the title 'Select Connection'. It contains a dropdown menu labeled 'Connection: VideoCompany' and a toolbar with a green plus sign and a pencil icon. Below the dialog are two windows: one titled 'Results' showing the query 'anonymous block completed' and another titled 'Script Output'.

- 5) Consulte la tabla JOBS para ver los cambios y, a continuación, confirmelos.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_04\_05.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

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

The screenshot shows the Oracle SQL Developer interface. At the top, there is a 'Select Connection' dialog box with the title 'Select Connection'. It contains a dropdown menu labeled 'Connection: VideoCompany' and a toolbar with a green plus sign and a pencil icon. Below the dialog is a results window showing the output of the query:

| JOB_ID  | JOB_TITLE      | MIN_SALARY | MAX_SALARY |
|---------|----------------|------------|------------|
| SY_ANAL | System Analyst | 7000       | 14000      |

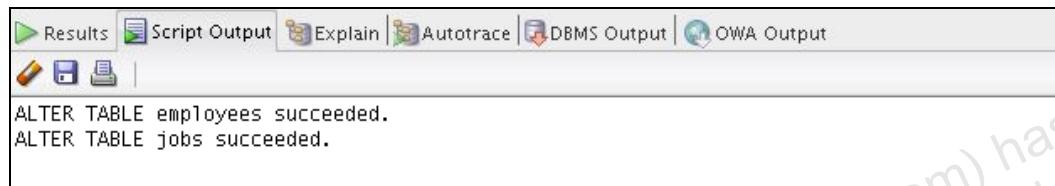
1 rows selected

- 6) Active los disparadores de las tablas EMPLOYEES y JOBS.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_04\_06.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

**Solución a la Práctica 1-4: Actualización de los Salarios Mínimos y Máximos de un Trabajo (continuación)**

```
ALTER TABLE employees ENABLE ALL TRIGGERS;
ALTER TABLE jobs ENABLE ALL TRIGGERS;
```



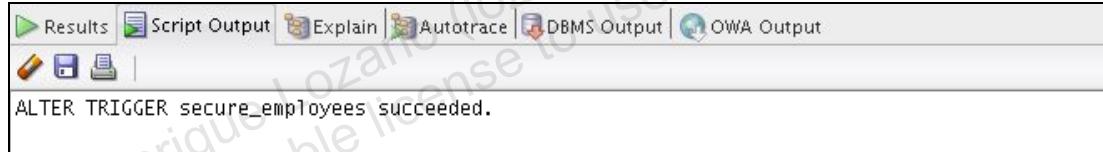
## Soluciones a la Práctica 1-5: Supervisión de los Salarios de los Empleados

En esta práctica, creará un procedimiento para supervisar si los salarios de los empleados han excedido la media para el tipo de trabajo.

- 1) Desactive el disparador SECURE\_EMPLOYEES.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_05\_01.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
ALTER TRIGGER secure_employees DISABLE;
```



- 2) En la tabla EMPLOYEES, agregue una columna EXCEED\_AVGSAL para almacenar hasta tres caracteres y un valor por defecto NO. Utilice una restricción de control para permitir los valores YES o NO.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_05\_02.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
ALTER TABLE employees (
 ADD (exceed_avgsal VARCHAR2(3) DEFAULT 'NO'
 CONSTRAINT employees_exceed_avgsal_ck
 CHECK (exceed_avgsal IN ('YES', 'NO')));
```

## Soluciones a la Práctica 1-5: Supervisión de los Salarios de los Empleados (continuación)

The screenshot shows two windows from Oracle SQL Developer. The top window is titled "Select Connection" with the sub-instruction "Select the connection you wish to use from the list, or create a new connection." It contains a dropdown menu set to "VideoCompany" and three buttons: "Help", "OK", and "Cancel". Below this is a results window with tabs for "Results", "Script Output", "Explain", "Autotrace", "DBMS Output", and "OWA Output". The "Results" tab is active and displays the message "ALTER TABLE employees succeeded.".

- 3) Cree un procedimiento almacenado denominado `CHECK_AVGSAL` que compruebe si el salario de cada empleado excede el salario medio para `JOB_ID`.
- El salario medio de un trabajo se calcula a partir de la información de la tabla `JOBS`.
  - Si el salario del empleado excede la media para el trabajo, actualice la columna `EXCEED_AVGSAL` de la tabla `EMPLOYEES` con el valor `YES`; de lo contrario, defina el valor en `NO`.
  - Utilice un cursor para seleccionar las filas del empleado con la opción `FOR UPDATE` de la consulta.
  - Agregue el manejo de excepciones para justificar el bloqueo de un registro.  
**Indicación:** el número de error de recurso bloqueado/ocupado es `-54`.
  - Escriba y utilice una función local denominada `GET_JOB_AVGSAL` para determinar el salario medio de un identificador de trabajo especificado como parámetro.

**Ejecute el script `/home/oracle/labs/plpu/solns/sol_ap_01_05_03.sql`. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:**

```
CREATE OR REPLACE PROCEDURE check_avgsal IS
 emp_exceed_avgsal_type employees.exceed_avgsal%type;
 CURSOR c_emp_csr IS
 SELECT employee_id, job_id, salary
 FROM employees
 FOR UPDATE;
 e_resource_busy EXCEPTION;
 PRAGMA EXCEPTION_INIT(e_resource_busy, -54);
 FUNCTION get_job_avgsal (jobid VARCHAR2) RETURN NUMBER IS
 avg_sal employees.salary%type;
 BEGIN
```

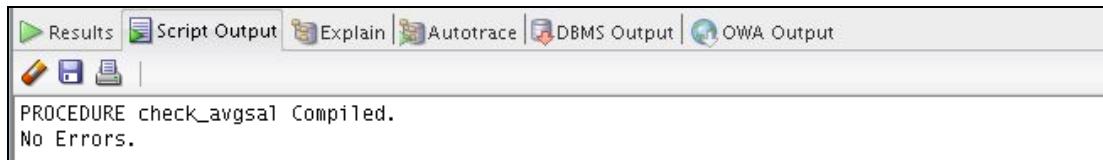
## Soluciones a la Práctica 1-5: Supervisión de los Salarios de los Empleados (continuación)

```

SELECT (max_salary + min_salary)/2 INTO avg_sal
FROM jobs
WHERE job_id = jobid;
RETURN avg_sal;
END;

BEGIN
FOR emprec IN c_emp_csr
LOOP
 emp_exceed_avgsal_type := 'NO';
 IF emprec.salary >= get_job_avgsal(emprec.job_id) THEN
 emp_exceed_avgsal_type := 'YES';
 END IF;
 UPDATE employees
 SET exceed_avgsal = emp_exceed_avgsal_type
 WHERE CURRENT OF c_emp_csr;
END LOOP;
EXCEPTION
 WHEN e_resource_busy THEN
 ROLLBACK;
 RAISE_APPLICATION_ERROR (-20001, 'Record is busy, try
later.');
END check_avgsal;
/
SHOW ERRORS

```



- 4) Ejecute el procedimiento CHECK\_AVGSAL. Para ver los resultados de las modificaciones, escriba una consulta para mostrar el identificador del empleado, el trabajo, el salario medio del trabajo, el salario del empleado y la columna del indicador exceed\_avgsal para empleados cuyos salarios exceden la media para el trabajo. Por último, confirme los cambios.

**Nota:** estos ejercicios se pueden utilizar como práctica adicional al describir cómo crear funciones.

## Soluciones a la Práctica 1-5: Supervisión de los Salarios de los Empleados (continuación)

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_05\_04.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
EXECUTE check_avgsal

SELECT e.employee_id, e.job_id, (j.max_salary-j.min_salary/2)
job_avgsal,
 e.salary, e.exceed_avgsal avg_exceeded
FROM employees e, jobs j
WHERE e.job_id = j.job_id
and e.exceed_avgsal = 'YES';

COMMIT;
```



| anonymous block completed |            |            |        |              |
|---------------------------|------------|------------|--------|--------------|
| EMPLOYEE_ID               | JOB_ID     | JOB_AVGSAL | SALARY | AVG_EXCEEDED |
| 103                       | IT_PROG    | 8000       | 9000   | YES          |
| 109                       | FI_ACCOUNT | 6900       | 9000   | YES          |
| 110                       | FI_ACCOUNT | 6900       | 8200   | YES          |
| 111                       | FI_ACCOUNT | 6900       | 7700   | YES          |
| 112                       | FI_ACCOUNT | 6900       | 7800   | YES          |
| 113                       | FI_ACCOUNT | 6900       | 6900   | YES          |
| 120                       | ST_MAN     | 5750       | 8000   | YES          |
| 121                       | ST_MAN     | 5750       | 8200   | YES          |
| 122                       | ST_MAN     | 5750       | 7900   | YES          |
| 137                       | ST_CLERK   | 4000       | 3600   | YES          |
| 141                       | ST_CLERK   | 4000       | 3500   | YES          |
| 150                       | SA REP     | 9000       | 10000  | YES          |
| 151                       | SA REP     | 9000       | 9500   | YES          |
| 152                       | SA REP     | 9000       | 9000   | YES          |
| 156                       | SA REP     | 9000       | 10000  | YES          |
| 157                       | SA REP     | 9000       | 9500   | YES          |
| 158                       | SA REP     | 9000       | 9000   | YES          |
| 162                       | SA REP     | 9000       | 10500  | YES          |
| 163                       | SA REP     | 9000       | 9500   | YES          |
| 168                       | SA REP     | 9000       | 11500  | YES          |
| 169                       | SA REP     | 9000       | 10000  | YES          |
| 170                       | SA REP     | 9000       | 9600   | YES          |
| 174                       | SA REP     | 9000       | 11000  | YES          |
| 184                       | SH_CLERK   | 4250       | 4200   | YES          |
| 185                       | SH_CLERK   | 4250       | 4100   | YES          |

**Soluciones a la Práctica 1-5: Supervisión de los Salarios de los Empleados (continuación)**

|     |            |       |       |     |
|-----|------------|-------|-------|-----|
| 192 | SH_CLERK   | 4250  | 4000  | YES |
| 201 | MK_MAN     | 10500 | 13000 | YES |
| 203 | HR REP     | 7000  | 6500  | YES |
| 204 | PR REP     | 8250  | 10000 | YES |
| 206 | AC ACCOUNT | 6900  | 8300  | YES |

30 rows selected  
COMMIT succeeded.

## Soluciones a la Práctica 1-6: Recuperación del Número Total de Años de Servicio de un Empleado

En esta práctica, creará un subprograma para recuperar el número de años de servicio de un empleado concreto.

- 1) Cree una función almacenada denominada GET\_YEARS\_SERVICE para recuperar el número total de años de servicio de un empleado concreto. La función debe aceptar el identificador de empleado como parámetro y devolver el número de años de servicio. Agregue la gestión de errores para justificar un identificador de empleado no válido.

**Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_06\_01.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:**

```

CREATE OR REPLACE FUNCTION get_years_service(
 p_emp.empid_type IN employees.employee_id%TYPE) RETURN
NUMBER IS
 CURSOR c_jobh_csr IS
 SELECT MONTHS_BETWEEN(end_date, start_date)/12
v_years_in_job
 FROM job_history
 WHERE employee_id = p_emp.empid_type;
 v_years_service NUMBER(2) := 0;
 v_years_in_job NUMBER(2) := 0;
BEGIN
 FOR jobh_rec IN c_jobh_csr
 LOOP
 EXIT WHEN c_jobh_csr%NOTFOUND;
 v_years_service := v_years_service +
jobh_rec.v_years_in_job;
 END LOOP;
 SELECT MONTHS_BETWEEN(SYSDATE, hire_date)/12 INTO
v_years_in_job
 FROM employees
 WHERE employee_id = p_emp.empid_type;
 v_years_service := v_years_service + v_years_in_job;
 RETURN ROUND(v_years_service);
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RAISE_APPLICATION_ERROR(-20348,
 'Employee with ID '|| p_emp.empid_type ||' does not
exist.');
 RETURN NULL;
END get_years_service;
/
SHOW ERRORS

```

## Soluciones a la Práctica 1-6: Recuperación del Número Total de Años de Servicio de un Empleado (continuación)

The screenshot shows the Oracle SQL Developer interface. At the top, there is a 'Select Connection' dialog box with the title 'Select Connection'. It contains the instruction 'Select the connection you wish to use from the list, or create a new connection.' Below this is a dropdown menu labeled 'Connection:' with 'VideoCompany' selected. To the right of the dropdown are two buttons: a green '+' button and a pencil icon. At the bottom of the dialog are three buttons: 'Help', 'OK', and 'Cancel'. Below the dialog is the main SQL developer window. The toolbar at the top of this window includes tabs for 'Results', 'Script Output', 'Explain', 'Autotrace', 'DBMS Output', and 'OWA Output'. There are also icons for 'New Query', 'Save', and 'Print'. The main area displays the following text:

```
FUNCTION get_years_service() Compiled.
No Errors.
```

- 2) Llame a la función GET\_YEARS\_SERVICE en una llamada a DBMS\_OUTPUT.PUT\_LINE para un empleado que tenga el identificador 999.  
**Ejecute el script**  
*/home/oracle/labs/plpu/solns/sol\_ap\_01\_06\_02.sql*. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
EXECUTE DBMS_OUTPUT.PUT_LINE(get_years_service(999))
```

The screenshot shows the Oracle SQL Developer interface. At the top, there is a 'Select Connection' dialog box with the title 'Select Connection'. It contains the instruction 'Select the connection you wish to use from the list, or create a new connection.' Below this is a dropdown menu labeled 'Connection:' with 'VideoCompany' selected. To the right of the dropdown are two buttons: a green '+' button and a pencil icon. At the bottom of the dialog are three buttons: 'Help', 'OK', and 'Cancel'. Below the dialog is the main SQL developer window. The toolbar at the top of this window includes tabs for 'Results', 'Script Output', 'Explain', 'Autotrace', 'DBMS Output', and 'OWA Output'. There are also icons for 'New Query', 'Save', and 'Print'. The main area displays the following text:

```
Error starting at line 3 in command:
EXECUTE DBMS_OUTPUT.PUT_LINE(get_years_service(999))
Error report:
ORA-20348: Employee with ID 999 does not exist.
ORA-06512: at "ORA62.GET_YEARS_SERVICE", line 22
ORA-06512: at line 1
```

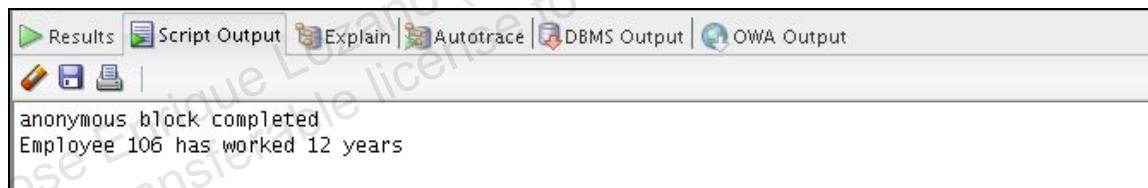
## **Soluciones a la Práctica 1-6: Recuperación del Número Total de Años de Servicio de un Empleado (continuación)**

- 3) Muestre el número de años de servicio para el empleado 106 mediante la llamada de DBMS\_OUTPUT.PUT\_LINE a la función GET\_YEARS\_SERVICE. Asegúrese de activar SERVEROUTPUT.

**Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_06\_03.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:**

```
SET SERVEROUTPUT ON

BEGIN
 DBMS_OUTPUT.PUT_LINE (
 'Employee 106 has worked ' || get_years_service(106) || '
years');
END;
/
```



- 4) Consulte los datos del empleado concreto en las tablas JOB\_HISTORY y EMPLOYEES para verificar que las modificaciones son precisas. Los valores representados en los resultados de esta página pueden variar de los valores obtenidos al ejecutar las consultas.

**Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_06\_04.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:**

```
SELECT employee_id, job_id,
 MONTHS_BETWEEN(end_date, start_date)/12 duration
 FROM job_history;

SELECT job_id, MONTHS_BETWEEN(SYSDATE, hire_date)/12 duration
 FROM employees
 WHERE employee_id = 106;
```

**Soluciones a la Práctica 1-6: Recuperación del Número Total de Años de Servicio de un Empleado (continuación)**

Results Script Output Explain Autotrace DBMS Output OWA Output

| EMPLOYEE_ID | JOB_ID     | DURATION                                   |
|-------------|------------|--------------------------------------------|
| 102         | IT_PROG    | 5.52956989247311827956989247311827956989   |
| 101         | AC_ACCOUNT | 4.09946236559139784946236559139784946237   |
| 101         | AC_MGR     | 3.38172043010752688172043010752688172043   |
| 201         | MK_REP     | 3.83870967741935483870967741935483870968   |
| 114         | ST_CLERK   | 1.7688172043010752688172043010752688172    |
| 122         | ST_CLERK   | 0.9973118279569892473118279569892473118283 |
| 200         | AD_ASST    | 5.75                                       |
| 176         | SA REP     | 0.7688172043010752688172043010752688172042 |
| 176         | SA MAN     | 0.9973118279569892473118279569892473118283 |
| 200         | AC_ACCOUNT | 4.49731182795698924731182795698924731183   |
| 106         | IT_PROG    | 11.53968678439864595778574273197929111908  |

11 rows selected

| JOB_ID  | DURATION |
|---------|----------|
| SY_ANAL | 0        |

1 rows selected

## Soluciones a la Práctica 1-7: Recuperación del Número Total de Trabajos Diferentes de un Empleado

En esta práctica, creará un programa para recuperar el número de trabajos diferentes que ha tenido un empleado durante su servicio.

- 1) Cree una función almacenada denominada `GET_JOB_COUNT` para recuperar el número total de trabajos diferentes que ha tenido un empleado.
  - a) La función debe aceptar el identificador de empleado en un parámetro y devolver el número de trabajos diferentes que ha tenido un empleado hasta la fecha, incluido el presente.
  - b) Agregue el manejo de excepciones para justificar un identificador de empleado no válido.
  - c) Una dos consultas con `UNION` y cuente las filas recuperadas en una tabla PL/SQL.
  - d) Utilice `FETCH` con `BULK COLLECT INTO` para obtener los trabajos únicos del empleado.

**Ejecute el script `/home/oracle/labs/plpu/solns/sol_ap_01_07_01.sql`. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:**

```

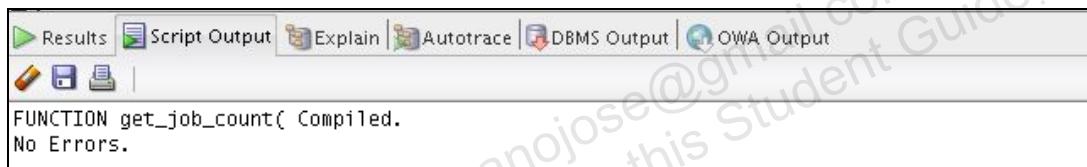
CREATE OR REPLACE FUNCTION get_job_count(
 p_emp.empid_type IN employees.employee_id%TYPE) RETURN
NUMBER IS
 TYPE jobs_table_type IS TABLE OF jobs.job_id%type;
 v_jobtab jobs_table_type;
 CURSOR c_empjob_csr IS
 SELECT job_id
 FROM job_history
 WHERE employee_id = p_emp.empid_type
 UNION
 SELECT job_id
 FROM employees
 WHERE employee_id = p_emp.empid_type;
BEGIN
 OPEN c_empjob_csr;
 FETCH c_empjob_csr BULK COLLECT INTO v_jobtab;
 CLOSE c_empjob_csr;
 RETURN v_jobtab.count;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RAISE_APPLICATION_ERROR(-20348,

```

## Soluciones a la Práctica 1-7: Recuperación del Número Total de Trabajos Diferentes de un Empleado (continuación)

```
'Employee with ID '|| p_emp.empid_type ||' does not
exist!');
 RETURN NULL;
END get_job_count;
/
SHOW ERRORS

FUNCTION get_job_count(Compiled.
No Errors.
```



- 2) Llame a la función para el empleado con el identificador 176. Asegúrese de activar SERVEROUTPUT.

**Nota:** estos ejercicios se pueden utilizar como práctica adicional al describir cómo crear paquetes.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_07\_02.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
SET SERVEROUTPUT ON

BEGIN
 DBMS_OUTPUT.PUT_LINE('Employee 176 worked on ' ||
 get_job_count(176) || ' different jobs.');
END;
/
```

**Soluciones a la Práctica 1-7: Recuperación del Número Total de Trabajos Diferentes de un Empleado (continuación)**A screenshot of the Oracle SQL Developer interface. At the top, there is a toolbar with several icons: "Results" (highlighted), "Script Output", "Explain", "Autotrace", "DBMS Output", and "OWA Output". Below the toolbar, there are two rows of small icons. The first row includes a pencil, a clipboard, a printer, and a vertical line. The second row includes a magnifying glass, a folder, and a vertical line. The main area of the interface displays the output of an anonymous block. The output shows the message "anonymous block completed" followed by "Employee 176 worked on 2 different jobs.".

## Soluciones a la Práctica 1-8: Creación de un Nuevo Paquete que Contenga los Procedimientos y las Funciones Recién Creados

En esta práctica, creará un paquete denominado EMPJOB\_PKG, que contiene los procedimientos NEW\_JOB, ADD\_JOB\_HIST y UPD\_JOBSAL, así como las funciones GET\_YEARS\_SERVICE y GET\_JOB\_COUNT.

- 1) Cree la especificación del paquete con toda la construcción del subprograma pública. Mueva cualquier tipo definido localmente del subprograma a la especificación del paquete.

**Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_08\_01.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:**

```
CREATE OR REPLACE PACKAGE empjob_pkg IS
 TYPE jobs_table_type IS TABLE OF jobs.job_id%TYPE;

 PROCEDURE add_job_hist(
 p_emp_id IN employees.employee_id%TYPE,
 p_new_jobid IN jobs.job_id%TYPE);

 FUNCTION get_job_count(
 p_emp_id IN employees.employee_id%TYPE) RETURN NUMBER;

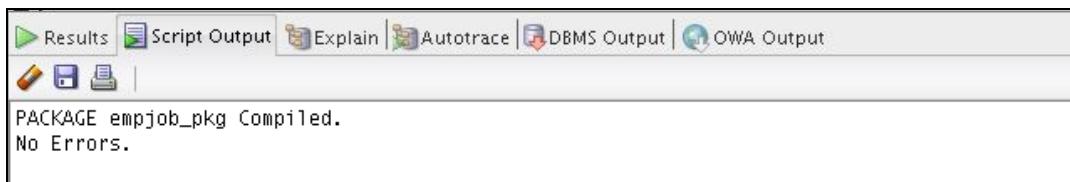
 FUNCTION get_years_service(
 p_emp_id IN employees.employee_id%TYPE) RETURN NUMBER;

 PROCEDURE new_job(
 p_jobid IN jobs.job_id%TYPE,
 p_title IN jobs.job_title%TYPE,
 p_minsal IN jobs.min_salary%TYPE);

 PROCEDURE upd_jobsal(
 p_jobid IN jobs.job_id%TYPE,
 p_new_minsal IN jobs.min_salary%TYPE,
 p_new_maxsal IN jobs.max_salary%TYPE);
END empjob_pkg;
/
SHOW ERRORS
```



## **Soluciones a la Práctica 1-8: Creación de un Nuevo Paquete que Contenga los Procedimientos y las Funciones Recién Creados (continuación)**



The screenshot shows the Oracle SQL Developer interface with the 'Script Output' tab selected. The output window displays the message: 'PACKAGE empjob\_pkg Compiled. No Errors.'.

- 2) Cree el cuerpo del paquete con la implantación del subprograma; recuerde eliminar de las implantaciones del subprograma cualquier tipo que haya movido a la especificación del paquete.

**Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_08\_02.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:**

```

CREATE OR REPLACE PACKAGE BODY empjob_pkg IS
 PROCEDURE add_job_hist(
 p_emp_id IN employees.employee_id%TYPE,
 p_new_jobid IN jobs.job_id%TYPE) IS
 BEGIN
 INSERT INTO job_history
 SELECT employee_id, hire_date, SYSDATE, job_id,
department_id
 FROM employees
 WHERE employee_id = p_emp_id;
 UPDATE employees
 SET hire_date = SYSDATE,
 job_id = p_new_jobid,
 salary = (SELECT min_salary + 500
 FROM jobs
 WHERE job_id = p_new_jobid)
 WHERE employee_id = p_emp_id;
 DBMS_OUTPUT.PUT_LINE ('Added employee ' || p_emp_id ||
 ' details to the JOB_HISTORY table');
 DBMS_OUTPUT.PUT_LINE ('Updated current job of employee ' ||
 p_emp_id|| ' to '|| p_new_jobid);
 EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RAISE_APPLICATION_ERROR (-20001, 'Employee does not
exist!');
 END add_job_hist;

 FUNCTION get_job_count(
 p_emp_id IN employees.employee_id%TYPE) RETURN NUMBER IS
 v_jobtab jobs_table_type;
 CURSOR c_empjob_csr IS
 SELECT job_id
 FROM job_history

```

## **Soluciones a la Práctica 1-8: Creación de un Nuevo Paquete que Contenga los Procedimientos y las Funciones Recién Creados (continuación)**

```

 WHERE employee_id = p_emp_id
 UNION
 SELECT job_id
 FROM employees
 WHERE employee_id = p_emp_id;
BEGIN
 OPEN c.empjob_csr;
 FETCH c.empjob_csr BULK COLLECT INTO v_jobtab;
 CLOSE c.empjob_csr;
 RETURN v_jobtab.count;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RAISE_APPLICATION_ERROR(-20348,
 'Employee with ID '|| p_emp_id ||' does not exist!');
 RETURN 0;
END get_job_count;

FUNCTION get_years_service(
 p_emp_id IN employees.employee_id%TYPE) RETURN NUMBER IS
CURSOR c.jobh_csr IS
 SELECT MONTHS_BETWEEN(end_date, start_date)/12
v_years_in_job
 FROM job_history
 WHERE employee_id = p_emp_id;
v_years_service NUMBER(2) := 0;
v_years_in_job NUMBER(2) := 0;
BEGIN
 FOR jobh_rec IN c.jobh_csr
 LOOP
 EXIT WHEN c.jobh_csr%NOTFOUND;
 v_years_service := v_years_service +
jobh_rec.v_years_in_job;
 END LOOP;
 SELECT MONTHS_BETWEEN(SYSDATE, hire_date)/12 INTO
v_years_in_job
 FROM employees
 WHERE employee_id = p_emp_id;
 v_years_service := v_years_service + v_years_in_job;
 RETURN ROUND(v_years_service);
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RAISE_APPLICATION_ERROR(-20348,
 'Employee with ID '|| p_emp_id ||' does not exist.');
 RETURN 0;
END get_years_service;

PROCEDURE new_job(
 p_jobid IN jobs.job_id%TYPE,
 p_title IN jobs.job_title%TYPE,

```

## **Soluciones a la Práctica 1-8: Creación de un Nuevo Paquete que Contenga los Procedimientos y las Funciones Recién Creados (continuación)**

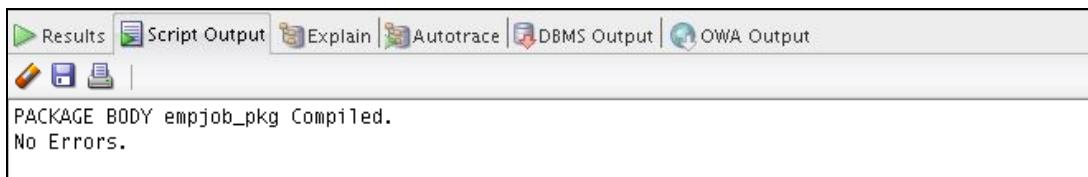
```

p_minsal IN jobs.min_salary%TYPE) IS
 v_maxsal jobs.max_salary%TYPE := 2 * p_minsal;
BEGIN
 INSERT INTO jobs(job_id, job_title, min_salary,
max_salary)
 VALUES (p_jobid, p_title, p_minsal, v_maxsal);
 DBMS_OUTPUT.PUT_LINE ('New row added to JOBS table:');
 DBMS_OUTPUT.PUT_LINE (p_jobid || ' ' || p_title || ' ' ||
 p_minsal || ' ' || v_maxsal);
END new_job;

PROCEDURE upd_jobsal(
 p_jobid IN jobs.job_id%type,
 p_new_minsal IN jobs.min_salary%type,
 p_new_maxsal IN jobs.max_salary%type) IS
 v_dummy PLS_INTEGER;
 e_resource_busy EXCEPTION;
 e_sal_error EXCEPTION;
 PRAGMA EXCEPTION_INIT (e_resource_busy , -54);
BEGIN
 IF (p_new_maxsal < p_new_minsal) THEN
 RAISE e_sal_error;
 END IF;
 SELECT 1 INTO v_dummy
 FROM jobs
 WHERE job_id = p_jobid
 FOR UPDATE OF min_salary NOWAIT;
 UPDATE jobs
 SET min_salary = p_new_minsal,
 max_salary = p_new_maxsal
 WHERE job_id = p_jobid;
EXCEPTION
 WHEN e_resource_busy THEN
 RAISE_APPLICATION_ERROR (-20001,
 'Job information is currently locked, try later.');
 WHEN NO_DATA_FOUND THEN
 RAISE_APPLICATION_ERROR(-20001, 'This job ID does not
exist');
 WHEN e_sal_error THEN
 RAISE_APPLICATION_ERROR(-20001,
 'Data error: Max salary should be more than min
salary');
END upd_jobsal;
END empjob_pkg;
/
SHOW ERRORS

```

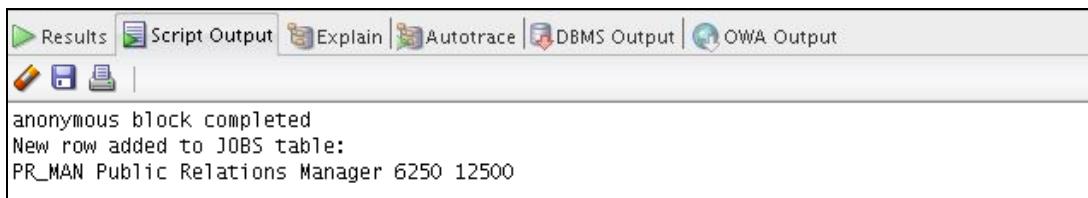
## Soluciones a la Práctica 1-8: Creación de un Nuevo Paquete que Contenga los Procedimientos y las Funciones Recién Creados (continuación)



- 3) Llame al procedimiento EMPJOB\_PKG.NEW\_JOB para crear un nuevo trabajo con el identificador PR\_MAN, el cargo Public Relations Manager y el salario 6250. Asegúrese de activar SERVEROUTPUT.

Ejecute el script /home/oracle/labs/plpu/sols/sol\_ap\_01\_08\_03.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
SET SERVEROUTPUT ON
EXECUTE empjob_pkg.new_job('PR_MAN', 'Public Relations
Manager', 6250)
```



## **Soluciones a la Práctica 1-8: Creación de un Nuevo Paquete que Contenga los Procedimientos y las Funciones Recién Creados (continuación)**

- 4) Llame al procedimiento EMPJOB\_PKG.ADD\_JOB\_HIST para modificar el trabajo del empleado con identificador 110 por el identificador de trabajo PR\_MAN.

**Nota:** debe desactivar el disparador UPDATE\_JOB\_HISTORY antes de ejecutar el procedimiento ADD\_JOB\_HIST y volver a activar el disparador después de ejecutar el procedimiento.

Ejecute el script **/home/oracle/labs/plpu/solns/sol\_ap\_01\_08\_04.sql**. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
ALTER TRIGGER update_job_history DISABLE;
EXECUTE empjob_pkg.add_job_hist(110, 'PR_MAN')
ALTER TRIGGER update_job_history ENABLE;
```



```
ALTER TRIGGER update_job_history succeeded.
anonymous block completed
Added employee 110 details to the JOB_HISTORY table
Updated current job of employee 110 to PR_MAN

ALTER TRIGGER update_job_history succeeded.
```

- 5) Consulte las tablas JOBS, JOB\_HISTORY y EMPLOYEES para verificar los resultados.

**Nota:** estos ejercicios se pueden utilizar como práctica adicional al describir cómo crear disparadores de base de datos.

Ejecute el script **/home/oracle/labs/plpu/solns/sol\_ap\_01\_08\_05.sql**. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
SELECT * FROM jobs WHERE job_id = 'PR_MAN';
SELECT * FROM job_history WHERE employee_id = 110;
SELECT job_id, salary FROM employees WHERE employee_id = 110;
```

## Soluciones a la Práctica 1-8: Creación de un Nuevo Paquete que Contenga los Procedimientos y las Funciones Recién Creados (continuación)



Results Script Output Explain Autotrace DBMS Output OWA Output

| JOB_ID          | JOB_TITLE                | MIN_SALARY | MAX_SALARY           |
|-----------------|--------------------------|------------|----------------------|
| PR_MAN          | Public Relations Manager | 6250       | 12500                |
| 1 rows selected |                          |            |                      |
| EMPLOYEE_ID     | START_DATE               | END_DATE   | JOB_ID DEPARTMENT_ID |
| 110             | 28-SEP-97                | 19-AUG-09  | FI_ACCOUNT 100       |
| 1 rows selected |                          |            |                      |
| JOB_ID          | SALARY                   |            |                      |
| PR_MAN          | 6750                     |            |                      |
| 1 rows selected |                          |            |                      |

## **Solución a la Práctica 1-9: Creación de un Disparador para Garantizar que los Salarios de los Empleados Estén en un Rango Aceptable**

En esta práctica, creará un disparador para asegurarse de que los salarios mínimos y máximos de un trabajo nunca se modifiquen, de forma que el salario de un empleado existente con ese identificador de trabajo esté fuera del nuevo rango especificado para el trabajo.

- 1) Cree un disparador denominado CHECK\_SAL\_RANGE que se dispare antes de cada fila que se actualice en las columnas MIN\_SALARY y MAX\_SALARY de la tabla JOBS.
  - a) Para cada valor de salario mínimo o máximo que se cambie, compruebe si el salario de cualquier empleado existente con ese identificador de trabajo de la tabla EMPLOYEES entra dentro del nuevo rango de salarios especificado para este identificador de trabajo.
  - b) Incluya el manejo de excepciones para cubrir un cambio de rango de salarios que afecte al registro de cualquier empleado existente.

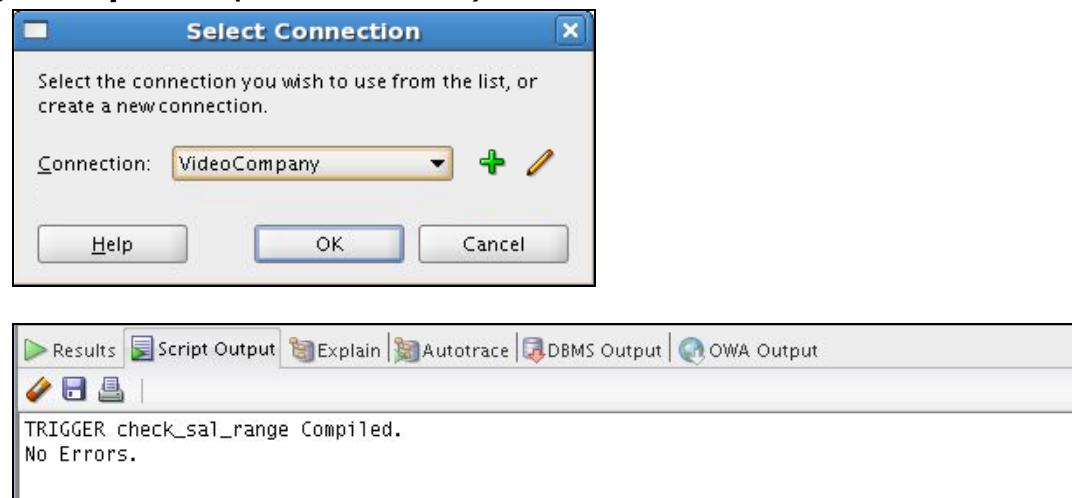
**Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_09\_01.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:**

```

CREATE OR REPLACE TRIGGER check_sal_range
BEFORE UPDATE OF min_salary, max_salary ON jobs
FOR EACH ROW
DECLARE
 v_minsal employees.salary%TYPE;
 v_maxsal employees.salary%TYPE;
 e_invalid_salrange EXCEPTION;
BEGIN
 SELECT MIN(salary), MAX(salary) INTO v_minsal, v_maxsal
 FROM employees
 WHERE job_id = :NEW.job_id;
 IF (v_minsal < :NEW.min_salary) OR (v_maxsal >
:NEW.max_salary) THEN
 RAISE e_invalid_salrange;
 END IF;
EXCEPTION
 WHEN e_invalid_salrange THEN
 RAISE_APPLICATION_ERROR(-20550,
 'Employees exist whose salary is out of the specified
 range. ||'
 'Therefore the specified salary range cannot be
 updated.');
 END check_sal_range;
/
SHOW ERRORS

```

**Solución a la Práctica 1-9: Creación de un Disparador para Garantizar que los Salarios de los Empleados Estén en un Rango Aceptable (continuación)**



- 2) Pruebe el disparador utilizando el trabajo SY\_ANAL y defina el nuevo salario mínimo en 5000 y el nuevo salario máximo en 7000. Antes de realizar los cambios, escriba una consulta para mostrar el rango de salarios actual para el identificador de trabajo SY\_ANAL y otra consulta para mostrar el identificador, apellido y salario del empleado para el mismo identificador de trabajo. Después de la actualización, consulte los cambios (si existen) realizados en la tabla JOBS para el identificador de trabajo especificado.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_09\_02.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

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

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

UPDATE jobs
SET min_salary = 5000, max_salary = 7000
WHERE job_id = 'SY_ANAL';

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



## Solución a la Práctica 1-9: Creación de un Disparador para Garantizar que los Salarios de los Empleados Estén en un Rango Aceptable (continuación)

The screenshot shows three separate SQL queries run in the 'Results' tab of Oracle SQL Developer:

```

 Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
 +-----+
 | JOB_ID JOB_TITLE MIN_SALARY MAX_SALARY |
 | :-----: :-----: :-----: :-----: |
 | SY_ANAL System Analyst 7000 14000 |
 | :-----: :-----: :-----: :-----: |
 | 1 rows selected |
 | EMPLOYEE_ID LAST_NAME SALARY |
 | :-----: :-----: :-----: |
 | 106 Pataballa 6500 |
 | :-----: :-----: :-----: |
 | 1 rows selected |
 | 1 rows updated |
 | JOB_ID JOB_TITLE MIN_SALARY MAX_SALARY |
 | :-----: :-----: :-----: :-----: |
 | SY_ANAL System Analyst 5000 7000 |
 | :-----: :-----: :-----: :-----: |
 | 1 rows selected |

```

- 3) Mediante el trabajo SY\_ANAL, defina el nuevo salario mínimo en 7000 y el nuevo salario máximo en 18000. Explique los resultados.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_01\_09\_03.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```

UPDATE jobs
SET min_salary = 7000, max_salary = 18000
WHERE job_id = 'SY_ANAL';

```



The screenshot shows the results of running the update script. An error message is displayed:

```

Error starting at line 1 in command:
UPDATE jobs
SET min_salary = 7000, max_salary = 18000
WHERE job_id = 'SY_ANAL'
Error report:
SQL Error: ORA-20550: Employees exist whose salary is out of the specified range. Therefore the specified salary range cannot be updated.
ORA-06512: at "ORA62.CHECK_SAL_RANGE", line 14
ORA-04088: error during execution of trigger 'ORA62.CHECK_SAL_RANGE'

```

### **Solución a la Práctica 1-9: Creación de un Disparador para Garantizar que los Salarios de los Empleados Estén en un Rango Aceptable (continuación)**

La actualización no consigue cambiar el rango de salarios debido a la funcionalidad que proporciona el disparador `CHECK_SAL_RANGE`, porque el empleado 106 con identificador de trabajo `SY_ANAL` tiene un salario de 6500, cifra inferior al salario mínimo para el nuevo rango de salarios especificado en la sentencia `UPDATE`.

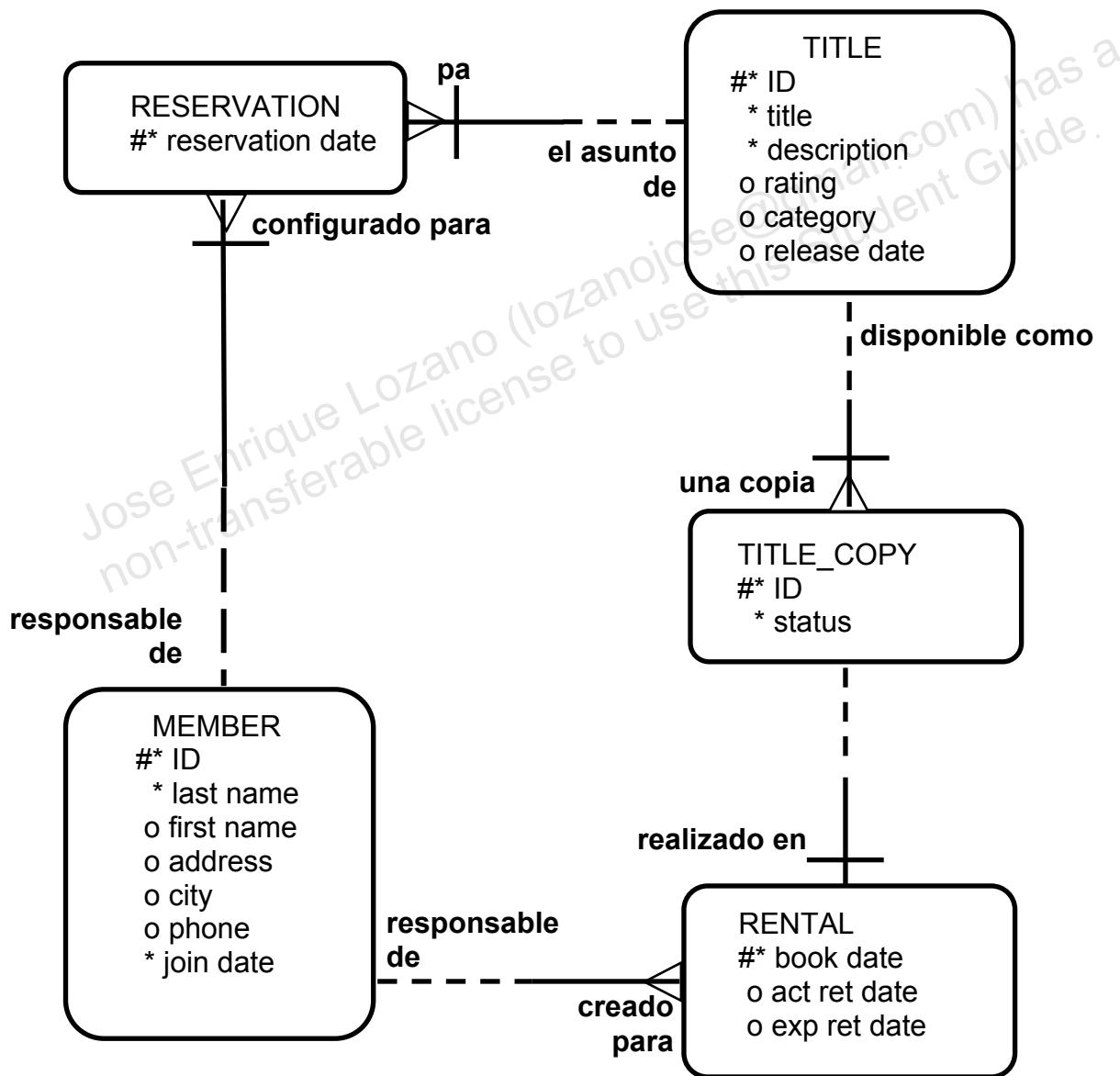
## Práctica 2

En este caso práctico, creará un paquete denominado VIDEO\_PKG que contenga los procedimientos y las funciones de una aplicación de un videoclub. Esta aplicación permite a los clientes convertirse en miembros del videoclub. Cualquier miembro podrá alquilar películas, devolver películas alquiladas y reservar películas. Además, creará un disparador para asegurarse de que los datos de las tablas de videos se modifican sólo durante las horas laborables.

Cree el paquete con SQL\*Plus y utilice el paquete DBMS\_OUTPUT proporcionado por Oracle para mostrar los mensajes.

La base de datos del videoclub contiene las siguientes tablas: TITLE, TITLE\_COPY, RENTAL, RESERVATION y MEMBER.

### Diagrama de entidad/relación de la base de datos de videoclub



## Práctica 2-1: Creación del Paquete VIDEO\_PKG

En esta práctica, creará un paquete denominado VIDEO\_PKG, que contenga los procedimientos y las funciones de una aplicación de un videoclub.

- 1) Cargue y ejecute el script /home/oracle/labs/plpu/labs/buildvid1.sql para crear todas las tablas y secuencias necesarias para este ejercicio.
- 2) Cargue y ejecute el script /home/oracle/labs/plpu/labs/buildvid2.sql para llenar todas las tablas creadas por el script buildvid1.sql.
- 3) Cree un paquete denominado VIDEO\_PKG con los siguientes procedimientos y funciones:
  - a) **NEW\_MEMBER:** procedimiento público que agrega un nuevo miembro a la tabla MEMBER. Para el número de identificador (ID) de miembro, utilice la secuencia MEMBER\_ID\_SEQ; para la fecha de unión, utilice SYSDATE. Transfiera los demás valores que deseé insertar en una nueva fila como parámetros.
  - b) **NEW\_RENTAL:** función pública sobrecargada para registrar un nuevo alquiler. Transfiera el número de identificador de título del video que el cliente desea alquilar, junto con el apellido del cliente o el número de identificador de miembro a la función. La función deberá devolver la fecha de vencimiento del video. Las fechas de vencimiento son tres días desde la fecha de alquiler del video. Si el estado de una película solicitada aparece como AVAILABLE en la tabla TITLE\_COPY para una copia de este título, actualice la tabla TITLE\_COPY y defina el estado en RENTED. Si no hay ninguna copia disponible, la función debe devolver el valor NULL. A continuación, inserte un nuevo registro en la tabla RENTAL, que identifique la fecha de reserva como la fecha de hoy, el número de identificador de copia, el número de identificador de título y la fecha de devolución esperada. Tenga en cuenta que puede que existan varios clientes con el mismo apellido. En este caso, haga que la función devuelva NULL y muestre una lista de todos los nombres de clientes que coincidan y sus números de identificador.
  - c) **RETURN\_MOVIE:** procedimiento público que actualiza el estado de un video (disponible, alquilado o dañado) y define la fecha de devolución. Transfiera el identificador de título, el identificador de copia y el estado a este procedimiento. Compruebe si existen reservas para ese título y muestre un mensaje, si está reservado. Actualice la tabla RENTAL y defina la fecha de devolución real en la fecha de hoy. Actualice el estado en la tabla TITLE\_COPY según el parámetro de estado transferido al procedimiento.
  - d) **RESERVE\_MOVIE:** procedimiento privado que se ejecuta sólo si todas las copias de videos solicitadas en el procedimiento NEW\_RENTAL tienen el estado RENTED. Transfiera el número de identificador de miembro y el número de identificador de título a este procedimiento. Inserte un nuevo registro en la tabla RESERVATION y registre la fecha de reserva, el número de identificador de miembro y el número de identificador de título. Imprima un mensaje que indique que una película está reservada y la fecha de devolución esperada.

## Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

- e) **EXCEPTION\_HANDLER:** procedimiento privado que se llama desde el manejador de excepciones de los programas públicos. Transfiera el número SQLCODE a este procedimiento y el nombre del programa (como cadena de texto) en el que se ha producido el error. Utilice RAISE\_APPLICATION\_ERROR para emitir un error personalizado. Empiece por una violación de clave única (-1) y una violación de clave ajena (-2292). Permita que el manejador de excepciones emita un error genérico para cualquier otro error.
- 4) Utilice los siguientes scripts situados en el directorio /home/oracle/labs/plpu/soln para probar las rutinas:
- Agregue dos miembros con sol\_ap\_02\_01\_04\_a.sql.
  - Agregue nuevos alquileres de videos con sol\_ap\_02\_01\_04\_b.sql.
  - Devuelva películas con el script sol\_ap\_02\_01\_04\_c.sql.
- 5) Las horas laborables para el videoclub son de 8:00 a.m. a 10:00 p.m. (de domingo a viernes) y de 8:00 a.m. a 12:00 p.m. (sábados). Para asegurarse de que las tablas sólo se puedan modificar durante estas horas, cree un procedimiento almacenado al que se llame con los disparadores de las tablas.
- Cree un procedimiento almacenado denominado TIME\_CHECK, que compruebe la hora actual con respecto a las horas laborables. Si la hora actual no está dentro de las horas laborables, utilice el procedimiento RAISE\_APPLICATION\_ERROR para emitir el mensaje adecuado.
  - Cree un disparador en cada una de las cinco tablas. Arranque el disparador antes de insertar, actualizar y suprimir los datos de las tablas. Llame al procedimiento TIME\_CHECK desde cada disparador.
  - Pruebe los disparadores.

**Nota:** para que el disparador falle, es posible que tenga que cambiar la hora para que esté fuera del rango de la hora actual de la clase. Por ejemplo, mientras realiza la prueba, puede que desee que las horas válidas de video del disparador sean desde las 18:00 p.m. a las 08:00 a.m.

## **Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG**

En esta práctica, creará un paquete denominado VIDEO\_PKG, que contenga los procedimientos y las funciones de una aplicación de un videoclub.

- 1) Cargue y ejecute el script /home/oracle/labs/plpu/labs/buildvid1.sql para crear todas las tablas y secuencias necesarias para este ejercicio.

**Ejecute el script /home/oracle/labs/plpu/labs/buildvid1.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:**

```

SET ECHO OFF
/* Script to build the Video Application (Part 1 -
buildvid1.sql)
 for the Oracle Introduction to Oracle with Procedure
Builder course.
 Created by: Debby Kramer Creation date: 12/10/95
 Last updated: 2/13/96
 Modified by Nagavalli Pataballa on 26-APR-2001
 For the course Introduction to Oracle9i: PL/SQL
 This part of the script creates tables and sequences that
are used
 by Part B of the Additional Practices of the course.
 Ignore the errors which appear due to dropping of table.
*/
DROP TABLE rental CASCADE CONSTRAINTS;
DROP TABLE reservation CASCADE CONSTRAINTS;
DROP TABLE title_copy CASCADE CONSTRAINTS;
DROP TABLE title CASCADE CONSTRAINTS;
DROP TABLE member CASCADE CONSTRAINTS;

PROMPT Please wait while tables are created.....

CREATE TABLE MEMBER
 (member_id NUMBER (10) CONSTRAINT member_id_pk
PRIMARY KEY
, last_name VARCHAR2(25)
 CONSTRAINT member_last_nn NOT NULL
, first_name VARCHAR2(25)
, address VARCHAR2(100)
, city VARCHAR2(30)
, phone VARCHAR2(25)
, join_date DATE DEFAULT SYSDATE
 CONSTRAINT join_date_nn NOT NULL)
/

CREATE TABLE TITLE
 (title_id NUMBER(10)
 CONSTRAINT title_id_pk PRIMARY KEY
, title VARCHAR2(60)

```

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

 CONSTRAINT title_nn NOT NULL
, description VARCHAR2(400)
 CONSTRAINT title_desc_nn NOT NULL
, rating VARCHAR2(4)
 CONSTRAINT title_rating_ck CHECK (rating IN
('G','PG','R','NC17','NR'))
, category VARCHAR2(20) DEFAULT 'DRAMA'
 CONSTRAINT title_categ_ck CHECK (category IN
('DRAMA','COMEDY','ACTION','CHILD','SCIFI','DOCUMENTARY'))
, release_date DATE)
/

CREATE TABLE TITLE_COPY
(copy_id NUMBER(10)
, title_id NUMBER(10)
 CONSTRAINT copy_title_id_fk
 REFERENCES title(title_id)
, status VARCHAR2(15)
 CONSTRAINT copy_status_nn NOT NULL
 CONSTRAINT copy_status_ck CHECK (status IN ('AVAILABLE',
'DESTROYED',
'RENTED', 'RESERVED'))
, CONSTRAINT copy_title_id_pk PRIMARY KEY(copy_id,
title_id))
/

CREATE TABLE RENTAL
(book_date DATE DEFAULT SYSDATE
, copy_id NUMBER(10)
, member_id NUMBER(10)
 CONSTRAINT rental_mbr_id_fk REFERENCES member(member_id)
, title_id NUMBER(10)
, act_ret_date DATE
, exp_ret_date DATE DEFAULT SYSDATE+2
, CONSTRAINT rental_copy_title_id_fk FOREIGN KEY (copy_id,
title_id)
 REFERENCES title_copy(copy_id,title_id)
, CONSTRAINT rental_id_pk PRIMARY KEY(book_date, copy_id,
title_id, member_id))
/

CREATE TABLE RESERVATION
(res_date DATE
, member_id NUMBER(10)
, title_id NUMBER(10)
, CONSTRAINT res_id_pk PRIMARY KEY(res_date, member_id,
title_id))
/

PROMPT Tables created.
DROP SEQUENCE title_id_seq;
DROP SEQUENCE member_id_seq;

PROMPT Creating Sequences...
CREATE SEQUENCE member_id_seq

```

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

START WITH 101
NOCACHE

CREATE SEQUENCE title_id_seq
 START WITH 92
 NOCACHE
/

PROMPT Sequences created.

PROMPT Run buildvid2.sql now to populate the above tables.

```



J. Enrique Lozano (lozanojose@gmail.com) has a  
Temporary license to use this Student Guide.

Results Script Output Explain Autotrace DBMS Output OWA Output

Error starting at line 12 in command:  
DROP TABLE rental CASCADE CONSTRAINTS  
Error report:  
SQL Error: ORA-00942: table or view does not exist  
00942. 00000 - "table or view does not exist"  
\*Cause:  
\*Action:

Error starting at line 13 in command:  
DROP TABLE reservation CASCADE CONSTRAINTS  
Error report:  
SQL Error: ORA-00942: table or view does not exist  
00942. 00000 - "table or view does not exist"  
\*Cause:  
\*Action:

Error starting at line 14 in command:  
DROP TABLE title\_copy CASCADE CONSTRAINTS  
Error report:

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:

Error starting at line 15 in command:
DROP TABLE title CASCADE CONSTRAINTS
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:

Error starting at line 16 in command:
DROP TABLE member CASCADE CONSTRAINTS
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:
Please wait while tables are created....
CREATE TABLE succeeded.
Tables created.

Error starting at line 80 in command:
DROP SEQUENCE title_id_seq
Error report:
SQL Error: ORA-02289: sequence does not exist
02289. 00000 - "sequence does not exist"
*Cause: The specified sequence does not exist, or the user does
not have the required privilege to perform this operation.
*Action: Make sure the sequence name is correct, and that you have
the right to perform the desired operation on this sequence.

Error starting at line 81 in command:
DROP SEQUENCE member_id_seq
Error report:
Error report:
SQL Error: ORA-02289: sequence does not exist
02289. 00000 - "sequence does not exist"
*Cause: The specified sequence does not exist, or the user does
not have the required privilege to perform this operation.
*Action: Make sure the sequence name is correct, and that you have
the right to perform the desired operation on this sequence.

Creating Sequences...
CREATE SEQUENCE succeeded.
CREATE SEQUENCE succeeded.
Sequences created.
Run buildvid2.sql now to populate the above tables.

```

- 2) Cargue y ejecute el script /home/oracle/labs/plpu/labs/buildvid2.sql para llenar todas las tablas creadas por el script buildvid1.sql.

**Ejecute el script /home/oracle/labs/plpu/labs/buildvid2.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:**

```
/* Script to build the Video Application (Part 2 -
buildvid2.sql)
```

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

This part of the script populates the tables that are
created using
buildvid1.sql
These are used by Part B of the Additional Practices of the
course.
You should run the script buildvid1.sql before running this
script to
 create the above tables.
*/
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Velasquez', 'Carmen',
'283 King Street', 'Seattle', '587-99-6666', '03-MAR-90');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Ngao', 'LaDoris',
'5 Modrany', 'Bratislava', '586-355-8882', '08-MAR-90');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Nagayama', 'Midori',
'68 Via Centrale', 'Sao Paolo', '254-852-5764', '17-JUN-
91');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Quick-To-See', 'Mark',
'6921 King Way', 'Lagos', '63-559-777', '07-APR-90');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Ropeburn', 'Audry',
'86 Chu Street', 'Hong Kong', '41-559-87', '04-MAR-90');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Urguhart', 'Molly',
'3035 Laurier Blvd.', 'Quebec', '418-542-9988', '18-JAN-
91');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Menchu', 'Roberta',
'Boulevard de Waterloo 41', 'Brussels', '322-504-2228',
'14-MAY-90');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Biri', 'Ben',
'398 High St.', 'Columbus', '614-455-9863', '07-APR-90');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Catchpole', 'Antoinette',
'88 Alfred St.', 'Brisbane', '616-399-1411', '09-FEB-92');

COMMIT;
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Willie and Christmas Too',
'All of Willie''s friends made a Christmas list for Santa,
but Willie has yet to create his own wish list.', 'G',
'CHILD', '05-OCT-95');
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)

```

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

VALUES (TITLE_ID_SEQ.NEXTVAL, 'Alien Again', 'Another
installment of science fiction history. Can the heroine save
the planet from the alien life form?', 'R', 'SCIFI',
'19-MAY-95');
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'The Glob', 'A meteor crashes
near a small American town and unleashes carivorous goo in
this classic.', 'NR', 'SCIFI', '12-AUG-95');
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'My Day Off', 'With a little
luck and a lot of ingenuity, a teenager skips school for a day
in New York.', 'PG', 'COMEDY', '12-JUL-95');
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Miracles on Ice', 'A six-
year-old has doubts about Santa Claus. But she discovers that
miracles really do exist.', 'PG', 'DRAMA', '12-SEP-95');
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Soda Gang', 'After
discovering a cache of drugs, a young couple find themselves
pitted against a vicious gang.', 'NR', 'ACTION', '01-JUN-95');
INSERT INTO title (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Interstellar Wars',
'Futuristic interstellar action movie. Can the rebels save the
humans from the evil Empire?', 'PG', 'SCIFI', '07-JUL-77');

COMMIT;

INSERT INTO title_copy VALUES (1,92, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,93, 'AVAILABLE');
INSERT INTO title_copy VALUES (2,93, 'RENTED');
INSERT INTO title_copy VALUES (1,94, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,95, 'AVAILABLE');
INSERT INTO title_copy VALUES (2,95, 'AVAILABLE');
INSERT INTO title_copy VALUES (3,95, 'RENTED');
INSERT INTO title_copy VALUES (1,96, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,97, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,98, 'RENTED');
INSERT INTO title_copy VALUES (2,98, 'AVAILABLE');
COMMIT;
INSERT INTO reservation VALUES (sysdate-1, 101, 93);
INSERT INTO reservation VALUES (sysdate-2, 106, 102);

COMMIT;

INSERT INTO rental VALUES (sysdate-1, 2, 101, 93, null,
sysdate+1);
INSERT INTO rental VALUES (sysdate-2, 3, 102, 95, null,
sysdate);

```

## **Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)**

```
INSERT INTO rental VALUES (sysdate-3, 1, 101, 98, null,
sysdate-1);
INSERT INTO rental VALUES (sysdate-4, 1, 106, 97, sysdate-2,
sysdate-2);
INSERT INTO rental VALUES (sysdate-3, 1, 101, 92, sysdate-2,
sysdate-1);

COMMIT;

PROMPT ** Tables built and data loaded **
```



## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

1 rows inserted
1 rows inserted
COMMIT succeeded.
1 rows inserted
COMMIT succeeded.
** Tables built and data loaded **

```

- 3) Cree un paquete denominado VIDEO\_PKG con los siguientes procedimientos y funciones:
- NEW\_MEMBER:** procedimiento público que agrega un nuevo miembro a la tabla MEMBER. Para el número de identificador (ID) de miembro, utilice la secuencia MEMBER\_ID\_SEQ; para la fecha de unión, utilice SYSDATE. Transfiera los demás valores que deseé insertar en una nueva fila como parámetros.
  - NEW\_RENTAL:** función pública sobrecargada para registrar un nuevo alquiler. Transfiera el número de identificador de título del video que el cliente desea alquilar, junto con el apellido del cliente o el número de identificador de miembro a la función. La función deberá devolver la fecha de vencimiento del video. Las fechas de vencimiento son tres días desde la fecha de alquiler del video. Si el estado de una película solicitada aparece como AVAILABLE en la tabla TITLE\_COPY para una copia de este título, actualice la tabla TITLE\_COPY y defina el estado en RENTED. Si no hay ninguna copia disponible, la función debe devolver el valor NULL. A continuación, inserte un nuevo registro en la tabla RENTAL, que identifique la fecha de reserva como la fecha de hoy, el número de identificador de copia, el número de identificador de título y la fecha de devolución esperada. Tenga en cuenta que puede que existan varios clientes con el mismo apellido. En este caso, haga que la función devuelva NULL y muestre una lista de todos los nombres de clientes que coincidan y sus números de identificador.
  - RETURN\_MOVIE:** procedimiento público que actualiza el estado de un video (disponible, alquilado o dañado) y define la fecha de devolución. Transfiera el identificador de título, el identificador de copia y el estado a este procedimiento. Compruebe si existen reservas para ese título y muestre un mensaje, si está reservado. Actualice la tabla RENTAL y defina la fecha de devolución real en la fecha de hoy. Actualice el estado en la tabla TITLE\_COPY según el parámetro de estado transferido al procedimiento.
  - RESERVE\_MOVIE:** procedimiento privado que se ejecuta sólo si todas las copias de videos solicitadas en el procedimiento NEW\_RENTAL tienen el estado RENTED. Transfiera el número de identificador de miembro y el número de identificador de título a este procedimiento. Inserte un nuevo registro en la tabla RESERVATION y registre la fecha de reserva, el número de identificador de miembro y el número de identificador de título. Imprima un mensaje que indique que una película está reservada y la fecha de devolución esperada.

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

- e) **EXCEPTION\_HANDLER:** procedimiento privado que se llama desde el manejador de excepciones de los programas públicos. Transfiera el número SQLCODE a este procedimiento y el nombre del programa (como cadena de texto) en el que se ha producido el error. Utilice RAISE\_APPLICATION\_ERROR para emitir un error personalizado. Empiece por una violación de clave única (-1) y una violación de clave ajena (-2292). Permita que el manejador de excepciones emita un error genérico para cualquier otro error.

Ejecute el script `/home/oracle/labs/plpu/solns/sol_ap_02_01_03.sql`. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

### Especificación del paquete VIDEO\_PKG

```
CREATE OR REPLACE PACKAGE video_pkg IS
 PROCEDURE new_member
 (p_lname IN member.last_name%TYPE,
 p_fname IN member.first_name%TYPE DEFAULT NULL,
 p_address IN member.address%TYPE DEFAULT NULL,
 p_city IN member.city%TYPE DEFAULT NULL,
 p_phone IN member.phone%TYPE DEFAULT NULL);

 FUNCTION new_rental
 (p_memberid IN rental.member_id%TYPE,
 p_titleid IN rental.title_id%TYPE)
 RETURN DATE;

 FUNCTION new_rental
 (p_membername IN member.last_name%TYPE,
 p_titleid IN rental.title_id%TYPE)
 RETURN DATE;

 PROCEDURE return_movie
 (p_titleid IN rental.title_id%TYPE,
 p_copyid IN rental.copy_id%TYPE,
 p_sts IN title_copy.status%TYPE);
END video_pkg;
/
SHOW ERRORS
```

```
CREATE OR REPLACE PACKAGE BODY video_pkg IS
 PROCEDURE exception_handler(errcode IN NUMBER, context IN VARCHAR2) IS
 BEGIN
 IF errcode = -1 THEN
 RAISE_APPLICATION_ERROR(-20001,
 'The number is assigned to this member is already in
use, |||
 'try again.');
 ELSIF errcode = -2291 THEN
```

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

 RAISE_APPLICATION_ERROR(-20002, context ||
 ' has attempted to use a foreign key value that is
invalid');
 ELSE
 RAISE_APPLICATION_ERROR(-20999, 'Unhandled error in ' ||
 context || '. Please contact your application ' ||
 'administrator with the following information: ' ||
 CHR(13) || SQLERRM);
 END IF;
END exception_handler;
PROCEDURE reserve_movie
(memberid IN reservation.member_id%TYPE,
 titleid IN reservation.title_id%TYPE) IS
CURSOR rented_csr IS
 SELECT exp_ret_date
 FROM rental
 WHERE title_id = titleid
 AND act_ret_date IS NULL;
BEGIN
 INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, memberid, titleid);
 COMMIT;
 FOR rented_rec IN rented_csr LOOP
 DBMS_OUTPUT.PUT_LINE('Movie reserved. Expected back on:
 ' || rented_rec.exp_ret_date);
 EXIT WHEN rented_csr%found;
 END LOOP;
EXCEPTION
 WHEN OTHERS THEN
 exception_handler(SQLCODE, 'RESERVE_MOVIE');
END reserve_movie;

PROCEDURE return_movie(
titleid IN rental.title_id%TYPE,
copyid IN rental.copy_id%TYPE,
sts IN title_copy.status%TYPE) IS
v_dummy VARCHAR2(1);
CURSOR res_csr IS
 SELECT *
 FROM reservation
 WHERE title_id = titleid;
BEGIN
 SELECT '' INTO v_dummy
 FROM title
 WHERE title_id = titleid;
 UPDATE rental
 SET act_ret_date = SYSDATE
 WHERE title_id = titleid
 AND copy_id = copyid AND act_ret_date IS NULL;
 UPDATE title_copy
 SET status = UPPER(sts)

```

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

 WHERE title_id = titleid AND copy_id = copyid;
FOR res_rec IN res_csr LOOP
 IF res_csr%FOUND THEN
 DBMS_OUTPUT.PUT_LINE('Put this movie on hold -- ' ||
 'reserved by member #' || res_rec.member_id);
 END IF;
END LOOP;
EXCEPTION
 WHEN OTHERS THEN
 exception_handler(SQLCODE, 'RETURN_MOVIE');
END return_movie;
FUNCTION new_rental(
 memberid IN rental.member_id%TYPE,
 titleid IN rental.title_id%TYPE) RETURN DATE IS
CURSOR copy_csr IS
 SELECT * FROM title_copy
 WHERE title_id = titleid
 FOR UPDATE;
 flag BOOLEAN := FALSE;
BEGIN

 FOR copy_rec IN copy_csr LOOP
 IF copy_rec.status = 'AVAILABLE' THEN
 UPDATE title_copy
 SET status = 'RENTED'
 WHERE CURRENT OF copy_csr;
 INSERT INTO rental(book_date, copy_id, member_id,
 title_id, exp_ret_date)
 VALUES (SYSDATE, copy_rec.copy_id, memberid,
 titleid, SYSDATE + 3);
 flag := TRUE;
 EXIT;
 END IF;
 END LOOP;
 COMMIT;
 IF flag THEN
 RETURN (SYSDATE + 3);
 ELSE
 reserve_movie(memberid, titleid);
 RETURN NULL;
 END IF;
EXCEPTION
 WHEN OTHERS THEN
 exception_handler(SQLCODE, 'NEW_RENTAL');
END new_rental;

FUNCTION new_rental(
 membername IN member.last_name%TYPE,
 titleid IN rental.title_id%TYPE) RETURN DATE IS
CURSOR copy_csr IS
 SELECT * FROM title_copy
 WHERE title_id = titleid

```

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

 FOR UPDATE;
flag BOOLEAN := FALSE;
memberid member.member_id%TYPE;
CURSOR member_csr IS
 SELECT member_id, last_name, first_name
 FROM member
 WHERE LOWER(last_name) = LOWER(membername)
 ORDER BY last_name, first_name;
BEGIN
 SELECT member_id INTO memberid
 FROM member
 WHERE lower(last_name) = lower(membername);
FOR copy_rec IN copy_csr LOOP
 IF copy_rec.status = 'AVAILABLE' THEN
 UPDATE title_copy
 SET status = 'RENTED'
 WHERE CURRENT OF copy_csr;
 INSERT INTO rental (book_date, copy_id, member_id,
 title_id, exp_ret_date)
 VALUES (SYSDATE, copy_rec.copy_id, memberid,
 titleid, SYSDATE + 3);
 flag := TRUE;
 EXIT;
 END IF;
END LOOP;
COMMIT;
IF flag THEN
 RETURN (SYSDATE + 3);
ELSE
 reserve_movie(memberid, titleid);
 RETURN NULL;
END IF;
EXCEPTION
 WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE(
 'Warning! More than one member by this name.');
 FOR member_rec IN member_csr LOOP
 DBMS_OUTPUT.PUT_LINE(member_rec.member_id || CHR(9) ||
 member_rec.last_name || ', ' ||
 member_rec.first_name);
 END LOOP;
 RETURN NULL;
 WHEN OTHERS THEN
 exception_handler(SQLCODE, 'NEW_RENTAL');
END new_rental;

PROCEDURE new_member(
 lname IN member.last_name%TYPE,
 fname IN member.first_name%TYPE DEFAULT NULL,
 address IN member.address%TYPE DEFAULT NULL,
 city IN member.city%TYPE DEFAULT NULL,
 phone IN member.phone%TYPE DEFAULT NULL) IS

```

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

BEGIN
 INSERT INTO member(member_id, last_name, first_name,
 address, city, phone, join_date)
 VALUES(member_id_seq.NEXTVAL, lname, fname,
 address, city, phone, SYSDATE);
 COMMIT;
CREATE OR REPLACE PACKAGE BODY video_pkg IS
 PROCEDURE exception_handler(errcode IN NUMBER, p_context IN VARCHAR2) IS
 BEGIN
 IF errcode = -1 THEN
 RAISE_APPLICATION_ERROR(-20001,
 'The number is assigned to this member is already in
use, ' ||
 'try again.');
 ELSIF errcode = -2291 THEN
 RAISE_APPLICATION_ERROR(-20002, p_context ||
 ' has attempted to use a foreign key value that is
invalid');
 ELSE
 RAISE_APPLICATION_ERROR(-20999, 'Unhandled error in ' ||
 p_context || '. Please contact your application ' ||
 'administrator with the following information: ' ||
 CHR(13) || SQLERRM);
 END IF;
 END exception_handler;

 PROCEDURE reserve_movie
 (p_memberid IN reservation.member_id%TYPE,
 p_titleid IN reservation.title_id%TYPE) IS
 CURSOR c_rented_csr IS
 SELECT exp_ret_date
 FROM rental
 WHERE title_id = p_titleid
 AND act_ret_date IS NULL;
 BEGIN
 INSERT INTO reservation (res_date, member_id, title_id)
 VALUES (SYSDATE, p_memberid, p_titleid);
 COMMIT;
 FOR rented_rec IN c_rented_csr LOOP
 DBMS_OUTPUT.PUT_LINE('Movie reserved. Expected back on:
 ' ||
 rented_rec.exp_ret_date);
 EXIT WHEN c_rented_csr%found;
 END LOOP;
 EXCEPTION
 WHEN OTHERS THEN
 exception_handler(SQLCODE, 'RESERVE_MOVIE');
 END reserve_movie;

 PROCEDURE return_movie(
 p_titleid IN rental.title_id%TYPE,

```

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

p_copyid IN rental.copy_id%TYPE,
p_sts IN title_copy.status%TYPE) IS
v_dummy VARCHAR2(1);
CURSOR c_res_csr IS
 SELECT *
 FROM reservation
 WHERE title_id = p_titleid;
BEGIN
 SELECT '' INTO v_dummy
 FROM title
 WHERE title_id = p_titleid;
UPDATE rental
 SET act_ret_date = SYSDATE
 WHERE title_id = p_titleid
 AND copy_id = p_copyid AND act_ret_date IS NULL;
UPDATE title_copy
 SET status = UPPER(p_sts)
 WHERE title_id = p_titleid AND copy_id = p_copyid;
FOR res_rec IN c_res_csr LOOP
 IF c_res_csr%FOUND THEN
 DBMS_OUTPUT.PUT_LINE('Put this movie on hold -- ||'
 'reserved by member #' || res_rec.member_id);
 END IF;
END LOOP;
EXCEPTION
 WHEN OTHERS THEN
 exception_handler(SQLCODE, 'RETURN_MOVIE');
END return_movie;

FUNCTION new_rental(
 p_memberid IN rental.member_id%TYPE,
 p_titleid IN rental.title_id%TYPE) RETURN DATE IS
CURSOR c_copy_csr IS
 SELECT * FROM title_copy
 WHERE title_id = p_titleid
 FOR UPDATE;
 v_flag BOOLEAN := FALSE;
BEGIN
 FOR copy_rec IN c_copy_csr LOOP
 IF copy_rec.status = 'AVAILABLE' THEN
 UPDATE title_copy
 SET status = 'RENTED'
 WHERE CURRENT OF c_copy_csr;
 INSERT INTO rental(book_date, copy_id, member_id,
 title_id, exp_ret_date)
 VALUES (SYSDATE, copy_rec.copy_id, p_memberid,
 p_titleid, SYSDATE + 3);
 v_flag := TRUE;
 EXIT;
 END IF;
 END LOOP;
 COMMIT;

```

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

 IF v_flag THEN
 RETURN (SYSDATE + 3);
 ELSE
 reserve_movie(p_memberid, p_titleid);
 RETURN NULL;
 END IF;
EXCEPTION
 WHEN OTHERS THEN
 exception_handler(SQLCODE, 'NEW_RENTAL');
 RETURN NULL;
END new_rental;

FUNCTION new_rental(
 p_membername IN member.last_name%TYPE,
 p_titleid IN rental.title_id%TYPE) RETURN DATE IS
CURSOR c_copy_csr IS
 SELECT * FROM title_copy
 WHERE title_id = p_titleid
 FOR UPDATE;
v_flag BOOLEAN := FALSE;
v_memberid member.member_id%TYPE;
CURSOR c_member_csr IS
 SELECT member_id, last_name, first_name
 FROM member
 WHERE LOWER(last_name) = LOWER(p_membername)
 ORDER BY last_name, first_name;
BEGIN
 SELECT member_id INTO v_memberid
 FROM member
 WHERE lower(last_name) = lower(p_membername);
 FOR copy_rec IN c_copy_csr LOOP
 IF copy_rec.status = 'AVAILABLE' THEN
 UPDATE title_copy
 SET status = 'RENTED'
 WHERE CURRENT OF c_copy_csr;
 INSERT INTO rental (book_date, copy_id, member_id,
 title_id, exp_ret_date)
 VALUES (SYSDATE, copy_rec.copy_id, v_memberid,
 p_titleid, SYSDATE + 3);
 v_flag := TRUE;
 EXIT;
 END IF;
 END LOOP;
 COMMIT;
 IF v_flag THEN
 RETURN(SYSDATE + 3);
 ELSE
 reserve_movie(v_memberid, p_titleid);
 RETURN NULL;
 END IF;
EXCEPTION

```

## **Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)**

```

WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE(
 'Warning! More than one member by this name.');
 FOR member_rec IN c_member_csr LOOP
 DBMS_OUTPUT.PUT_LINE(member_rec.member_id || CHR(9) ||
 member_rec.last_name || ', ' ||
 member_rec.first_name);
 END LOOP;
 RETURN NULL;
WHEN OTHERS THEN
 exception_handler(SQLCODE, 'NEW_RENTAL');
 RETURN NULL;
END new_rental;

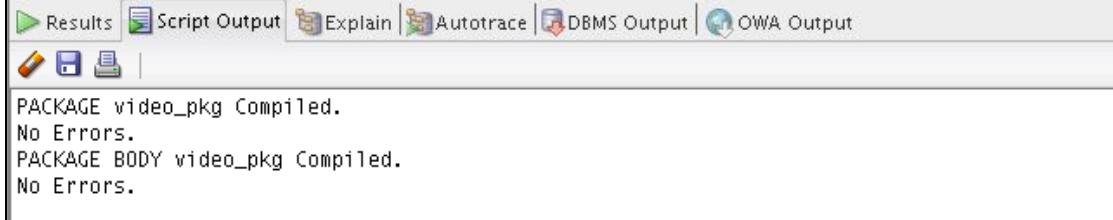
PROCEDURE new_member(
 p_lname IN member.last_name%TYPE,
 p_fname IN member.first_name%TYPE DEFAULT NULL,
 p_address IN member.address%TYPE DEFAULT NULL,
 p_city IN member.city%TYPE DEFAULT NULL,
 p_phone IN member.phone%TYPE DEFAULT NULL) IS
BEGIN
 INSERT INTO member(member_id, last_name, first_name,
 address, city, phone, join_date)
 VALUES(member_id_seq.NEXTVAL, p_lname, p_fname,
 p_address, p_city, p_phone, SYSDATE);
 COMMIT;
EXCEPTION
 WHEN OTHERS THEN
 exception_handler(SQLCODE, 'NEW_MEMBER');
END new_member;
END video_pkg;
/
SHOW ERRORS

```

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)



The screenshot shows the 'Select Connection' dialog box. It has a title bar 'Select Connection'. Inside, it says 'Select the connection you wish to use from the list, or create a new connection.' Below this is a dropdown menu labeled 'Connection:' with 'VideoCompany' selected. There are three buttons at the bottom: 'Help', 'OK', and 'Cancel'.

The screenshot shows the Oracle SQL Developer interface. At the top, there are tabs: 'Results', 'Script Output', 'Explain', 'Autotrace', 'DBMS Output', and 'OWA Output'. Below these are icons for 'New', 'Save', and 'Print'. The main area displays the following text:

```

PACKAGE video_pkg Compiled.
No Errors.
PACKAGE BODY video_pkg Compiled.
No Errors.

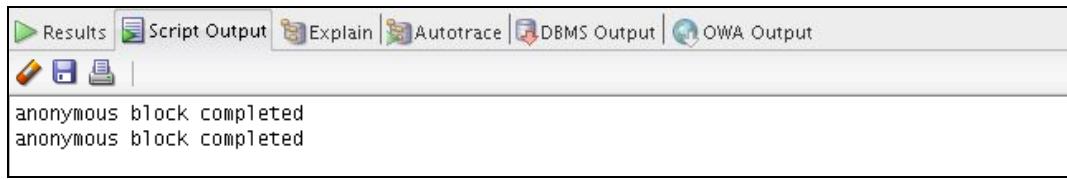
```

- 4) Utilice los siguientes scripts situados en el directorio /home/oracle/labs/plpu/soln para probar las rutinas. Asegúrese de activar SERVEROUTPUT:
- Agregue dos miembros con sol\_ap\_02\_01\_04\_a.sql.
- Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_02\_01\_04\_a.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```

SET SERVEROUTPUT ON
EXECUTE video_pkg.new_member('Haas', 'James', 'Chestnut
Street', 'Boston', '617-123-4567')
EXECUTE video_pkg.new_member('Biri', 'Allan', 'Hiawatha
Drive', 'New York', '516-123-4567')

```



## **Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)**

- b) Agregue nuevos alquileres de videos con `sol_ap_02_01_04_b.sql`.

Ejecute el script `/home/oracle/labs/plpu/solns/sol_ap_02_01_04_b.sql`. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(110, 98))
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(109, 93))
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(107, 98))
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental('Biri', 97))
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(97, 97))
```



```
anonymous block completed
02-JUL-09

anonymous block completed
02-JUL-09

anonymous block completed
Movie reserved. Expected back on: 28-JUN-09

anonymous block completed
Warning! More than one member by this name.
111 Biri, Allan
108 Biri, Ben

Error starting at line 7 in command:
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(97, 97))
Error report:
ORA-20002: NEW_RENTAL has attempted to use a foreign key value that is invalid
ORA-06512: at "ORA62.VIDEO_PKG", line 9
ORA-06512: at "ORA62.VIDEO_PKG", line 103
ORA-06512: at line 1
```

- c) Devuelva películas con el script `sol_ap_02_01_04_c.sql`.

Ejecute el script `/home/oracle/labs/plpu/solns/sol_ap_02_01_04_c.sql`. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```
SET SERVEROUTPUT ON

EXECUTE video_pkg.return_movie(98, 1, 'AVAILABLE')
EXECUTE video_pkg.return_movie(95, 3, 'AVAILABLE')
EXECUTE video_pkg.return_movie(111, 1, 'RENTED')
```



anonymous block completed  
Put this movie on hold -- reserved by member #107

anonymous block completed

Error starting at line 5 in command:  
EXECUTE video\_pkg.return\_movie(111, 1, 'RENTED')  
Error report:  
ORA-20999: Unhandled error in RETURN\_MOVIE. Please contact your application administrator with the following information:  
ORA-01403: no data found  
ORA-06512: at "ORA62.VIDEO\_PKG", line 12  
ORA-06512: at "ORA62.VIDEO\_PKG", line 69  
ORA-06512: at line 1

- 5) Las horas laborables para el videoclub son de 8:00 a.m. a 10:00 p.m. (de domingo a viernes) y de 8:00 a.m. a 12:00 p.m. (sábados). Para asegurarse de que las tablas sólo se puedan modificar durante estas horas, cree un procedimiento almacenado al que se llame con los disparadores de las tablas.

- a) Cree un procedimiento almacenado denominado TIME\_CHECK, que compruebe la hora actual con respecto a las horas laborables. Si la hora actual no está dentro de las horas laborables, utilice el procedimiento RAISE\_APPLICATION\_ERROR para emitir el mensaje adecuado.

Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_02\_01\_05\_a.sql. El código, la petición de datos de conexión y el resultado se muestran de la siguiente forma:

```
CREATE OR REPLACE PROCEDURE time_check IS
BEGIN
 IF ((TO_CHAR(SYSDATE, 'D') BETWEEN 1 AND 6) AND
 (TO_DATE(TO_CHAR(SYSDATE, 'hh24:mi'), 'hh24:mi') NOT
 BETWEEN
```

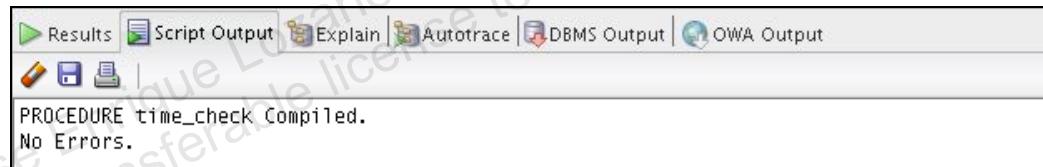
## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

 TO_DATE('08:00', 'hh24:mi') AND TO_DATE('22:00',
'hh24:mi'))
 OR ((TO_CHAR(SYSDATE, 'D') = 7)
 AND (TO_DATE(TO_CHAR(SYSDATE, 'hh24:mi'),
'hh24:mi') NOT BETWEEN
 TO_DATE('08:00', 'hh24:mi') AND TO_DATE('24:00',
'hh24:mi'))) THEN
 RAISE_APPLICATION_ERROR(-20999,
 'Data changes restricted to office hours.');
 END IF;
END time_check;
/
SHOW ERRORS

PROCEDURE time_check Compiled.
No Errors.

```



- b) Cree un disparador en cada una de las cinco tablas. Arranque el disparador antes de insertar, actualizar y suprimir los datos de las tablas. Llame al procedimiento TIME\_CHECK desde cada disparador.

**Ejecute el script /home/oracle/labs/plpu/solns/sol\_ap\_02\_01\_05\_b.sql. El código y el resultado se muestran de la siguiente forma:**

```

CREATE OR REPLACE TRIGGER member_trig
 BEFORE INSERT OR UPDATE OR DELETE ON member
CALL time_check
/

CREATE OR REPLACE TRIGGER rental_trig
 BEFORE INSERT OR UPDATE OR DELETE ON rental
CALL time_check
/

```

## Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)

```

CREATE OR REPLACE TRIGGER title_copy_trig
 BEFORE INSERT OR UPDATE OR DELETE ON title_copy
CALL time_check
/

CREATE OR REPLACE TRIGGER title_trig
 BEFORE INSERT OR UPDATE OR DELETE ON title
CALL time_check
/
CREATE OR REPLACE TRIGGER reservation_trig
 BEFORE INSERT OR UPDATE OR DELETE ON reservation
CALL time_check
/
TRIGGER member_trig Compiled.
TRIGGER rental_trig Compiled.
TRIGGER title_copy_trig Compiled.
TRIGGER title_trig Compiled.
TRIGGER reservation_trig Compiled.

```



```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
[Icons] | [Icon] | [Icon] | [Icon] | [Icon] | [Icon]
TRIGGER member_trig Compiled.
TRIGGER rental_trig Compiled.
TRIGGER title_copy_trig Compiled.
TRIGGER title_trig Compiled.
TRIGGER reservation_trig Compiled.

```

- c) Pruebe los disparadores.

**Nota:** para que el disparador falle, es posible que tenga que cambiar la hora para que esté fuera del rango de la hora actual de la clase. Por ejemplo, mientras realiza la prueba, puede que desee que las horas válidas de vídeo del disparador sean desde las 18:00 p.m. a las 08:00 a.m.

## **Soluciones a la Práctica 2-1: Creación del Paquete VIDEO\_PKG (continuación)**

Ejecute el script `/home/oracle/labs/plpu/solns/sol_ap_02_01_05_C.sql`. El código y el resultado se muestran de la siguiente forma:

```
-- First determine current timezone and time
SELECT SESSIONTIMEZONE,
 TO_CHAR(CURRENT_DATE, 'DD-MON-YYYY HH24:MI')
 CURR_DATE
 FROM DUAL;

-- Change your time zone usinge [+|-]HH:MI format such that
-- the current time returns a time between 6pm and 8am

ALTER SESSION SET TIME_ZONE='-07:00';

-- Add a new member (for a sample test)

EXECUTE video_pkg.new_member('Elias', 'Elliane', 'Vine
Street', 'California', '789-123-4567')

BEGIN video_pkg.new_member('Elias', 'Elliane', 'Vine
Street', 'California', '789-123-4567'); END;

-- Restore the original time zone for your session.

ALTER SESSION SET TIME_ZONE='-07:00';
```

|                                                                                                |  | SESSIONTIMEZONE | CURR_DATE         |
|------------------------------------------------------------------------------------------------|--|-----------------|-------------------|
|                                                                                                |  | Etc/GMT-7       | 29-JUN-2009 21:51 |
| 1 rows selected                                                                                |  |                 |                   |
| <pre>ALTER SESSION SET succeeded. anonymous block completed ALTER SESSION SET succeeded.</pre> |  |                 |                   |

Jose Enrique Lozano (lozanojose@gmail.com) has a  
non-transferable license to use this Student Guide.

## Descripciones de las Tablas

ORACLE

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

## Descripción de los Esquemas

### Descripción General

La compañía de ejemplo que se utiliza en los esquemas de ejemplo de la base de datos Oracle opera en todo el mundo para surtir pedidos de varios productos distintos. La compañía tiene tres divisiones:

- **Human Resources:** realiza un seguimiento de la información de los empleados e instalaciones
- **Order Entry:** realiza un seguimiento de los inventarios de productos y de las ventas a través de distintos canales
- **Sales History:** realiza un seguimiento de los datos estadísticos de negocio para facilitar las decisiones de negocio

Cada una de estas divisiones se representa mediante un esquema. En este curso, accederá a los objetos de todos los esquemas. No obstante, los ejemplos, las demostraciones y las prácticas se centran en el esquema de Human Resources (HR).

Todos los scripts necesarios para crear los esquemas de ejemplo están en la carpeta \$ORACLE\_HOME/demo/schema/.

### Human Resources (HR)

Se trata del esquema utilizado en este curso. En los registros de Human Resources (HR), cada empleado tiene un número de identificación, una dirección de correo electrónico, un código de identificación de trabajo, un salario y un superior. Algunos empleados ganan comisiones además de su salario.

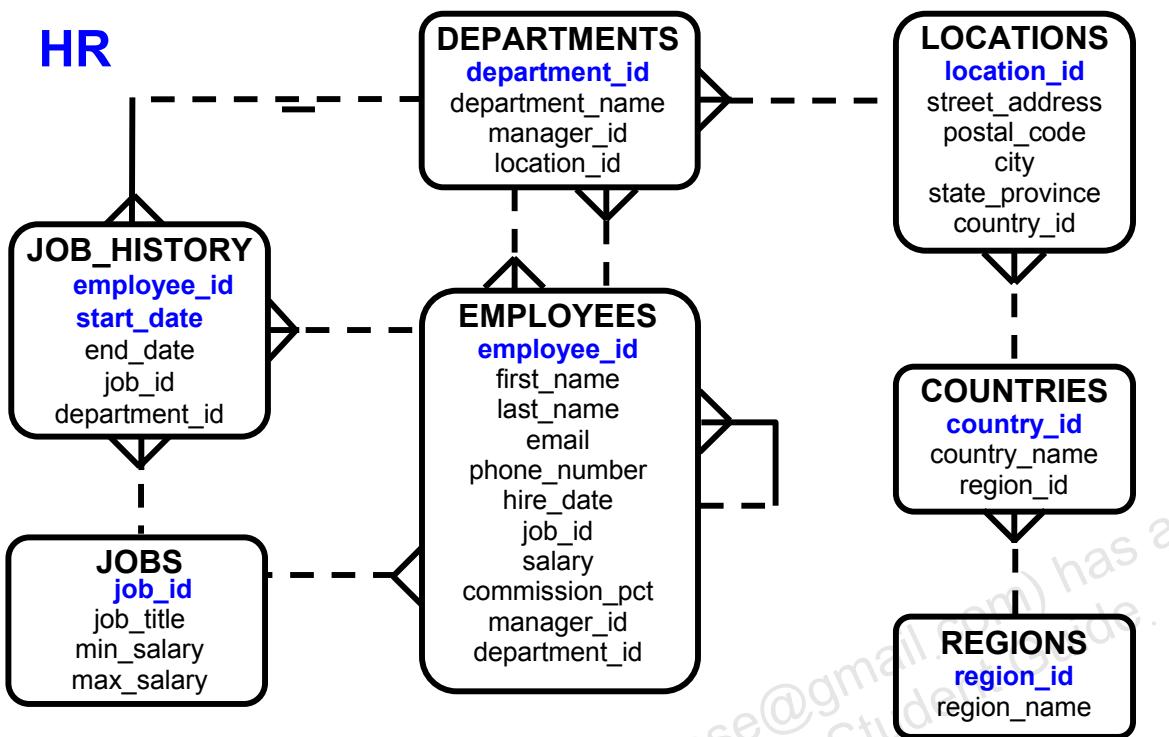
La compañía también registra información sobre trabajos dentro de la organización. Cada trabajo dispone de un código de identificación, un cargo y un rango de salario máximo y mínimo para el trabajo. Algunos empleados llevan bastante tiempo en la compañía y han desempeñado distintos trabajos en ella. Cuando un empleado dimite, se registran la duración del puesto de trabajo del empleado, el número de identificación del trabajo y el departamento.

La compañía de ejemplo se extiende por distintas regiones, por lo que registra las ubicaciones de sus almacenes y departamentos. A cada empleado se le asigna un departamento y cada departamento se identifica mediante un número de departamento único y una abreviatura. A cada departamento se le asocia una ubicación y cada ubicación tiene una dirección completa que incluye la calle, el código postal, la ciudad, el estado o provincia y el código del país.

En los lugares donde se encuentran los departamentos y los almacenes, la compañía registra detalles como el nombre del país, el símbolo y el nombre de la divisa, y la región en la que se encuentra dicho país.

## Diagrama de Entidad/Relación de HR

Unauthorized reproduction or distribution prohibited. Copyright© 2013, Oracle and/or its affiliates.



## Descripciones de la Tabla Human Resources (HR)

DESCRIBE countries

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

SELECT \* FROM countries

| COUNTRY_ID | COUNTRY_NAME         | REGION_ID |
|------------|----------------------|-----------|
| 1 AR       | Argentina            | 2         |
| 2 AU       | Australia            | 3         |
| 3 BE       | Belgium              | 1         |
| 4 BR       | Brazil               | 2         |
| 5 CA       | Canada               | 2         |
| 6 CH       | Switzerland          | 1         |
| 7 CN       | China                | 3         |
| 8 DE       | Germany              | 1         |
| 9 DK       | Denmark              | 1         |
| 10 EG      | Egypt                | 4         |
| 11 FR      | France               | 1         |
| 12 HK      | HongKong             | 3         |
| 13 IL      | Israel               | 4         |
| 14 IN      | India                | 3         |
| 15 IT      | Italy                | 1         |
| 16 JP      | Japan                | 3         |
| 17 KW      | Kuwait               | 4         |
| 18 MX      | Mexico               | 2         |
| 19 NG      | Nigeria              | 4         |
| 20 NL      | Netherlands          | 1         |
| 21 SG      | Singapore            | 3         |
| 22 UK      | United Kingdom       | 1         |
| 23 US      | United States of ... | 2         |
| 24 ZM      | Zambia               | 4         |
| 25 ZW      | Zimbabwe             | 4         |

## Descripciones de la Tabla Human Resources (HR) (continuación)

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 |
|---------------|--------------------------|------------|-------------|
| 1             | 10 Administration        | 200        | 1700        |
| 2             | 20 Marketing             | 201        | 1800        |
| 3             | 30 Purchasing            | 114        | 1700        |
| 4             | 40 Human Resources       | 203        | 2400        |
| 5             | 50 Shipping              | 121        | 1500        |
| 6             | 60 IT                    | 103        | 1400        |
| 7             | 70 Public Relations      | 204        | 2700        |
| 8             | 80 Sales                 | 145        | 2500        |
| 9             | 90 Executive             | 100        | 1700        |
| 10            | 100 Finance              | 108        | 1700        |
| 11            | 110 Accounting           | 205        | 1700        |
| 12            | 120 Treasury             | (null)     | 1700        |
| 13            | 130 Corporate Tax        | (null)     | 1700        |
| 14            | 140 Control And Credit   | (null)     | 1700        |
| 15            | 150 Shareholder Services | (null)     | 1700        |
| 16            | 160 Benefits             | (null)     | 1700        |
| 17            | 170 Manufacturing        | (null)     | 1700        |
| 18            | 180 Construction         | (null)     | 1700        |
| 19            | 190 Contracting          | (null)     | 1700        |
| 20            | 200 Operations           | (null)     | 1700        |
| 21            | 210 IT Support           | (null)     | 1700        |
| 22            | 220 NOC                  | (null)     | 1700        |
| 23            | 230 IT Helpdesk          | (null)     | 1700        |
| 24            | 240 Government Sales     | (null)     | 1700        |
| 25            | 250 Retail Sales         | (null)     | 1700        |
| 26            | 260 Recruiting           | (null)     | 1700        |
| 27            | 270 Payroll              | (null)     | 1700        |
| 28            | 980 Education            | (null)     | 2500        |
| 29            | 280 Training             | (null)     | 2400        |

## Descripciones de la Tabla Human Resources (HR) (continuación)

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)    |

SELECT \* FROM employees

| EMPLOYEE_ID | FIRST_NAME      | LAST_NAME   | EMAIL      | PHONE_NUMBER | HIRE_DATE | JOB_ID     | SALARY | COMMISSION_PCT | MANAGER_ID | DEPARTMENT_ID |
|-------------|-----------------|-------------|------------|--------------|-----------|------------|--------|----------------|------------|---------------|
| 1           | 100 Steven      | King        | SKING      | 515.123.4567 | 17-JUN-87 | AD_PRES    | 24000  | (null)         | (null)     | 90            |
| 1           | 100 Steven      | King        | SKING      | 515.123.4567 | 17-JUN-87 | AD_PRES    | 24000  | (null)         | (null)     | 90            |
| 3           | 102 Lex         | De Haan     | LDEHAAN    | 515.123.4569 | 13-JAN-93 | AD_VP      | 17000  | (null)         | 100        | 90            |
| 4           | 103 Alexander   | Hunold      | AHUNOLD    | 590.423.4567 | 03-JAN-90 | IT_PROG    | 9000   | (null)         | 102        | 60            |
| 5           | 104 Bruce       | Ernst       | BERNSTEIN  | 590.423.4568 | 21-MAY-91 | IT_PROG    | 6000   | (null)         | 103        | 60            |
| 6           | 105 David       | Austin      | DAUSTIN    | 590.423.4569 | 25-JUN-97 | IT_PROG    | 4800   | (null)         | 103        | 60            |
| 7           | 106 Valli       | Pataballa   | VPATABALLA | 590.423.4560 | 05-FEB-98 | IT_PROG    | 4800   | (null)         | 103        | 60            |
| 8           | 107 Diana       | Lorentz     | DLORENTZ   | 590.423.5567 | 07-FEB-99 | IT_PROG    | 4200   | (null)         | 103        | 60            |
| 9           | 108 Nancy       | Greenberg   | NGRGREENB  | 515.124.4569 | 17-AUG-94 | FI_MGR     | 12000  | (null)         | 101        | 100           |
| 10          | 109 Daniel      | Faviet      | DFAVIET    | 515.124.4169 | 16-AUG-94 | FI_ACCOUNT | 9000   | (null)         | 108        | 100           |
| 11          | 110 John        | Chen        | JCHEN      | 515.124.4269 | 28-SEP-97 | FI_ACCOUNT | 8200   | (null)         | 108        | 100           |
| 12          | 111 Ismael      | Sciarra     | ISCIARRA   | 515.124.4369 | 30-SEP-97 | FI_ACCOUNT | 7700   | (null)         | 108        | 100           |
| 13          | 112 Jose Manuel | Urman       | JMURMAN    | 515.124.4469 | 07-MAR-98 | FI_ACCOUNT | 7800   | (null)         | 108        | 100           |
| 14          | 113 Luis        | Popp        | LPOPP      | 515.124.4567 | 07-DEC-99 | FI_ACCOUNT | 6900   | (null)         | 108        | 100           |
| 15          | 114 Den         | Raphaely    | DRAPEL     | 515.127.4561 | 07-DEC-94 | PU_MAN     | 11000  | (null)         | 100        | 30            |
| 16          | 115 Alexander   | Khoo        | AKHOO      | 515.127.4562 | 18-MAY-95 | PU_CLERK   | 3100   | (null)         | 114        | 30            |
| 17          | 116 Shelli      | Baida       | SBAIL      | 515.127.4563 | 24-DEC-97 | PU_CLERK   | 2900   | (null)         | 114        | 30            |
| 18          | 117 Sigal       | Tobias      | STOBIA     | 515.127.4564 | 24-JUL-97 | PU_CLERK   | 2800   | (null)         | 114        | 30            |
| 19          | 118 Guy         | Himuro      | GHIMURO    | 515.127.4565 | 15-NOV-98 | PU_CLERK   | 2600   | (null)         | 114        | 30            |
| 20          | 119 Karen       | Colmenares  | KCOLEM     | 515.127.4566 | 10-AUG-99 | PU_CLERK   | 2500   | (null)         | 114        | 30            |
| 21          | 120 Matthew     | Weiss       | MWEISS     | 650.123.1234 | 18-JUL-96 | ST_MAN     | 8000   | (null)         | 100        | 50            |
| 22          | 121 Adam        | Fripp       | AFRIPP     | 650.123.2234 | 10-APR-97 | ST_MAN     | 8200   | (null)         | 100        | 50            |
| 23          | 122 Payam       | Kaufling    | PKAUF      | 650.123.3234 | 01-MAY-95 | ST_MAN     | 7900   | (null)         | 100        | 50            |
| 24          | 123 Shanta      | Vollman     | SVOLL      | 650.123.4234 | 10-OCT-97 | ST_MAN     | 6500   | (null)         | 100        | 50            |
| 25          | 124 Kevin       | Mourgos     | KMOUR      | 650.123.5234 | 16-NOV-99 | ST_MAN     | 5800   | (null)         | 100        | 50            |
| 26          | 125 Julia       | Nayer       | JNAYER     | 650.124.1214 | 16-JUL-97 | ST_CLERK   | 3200   | (null)         | 120        | 50            |
| 27          | 126 Irene       | Mikkilineni | IMIKKI     | 650.124.1224 | 28-SEP-98 | ST_CLERK   | 2700   | (null)         | 120        | 50            |
| 28          | 127 James       | Landry      | JLAUDRY    | 650.124.1334 | 14-JAN-99 | ST_CLERK   | 2400   | (null)         | 120        | 50            |
| 29          | 128 Steven      | Markle      | SMAKLE     | 650.124.1434 | 08-MAR-00 | ST_CLERK   | 2200   | (null)         | 120        | 50            |
| 30          | 129 Laura       | Bissot      | LBISOT     | 650.124.5234 | 20-AUG-97 | ST_CLERK   | 3300   | (null)         | 121        | 50            |
| 31          | 130 Mozhe       | Atkinson    | MATKIN     | 650.124.6234 | 30-OCT-97 | ST_CLERK   | 2800   | (null)         | 121        | 50            |
| 32          | 131 James       | Marlow      | JAMAR      | 650.124.7234 | 16-FEB-97 | ST_CLERK   | 2500   | (null)         | 121        | 50            |
| 33          | 132 TJ          | Olson       | TJOLSON    | 650.124.8234 | 10-APR-99 | ST_CLERK   | 2100   | (null)         | 121        | 50            |
| 34          | 133 Jason       | Mallin      | JMALLIN    | 650.127.1934 | 14-JUN-96 | ST_CLERK   | 3300   | (null)         | 122        | 50            |
| 35          | 134 Michael     | Rogers      | MROGER     | 650.127.1834 | 26-AUG-98 | ST_CLERK   | 2900   | (null)         | 122        | 50            |
| 36          | 135 Ki          | Gee         | KGEE       | 650.127.1734 | 12-DEC-99 | ST_CLERK   | 2400   | (null)         | 122        | 50            |
| 37          | 136 Hazel       | Philtanker  | HPHTANK    | 650.127.1634 | 06-FEB-00 | ST_CLERK   | 2200   | (null)         | 122        | 50            |
| 38          | 137 Renske      | Ladwig      | RLALDWIG   | 650.121.1234 | 14-JUL-95 | ST_CLERK   | 3600   | (null)         | 123        | 50            |

## Descripciones de la Tabla Human Resources (HR) (continuación)

### Employees (continuación)

|    |                 |           |         |                   |           |          |       |        |     |    |
|----|-----------------|-----------|---------|-------------------|-----------|----------|-------|--------|-----|----|
| 39 | 138 Stephen     | Stiles    | SSTI... | 650.121.2034      | 26-OCT-97 | ST_CLERK | 3200  | (null) | 123 | 50 |
| 40 | 139 John        | Seo       | JSEO    | 650.121.2019      | 12-FEB-98 | ST_CLERK | 2700  | (null) | 123 | 50 |
| 41 | 140 Joshua      | Patel     | JPAT... | 650.121.1834      | 06-APR-98 | ST_CLERK | 2500  | (null) | 123 | 50 |
| 42 | 141 Trenna      | Rajs      | TRAJS   | 650.121.8009      | 17-OCT-95 | ST_CLERK | 3500  | (null) | 124 | 50 |
| 43 | 142 Curtis      | Davies    | CDA...  | 650.121.2994      | 29-JAN-97 | ST_CLERK | 3100  | (null) | 124 | 50 |
| 44 | 143 Randall     | Matos     | RMA...  | 650.121.2874      | 15-MAR-98 | ST_CLERK | 2600  | (null) | 124 | 50 |
| 45 | 144 Peter       | Vargas    | PVA...  | 650.121.2004      | 09-JUL-98 | ST_CLERK | 2500  | (null) | 124 | 50 |
| 46 | 145 John        | Russell   | JRU...  | 011.44.1344.42... | 01-OCT-96 | SA_MAN   | 14000 | 0.4    | 100 | 80 |
| 47 | 146 Karen       | Partners  | KPA...  | 011.44.1344.46... | 05-JAN-97 | SA_MAN   | 13500 | 0.3    | 100 | 80 |
| 48 | 147 Alberto     | Errazuriz | AER...  | 011.44.1344.42... | 10-MAR-97 | SA_MAN   | 12000 | 0.3    | 100 | 80 |
| 49 | 148 Gerald      | Cambrault | GCA...  | 011.44.1344.61... | 15-OCT-99 | SA_MAN   | 11000 | 0.3    | 100 | 80 |
| 50 | 149 Eleni       | Zlotkey   | EZL...  | 011.44.1344.42... | 29-JAN-00 | SA_MAN   | 10500 | 0.2    | 100 | 80 |
| 51 | 150 Peter       | Tucker    | PTU...  | 011.44.1344.12... | 30-JAN-97 | SA REP   | 10000 | 0.3    | 145 | 80 |
| 52 | 151 David       | Bernstein | DBE...  | 011.44.1344.34... | 24-MAR-97 | SA REP   | 9500  | 0.25   | 145 | 80 |
| 53 | 152 Peter       | Hall      | PHALL   | 011.44.1344.47... | 20-AUG-97 | SA REP   | 9000  | 0.25   | 145 | 80 |
| 54 | 153 Christopher | Olsen     | COL...  | 011.44.1344.49... | 30-MAR-98 | SA REP   | 8000  | 0.2    | 145 | 80 |
| 55 | 154 Nanette     | Cambrault | NCA...  | 011.44.1344.98... | 09-DEC-98 | SA REP   | 7500  | 0.2    | 145 | 80 |
| 56 | 155 Oliver      | Tuvault   | OTU...  | 011.44.1344.48... | 23-NOV-99 | SA REP   | 7000  | 0.15   | 145 | 80 |
| 57 | 156 Janette     | King      | JKING   | 011.44.1345.42... | 30-JAN-96 | SA REP   | 10000 | 0.35   | 146 | 80 |
| 58 | 157 Patrick     | Sully     | PSU...  | 011.44.1345.92... | 04-MAR-96 | SA REP   | 9500  | 0.35   | 146 | 80 |
| 59 | 158 Allan       | McEwen    | AMC...  | 011.44.1345.82... | 01-AUG-96 | SA REP   | 9000  | 0.35   | 146 | 80 |
| 60 | 159 Lindsey     | Smith     | LSMI... | 011.44.1345.72... | 10-MAR-97 | SA REP   | 8000  | 0.3    | 146 | 80 |
| 61 | 160 Louise      | Doran     | LDO...  | 011.44.1345.62... | 15-DEC-97 | SA REP   | 7500  | 0.3    | 146 | 80 |
| 62 | 161 Sarath      | Sewall    | SSE...  | 011.44.1345.52... | 03-NOV-98 | SA REP   | 7000  | 0.25   | 146 | 80 |
| 63 | 162 Clara       | Vishney   | CVIS... | 011.44.1346.12... | 11-NOV-97 | SA REP   | 10500 | 0.25   | 147 | 80 |
| 64 | 163 Danielle    | Greene    | DGR...  | 011.44.1346.22... | 19-MAR-99 | SA REP   | 9500  | 0.15   | 147 | 80 |
| 65 | 164 Mattea      | Marvins   | MMA...  | 011.44.1346.32... | 24-JAN-00 | SA REP   | 7200  | 0.1    | 147 | 80 |
| 66 | 165 David       | Lee       | DLEE    | 011.44.1346.52... | 23-FEB-00 | SA REP   | 6800  | 0.1    | 147 | 80 |
| 67 | 166 Sundar      | Ande      | SAN...  | 011.44.1346.62... | 24-MAR-00 | SA REP   | 6400  | 0.1    | 147 | 80 |
| 68 | 167 Amit        | Banda     | ABA...  | 011.44.1346.72... | 21-APR-00 | SA REP   | 6200  | 0.1    | 147 | 80 |
| 69 | 168 Lisa        | Ozer      | LOZER   | 011.44.1343.92... | 11-MAR-97 | SA REP   | 11500 | 0.25   | 148 | 80 |
| 70 | 169 Harrison    | Bloom     | HBL...  | 011.44.1343.82... | 23-MAR-98 | SA REP   | 10000 | 0.2    | 148 | 80 |
| 71 | 170 Tayler      | Fox       | TFOX    | 011.44.1343.72... | 24-JAN-98 | SA REP   | 9600  | 0.2    | 148 | 80 |
| 72 | 171 William     | Smith     | WSM...  | 011.44.1343.62... | 23-FEB-99 | SA REP   | 7400  | 0.15   | 148 | 80 |
| 73 | 172 Elizabeth   | Bates     | EBA...  | 011.44.1343.52... | 24-MAR-99 | SA REP   | 7300  | 0.15   | 148 | 80 |
| 74 | 173 Sundita     | Kumar     | SKU...  | 011.44.1343.32... | 21-APR-00 | SA REP   | 6100  | 0.1    | 148 | 80 |

## Descripciones de la Tabla Human Resources (HR) (continuación)

### Employees (continuación)

|     |               |            |          |                   |           |            |       |        |     |        |
|-----|---------------|------------|----------|-------------------|-----------|------------|-------|--------|-----|--------|
| 75  | 174 Ellen     | Abel       | EABEL    | 011.44.1644.42... | 11-MAY-96 | SA_REP     | 11000 | 0.3    | 149 | 80     |
| 76  | 175 Alyssa    | Hutton     | AHUTT    | 011.44.1644.42... | 19-MAR-97 | SA_REP     | 8800  | 0.25   | 149 | 80     |
| 77  | 176 Jonathon  | Taylor     | JTA...   | 011.44.1644.42... | 24-MAR-98 | SA_REP     | 8600  | 0.2    | 149 | 80     |
| 78  | 177 Jack      | Livingston | JLIVI... | 011.44.1644.42... | 23-APR-98 | SA_REP     | 8400  | 0.2    | 149 | 80     |
| 79  | 178 Kimberely | Grant      | KGR...   | 011.44.1644.42... | 24-MAY-99 | SA_REP     | 7000  | 0.15   | 149 | (null) |
| 80  | 179 Charles   | Johnson    | CJO...   | 011.44.1644.42... | 04-JAN-00 | SA_REP     | 6200  | 0.1    | 149 | 80     |
| 81  | 180 Winston   | Taylor     | WT...    | 650.507.9876      | 24-JAN-98 | SH_CLERK   | 3200  | (null) | 120 | 50     |
| 82  | 181 Jean      | Fleaur     | JFLE...  | 650.507.9877      | 23-FEB-98 | SH_CLERK   | 3100  | (null) | 120 | 50     |
| 83  | 182 Martha    | Sullivan   | MSU...   | 650.507.9878      | 21-JUN-99 | SH_CLERK   | 2500  | (null) | 120 | 50     |
| 84  | 183 Girard    | Geoni      | GGE...   | 650.507.9879      | 03-FEB-00 | SH_CLERK   | 2800  | (null) | 120 | 50     |
| 85  | 184 Nandita   | Sarchand   | NSA...   | 650.509.1876      | 27-JAN-96 | SH_CLERK   | 4200  | (null) | 121 | 50     |
| 86  | 185 Alexis    | Bull       | ABULL    | 650.509.2876      | 20-FEB-97 | SH_CLERK   | 4100  | (null) | 121 | 50     |
| 87  | 186 Julia     | Dellinger  | JDEL...  | 650.509.3876      | 24-JUN-98 | SH_CLERK   | 3400  | (null) | 121 | 50     |
| 88  | 187 Anthony   | Cabrio     | ACA...   | 650.509.4876      | 07-FEB-99 | SH_CLERK   | 3000  | (null) | 121 | 50     |
| 89  | 188 Kelly     | Chung      | KCH...   | 650.505.1876      | 14-JUN-97 | SH_CLERK   | 3800  | (null) | 122 | 50     |
| 90  | 189 Jennifer  | Dilly      | JDILLY   | 650.505.2876      | 13-AUG-97 | SH_CLERK   | 3600  | (null) | 122 | 50     |
| 91  | 190 Timothy   | Gates      | TGA...   | 650.505.3876      | 11-JUL-98 | SH_CLERK   | 2900  | (null) | 122 | 50     |
| 92  | 191 Randall   | Perkins    | RPE...   | 650.505.4876      | 19-DEC-99 | SH_CLERK   | 2500  | (null) | 122 | 50     |
| 93  | 192 Sarah     | Bell       | SBELL    | 650.501.1876      | 04-FEB-96 | SH_CLERK   | 4000  | (null) | 123 | 50     |
| 94  | 193 Britney   | Everett    | BEV...   | 650.501.2876      | 03-MAR-97 | SH_CLERK   | 3900  | (null) | 123 | 50     |
| 95  | 194 Samuel    | McCain     | SMC...   | 650.501.3876      | 01-JUL-98 | SH_CLERK   | 3200  | (null) | 123 | 50     |
| 96  | 195 Vance     | Jones      | VJO...   | 650.501.4876      | 17-MAR-99 | SH_CLERK   | 2800  | (null) | 123 | 50     |
| 97  | 196 Alana     | Walsh      | AW...    | 650.507.9811      | 24-APR-98 | SH_CLERK   | 3100  | (null) | 124 | 50     |
| 98  | 197 Kevin     | Feehey     | KFEE...  | 650.507.9822      | 23-MAY-98 | SH_CLERK   | 3000  | (null) | 124 | 50     |
| 99  | 198 Donald    | O'Connell  | DOC...   | 650.507.9833      | 21-JUN-99 | SH_CLERK   | 2600  | (null) | 124 | 50     |
| 100 | 199 Douglas   | Grant      | DGR...   | 650.507.9844      | 13-JAN-00 | SH_CLERK   | 2600  | (null) | 124 | 50     |
| 101 | 200 Jennifer  | Whalen     | JWH...   | 515.123.4444      | 17-SEP-87 | AD_ASST    | 4400  | (null) | 101 | 10     |
| 102 | 201 Michael   | Hartstein  | MHA...   | 515.123.5555      | 17-FEB-96 | MK_MAN     | 13000 | (null) | 100 | 20     |
| 103 | 202 Pat       | Fay        | PFAY     | 603.123.6666      | 17-AUG-97 | MK_REP     | 6000  | (null) | 201 | 20     |
| 104 | 203 Susan     | Mavris     | SMA...   | 515.123.7777      | 07-JUN-94 | HR_REP     | 6500  | (null) | 101 | 40     |
| 105 | 204 Hermann   | Baer       | HBA...   | 515.123.8888      | 07-JUN-94 | PR_REP     | 10000 | (null) | 101 | 70     |
| 106 | 205 Shelley   | Higgins    | SHIG...  | 515.123.8080      | 07-JUN-94 | AC_MGR     | 12000 | (null) | 101 | 110    |
| 107 | 206 William   | Gietz      | WGI...   | 515.123.8181      | 07-JUN-94 | AC_ACCOUNT | 8300  | (null) | 205 | 110    |

## Descripciones de la Tabla Human Resources (HR) (continuación)

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_DATE    | END_DATE  | JOB_ID     | DEPARTMENT_ID |
|-------------|---------------|-----------|------------|---------------|
| 1           | 102 13-JAN-93 | 24-JUL-98 | IT_PROG    | 60            |
| 2           | 101 21-SEP-89 | 27-OCT-93 | AC_ACCOUNT | 110           |
| 3           | 101 28-OCT-93 | 15-MAR-97 | AC_MGR     | 110           |
| 4           | 201 17-FEB-96 | 19-DEC-99 | MK_REP     | 20            |
| 5           | 114 24-MAR-98 | 31-DEC-99 | ST_CLERK   | 50            |
| 6           | 122 01-JAN-99 | 31-DEC-99 | ST_CLERK   | 50            |
| 7           | 200 17-SEP-87 | 17-JUN-93 | AD_ASST    | 90            |
| 8           | 176 24-MAR-98 | 31-DEC-98 | SA_REP     | 80            |
| 9           | 176 01-JAN-99 | 31-DEC-99 | SA_MAN     | 80            |
| 10          | 200 01-JUL-94 | 31-DEC-98 | AC_ACCOUNT | 90            |

## Descripciones de la Tabla Human Resources (HR) (continuación)

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 |
|--------------|---------------------------------|------------|------------|
| 1 AD_PRES    | President                       | 20000      | 40000      |
| 2 AD_VP      | Administration Vice President   | 15000      | 30000      |
| 3 AD_ASST    | Administration Assistant        | 3000       | 6000       |
| 4 FI_MGR     | Finance Manager                 | 8200       | 16000      |
| 5 FI_ACCOUNT | Accountant                      | 4200       | 9000       |
| 6 AC_MGR     | Accounting Manager              | 8200       | 16000      |
| 7 AC_ACCOUNT | Public Accountant               | 4200       | 9000       |
| 8 SA_MAN     | Sales Manager                   | 10000      | 20000      |
| 9 SA_REP     | Sales Representative            | 6000       | 12000      |
| 10 PU_MAN    | Purchasing Manager              | 8000       | 15000      |
| 11 PU_CLERK  | Purchasing Clerk                | 2500       | 5500       |
| 12 ST_MAN    | Stock Manager                   | 5500       | 8500       |
| 13 ST_CLERK  | Stock Clerk                     | 2000       | 5000       |
| 14 SH_CLERK  | Shipping Clerk                  | 2500       | 5500       |
| 15 IT_PROG   | Programmer                      | 4000       | 10000      |
| 16 MK_MAN    | Marketing Manager               | 9000       | 15000      |
| 17 MK_REP    | Marketing Representative        | 4000       | 9000       |
| 18 HR_REP    | Human Resources Representative  | 4000       | 9000       |
| 19 PR_REP    | Public Relations Representative | 4500       | 10500      |

## Descripciones de la Tabla Human Resources (HR) (continuación)

DESCRIBE locations

| 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 locations

| LOCATION_ID | STREET_ADDRESS                                | POSTAL_CODE | CITY                | STATE_PROVINCE    | COUNTRY_ID |
|-------------|-----------------------------------------------|-------------|---------------------|-------------------|------------|
| 1           | 1000 1297 Via Cola di Rie                     | 00989       | Roma                | (null)            | IT         |
| 2           | 1100 93091 Calle della Testa                  | 10934       | Venice              | (null)            | IT         |
| 3           | 1200 2017 Shinjuku-ku                         | 1689        | Tokyo               | Tokyo Prefecture  | JP         |
| 4           | 1300 9450 Kamiya-cho                          | 6823        | Hiroshima           | (null)            | JP         |
| 5           | 1400 2014 Jabberwocky Rd                      | 26192       | Southlake           | Texas             | US         |
| 6           | 1500 2011 Interiors Blvd                      | 99236       | South San Francisco | California        | US         |
| 7           | 1600 2007 Zagora St                           | 50090       | South Brunswick     | New Jersey        | US         |
| 8           | 1700 2004 Charade Rd                          | 98199       | Seattle             | Washington        | US         |
| 9           | 1800 147 Spadina Ave                          | M5V 2L7     | Toronto             | Ontario           | CA         |
| 10          | 1900 6092 Boxwood St                          | Y5W 9T2     | Whitehorse          | Yukon             | CA         |
| 11          | 2000 40-5-12 Laolianggen                      | 190518      | Beijing             | (null)            | CN         |
| 12          | 2100 1298 Vileparle (E)                       | 490231      | Bombay              | Maharashtra       | IN         |
| 13          | 2200 12-98 Victoria Street                    | 2901        | Sydney              | New South Wales   | AU         |
| 14          | 2300 198 Clementi North                       | 540198      | Singapore           | (null)            | SG         |
| 15          | 2400 8204 Arthur St                           | (null)      | London              | (null)            | UK         |
| 16          | 2500 Magdalen Centre, The Oxford Science Park | OX9 9ZB     | Oxford              | Oxford            | UK         |
| 17          | 2600 9702 Chester Road                        | 09629850293 | Stretford           | Manchester        | UK         |
| 18          | 2700 Schwanthalerstr. 7031                    | 80925       | Munich              | Bavaria           | DE         |
| 19          | 2800 Rua Frei Caneca 1360                     | 01307-002   | Sao Paulo           | Sao Paulo         | BR         |
| 20          | 2900 20 Rue des Corps-Saints                  | 1730        | Geneva              | Geneve            | CH         |
| 21          | 3000 Murtenstrasse 921                        | 3095        | Bern                | BE                | CH         |
| 22          | 3100 Pieter Breughelstraat 837                | 3029SK      | Utrecht             | Utrecht           | NL         |
| 23          | 3200 Mariano Escobedo 9991                    | 11932       | Mexico City         | Distrito Federal, | MX         |

**Descripciones de la Tabla Human Resources (HR) (continuación)**

DESCRIBE regions

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

SELECT \* FROM locations

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

# C

## Uso de SQL Developer

ORACLE

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

## Objetivos

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

- Mostrar las funciones clave de Oracle SQL Developer
- Identificar las opciones de menú de Oracle SQL Developer
- Crear una conexión a la base de datos
- Gestionar objetos de bases de datos
- Utilizar la hoja de trabajo de SQL
- Guardar y ejecutar scripts SQL
- Crear y guardar informes



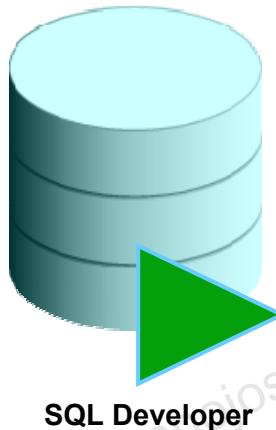
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

### Objetivos

En este apéndice, se presentará la herramienta gráfica denominada SQL Developer. Aprenderá cómo utilizar SQL Developer para las tareas de desarrollo de la base de datos. Aprenderá cómo utilizar la hoja de trabajo de SQL para ejecutar sentencias y scripts SQL.

## ¿Qué Es Oracle SQL Developer?

- Oracle SQL Developer es una herramienta gráfica que mejora la productividad y simplifica las tareas de desarrollo de la base de datos.
- Puede conectarse a cualquier esquema de base de datos de destino de Oracle mediante la autenticación estándar de la base de datos Oracle.



ORACLE

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

### ¿Qué Es Oracle SQL Developer?

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

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

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

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

La versión SQL Developer 1.2 está muy integrada con *Developer Migration Workbench*, que proporciona a los usuarios un único punto para examinar los datos y objetos de la base de datos de bases de datos de terceros, así como para migrar desde estas bases de datos a Oracle. También puede conectar a esquemas de bases de datos de terceros seleccionadas (no Oracle) como MySQL, Microsoft SQL Server y Microsoft Access, así como ver los metadatos y los datos de esas bases de datos.

Además, SQL Developer incluye soporte para Oracle Application Express 3.0.1 (Oracle APEX).

## Especificaciones de SQL Developer

- Incluido con Oracle Database 11g Versión 2
- Desarrollado en Java
- Soporta plataformas Windows, Linux y Mac OS X
- Conectividad por defecto mediante el controlador Java Database Connectivity (JDBC) fino
- Se conecta a Oracle Database versión 9.2.0.1 y posteriores
- Se descarga de forma gratuita desde el siguiente enlace:
  - [http://www.oracle.com/technology/products/database/sql\\_developer/index.html](http://www.oracle.com/technology/products/database/sql_developer/index.html)



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

### Especificaciones de SQL Developer

Oracle SQL Developer 1.5 se incluye con Oracle Database 11g Versión 2. SQL Developer está desarrollado en Java, aprovechando Oracle JDeveloper IDE (entorno de desarrollo de integración). Por tanto, se trata de una herramienta entre plataformas. La herramienta se ejecuta en plataformas con el sistema operativo (SO) Windows, Linux y Mac.

La conectividad por defecto a la base de datos se realiza a través del controlador JDBC fino, por lo que no es necesario el directorio raíz de Oracle. SQL Developer no necesita un instalador y simplemente necesita descomprimir los archivos descargados. Con SQL Developer, los usuarios se pueden conectar a Oracle Database 9.2.0.1 y versiones posteriores y a todas las ediciones de Oracle Database, incluida Express Edition.

#### Nota

Para las versiones de Oracle Database anteriores a Oracle Database 11g Versión 2, tendrá que descargar e instalar SQL Developer. SQL Developer 1.5 se descarga de forma gratuita desde el siguiente enlace:

[http://www.oracle.com/technology/products/database/sql\\_developer/index.html](http://www.oracle.com/technology/products/database/sql_developer/index.html).

Para obtener instrucciones sobre cómo instalar SQL Developer, puede visitar el siguiente enlace:

[http://download.oracle.com/docs/cd/E12151\\_01/index.htm](http://download.oracle.com/docs/cd/E12151_01/index.htm)



## Interfaz de SQL Developer 1.5

La interfaz de SQL Developer 1.5 contiene tres separadores principales de navegación, de izquierda a derecha:

- **Separador Connections:** si utiliza este separador, puede examinar los objetos y usuarios de la base de datos a los que tiene acceso.
- **Separador Files:** este separador, que se identifica mediante el icono de carpeta Files, le permite acceder a archivos desde la máquina local sin tener que utilizar el menú File > Open.
- **Separador Reports:** este separador, que se identifica mediante el icono Reports, le permite ejecutar informes predefinidos o crear y agregar sus propios informes.

### Navegación General y Uso

SQL Developer utiliza la parte izquierda de la navegación para buscar y seleccionar objetos y la parte derecha para mostrar información sobre los objetos seleccionados. También puede personalizar muchos aspectos de la apariencia y del comportamiento de SQL Developer mediante la definición de preferencias.

**Nota:** debe definir al menos una conexión para poder conectar a un esquema de base de datos y emitir consultas SQL o ejecutar procedimientos/funciones.

## Interfaz de SQL Developer 1.5 (continuación)

### Menús

Los siguientes menús contienen entradas estándar, además de entradas de funciones específicas de SQL Developer.

- **View:** contiene opciones que afectan a lo que se muestra en la interfaz de SQL Developer
- **Navigate:** contiene opciones para acceder a distintos paneles y para ejecutar subprogramas
- **Run:** contiene las opciones Run File y Execution Profile que son relevantes en la selección de una función o un procedimiento, así como en las opciones de depuración
- **Source:** contiene opciones para utilizarlas al editar funciones y procedimientos
- **Versioning:** proporciona soporte integrado para las siguientes versiones y sistemas de control de origen: sistema de versiones concurrentes (CVS) y subversión
- **Migración:** contiene opciones relacionadas con la migración de bases de datos de terceros a Oracle
- **Tools:** llama a las herramientas de SQL Developer como SQL\*Plus, Preferences y la hoja de trabajo de SQL

**Nota:** el menú Run también contiene opciones relevantes cuando se selecciona una función o procedimiento para su depuración. Se trata de las mismas opciones que se encuentran en el menú Debug de la versión 1.2.

## Creación de una Conexión a la Base Datos

- Debe tener al menos una conexión a la base de datos para utilizar SQL Developer.
- Puede crear y probar conexiones para varias:
  - Bases de Datos
  - Esquemas
- SQL Developer importa automáticamente las conexiones definidas en el archivo `tnsnames.ora` al sistema.
- Puede exportar conexiones a un archivo de lenguaje extensible de marcas (XML).
- Cada conexión a la base de datos adicional creada se muestra en la jerarquía de Connections Navigator.



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

### Creación de una Conexión a la Base Datos

Una conexión es un objeto de SQL Developer que especifica la información necesaria para conectarse a una base de datos concreta como usuario específico de dicha base de datos. Para utilizar SQL Developer, debe tener al menos una conexión a la base de datos que puede ser existente, creada o importada.

Puede crear y probar conexiones para varias bases de datos y esquemas.

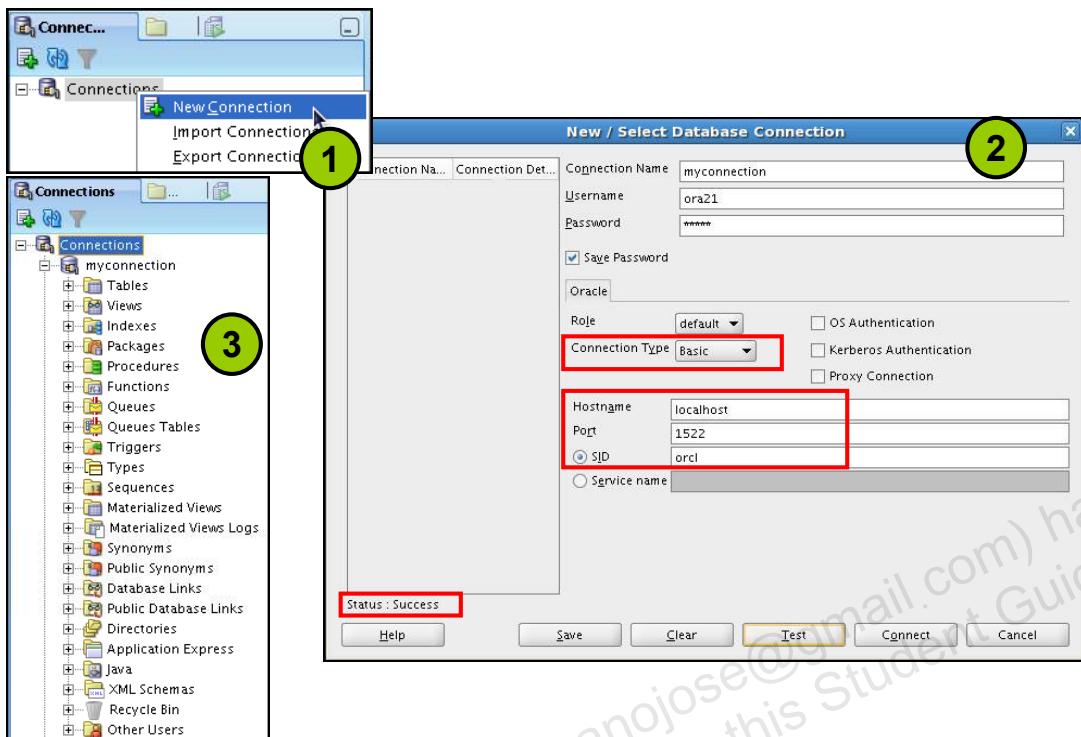
Por defecto, el archivo `tnsnames.ora` se encuentra en el directorio `$ORACLE_HOME/network/admin`, pero también puede estar en el directorio especificado por el valor de registro o la variable de entorno `TNS_ADMIN`. Al iniciar SQL Developer, cuando se muestra el cuadro de diálogo Database Connections, SQL Developer importa automáticamente las conexiones definidas en el archivo `tnsnames.ora` al sistema.

**Nota:** en los sistemas Windows, si existe el archivo `tnsnames.ora`, pero SQL Developer no está utilizando sus conexiones, defina `TNS_ADMIN` como una variable de entorno del sistema.

Puede exportar conexiones a un archivo XML para volver a utilizarlas más tarde.

Puede crear conexiones adicionales para conectarse a la misma base de datos, pero como usuarios diferentes o para conectarse a distintas bases de datos.

## Creación de una Conexión a la Base Datos



ORACLE

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

### Creación de una Conexión a la Base Datos (continuación)

Para crear una conexión a la base de datos, realice los siguientes pasos:

1. En la página con separadores Connections, haga clic con el botón derecho en **Connections** y seleccione **New Connection**.
2. En la ventana New>Select Database Connection, introduzca el nombre de la conexión. Introduzca el nombre de usuario y la contraseña del esquema al que desea conectarse.
  - a) En el cuadro desplegable Role, puede seleccionar el valor por defecto o SYSDBA (seleccione SYSDBA para el usuario sys o cualquier usuario con privilegios de administrador de la base de datos).
  - b) También puede seleccionar el tipo de conexión como:
    - **Basic:** en este tipo, introduzca el nombre de host y el SID de la base de datos a la que desea conectarse. El puerto ya está definido en 1521. También puede optar por introducir el nombre de servicio directamente si utiliza una conexión de base de datos remota.
    - **TNS:** puede seleccionar cualquiera de los alias de base de datos importados del archivo `tnsnames.ora`.
    - **LDAP:** puede consultar los servicios de base de datos en Oracle Internet Directory, componente de Oracle Identity Management.
    - **Advanced:** puede definir una dirección URL de JDBC personalizada para conectarse a la base de datos.

## Creación de una Conexión a la Base Datos (continuación)

- a) Haga clic en Test para asegurarse de que la conexión se ha definido correctamente.
- b) Haga clic en Connect.

Si activa la casilla de control Save Password, la contraseña se guarda en un archivo XML.

De este modo, cuando cierre la conexión a SQL Developer y la vuelva a abrir, no se le pedirá la contraseña.

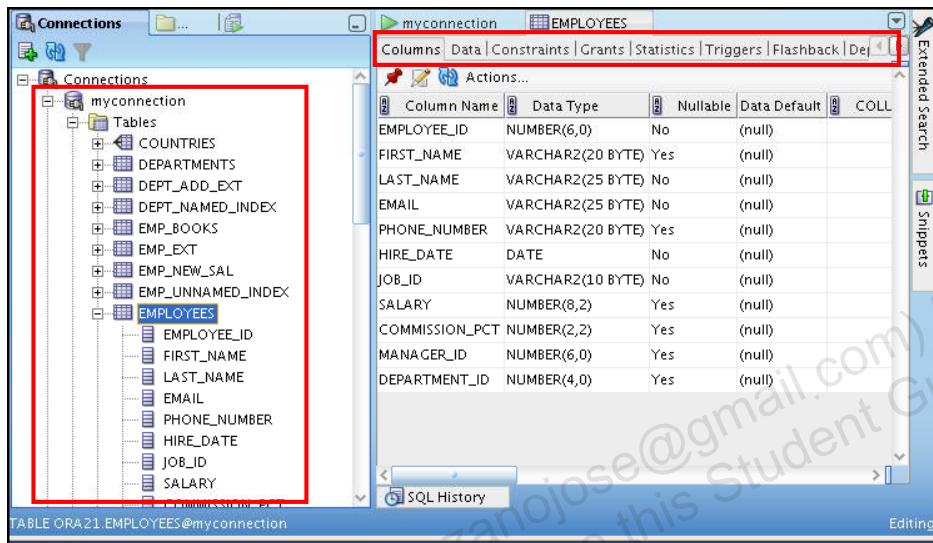
3. La conexión se agrega a Connections Navigator. Puede ampliar la conexión para ver los objetos de la base de datos y las definiciones de objetos, por ejemplo, dependencias, detalles, estadísticas, etc.

**Nota:** en la misma ventana New>Select Database Connection, puede definir las conexiones a orígenes de datos que no sean Oracle mediante los separadores Access, MySQL y SQL Server. Sin embargo, estas conexiones son conexiones de sólo lectura que le permiten examinar los objetos y datos de ese origen de datos.

## Exploración de Objetos de Bases de Datos

Utilice Connections Navigator para:

- Examinar los objetos de un esquema de base de datos
- Revisar las definiciones de objetos de forma rápida



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

**ORACLE**

## Exploración de Objetos de Bases de Datos

Una vez creada la conexión a la base de datos, puede utilizar Connections Navigator para examinar muchos objetos de un esquema de base de datos, entre los que se incluyen tablas, vistas, índices, paquetes, procedimientos, disparadores y tipos.

Puede ver la definición de los objetos desglosados en separadores de información que se transfieren al diccionario de datos. Por ejemplo, si selecciona una tabla en Navigator, se muestran los detalles sobre las columnas, restricciones, permisos, estadísticas, disparadores, etc. en una página con separadores fáciles de leer.

Si desea ver la definición de la tabla EMPLOYEES como se muestra en la diapositiva, realice los siguientes pasos:

1. Amplíe el nodo Connections de Connections Navigator.
2. Amplíe Tables.
3. Haga clic en EMPLOYEES. Por defecto, se selecciona el separador Columns. Muestra la descripción de la columna de la tabla. Mediante el separador Data, puede ver los datos de la tabla y también introducir nuevas filas, datos de actualización y confirmar estos cambios en la base de datos.

## Visualización de la Estructura de la Tabla

Utilice el comando DESCRIBE para mostrar la estructura de una tabla:

The screenshot shows the Oracle SQL Developer interface. At the top, there's a toolbar with various icons. Below it, a status bar displays "myconnection" and "1.69686902 seconds". The main area has a title bar "DESC EMPLOYEES". Below the title bar is a toolbar with buttons for Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. The main content area displays the results of the DESCRIBE command for the EMPLOYEES table. The output is a table with three columns: Name, Null, and Type. The columns are separated by dashed lines. The data rows are as follows:

| 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)    |
| 11 rows selected |          |              |

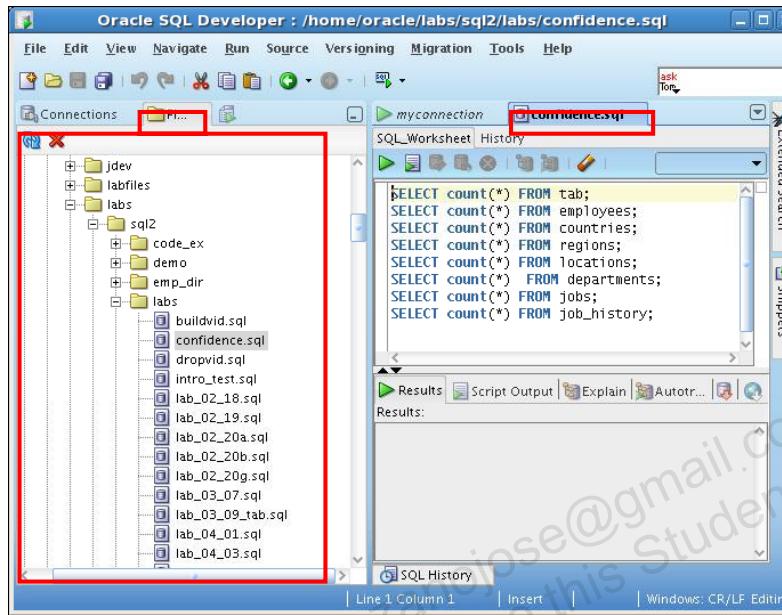
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

## Visualización de la Estructura de la Tabla

En SQL Developer, también puede mostrar la estructura de una tabla mediante el comando DESCRIBE. El resultado del comando es una visualización de los nombres de columna y tipos de dato, así como una indicación de si una columna debe contener datos.

## Examen de Archivos

Utilice File Navigator para explorar el sistema de archivos y abrir los archivos del sistema.



ORACLE

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

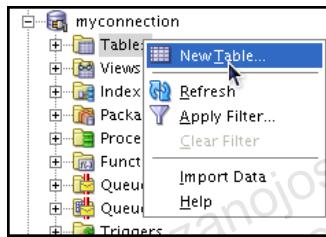
### Exploración de Objetos de Bases de Datos

Puede utilizar File Navigator para examinar y abrir los archivos del sistema.

- Para ver el navegador de archivos, haga clic en el separador Files o en View > Files.
- Para ver el contenido de un archivo, haga clic dos veces en el nombre de un archivo para ver su contenido en el área SQL worksheet.

## Creación de un Objeto de Esquema

- SQL Developer soporta la creación de cualquier objeto de esquema mediante:
  - Ejecución de una sentencia SQL en la hoja de trabajo de SQL
  - Uso del menú contextual
- Editar los objetos mediante un cuadro de diálogo de edición o con uno de los muchos menús sensibles al contexto.
- Ver el lenguaje de definición de datos (DDL) para los ajustes, como la creación de un nuevo objeto o la edición de un objeto de esquema existente.



ORACLE

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

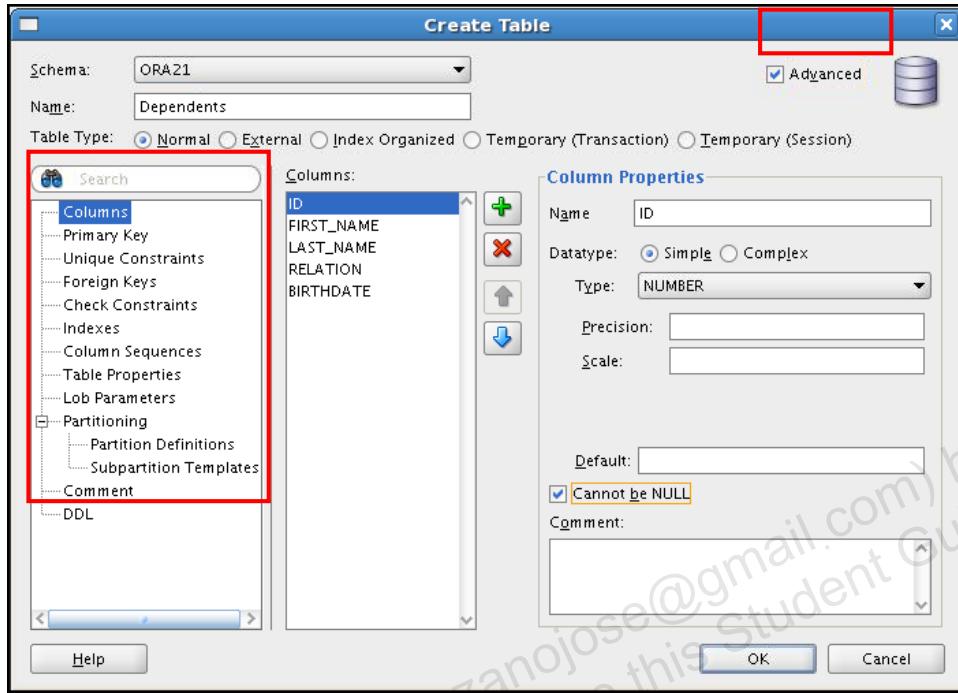
### Creación de un Objeto de Esquema

SQL Developer soporta la creación de cualquier objeto de esquema mediante la ejecución de una sentencia SQL en la hoja de trabajo de SQL. Por otro lado, puede crear objetos mediante los menús contextuales. Una vez creado, puede editar los objetos con un cuadro de diálogo de edición o con uno de los muchos menús sensibles al contexto.

Una vez que se han creado los nuevos objetos o se han editado los existentes, el DDL de estos ajustes estará disponible para la revisión. Una opción Export DDL está disponible si desea crear el DDL completo para uno o más objetos en el esquema.

La diapositiva muestra cómo crear una tabla mediante el menú contextual. Para abrir un cuadro de diálogo para la creación de una nueva tabla, haga clic con el botón derecho en Tables y seleccione New Table. Los cuadros de diálogo para la creación y edición de objetos de la base de datos tienen varios separadores, cada uno de ellos contiene un agrupamiento lógico de propiedades para dicho tipo de objeto.

## Creación de una Nueva Tabla: Ejemplo



ORACLE

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

### Creación de una Nueva Tabla: Ejemplo

En el cuadro de diálogo Create Table, si no activa la casilla de control Advanced, puede crear una tabla de forma rápida especificando columnas y algunas de las funciones más utilizadas.

Si activa la casilla de control Advanced, el cuadro de diálogo Create Table cambia a otro con varias opciones, en el que puede especificar un juego ampliado de funciones mientras crea la tabla.

En el ejemplo de la diapositiva se muestra cómo crear la tabla DEPENDENTS activando la casilla de control Advanced.

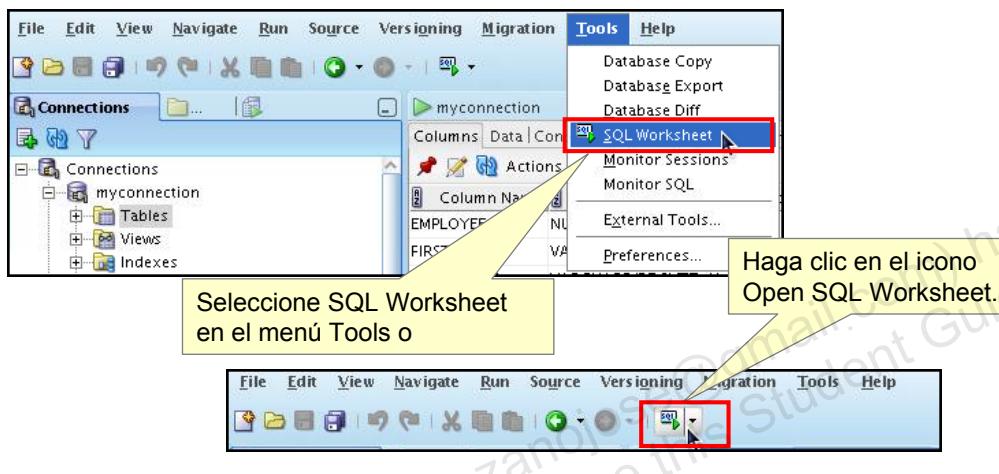
Para crear una nueva tabla, realice los siguientes pasos:

1. En Connections Navigator, haga clic con el botón derecho en Tables.
2. Seleccione Create TABLE.
3. En el cuadro de diálogo Create Table, seleccione Advanced.
4. Especifique la información de columna.
5. Haga clic en OK.

Aunque no es necesario, también debe especificar una clave primaria con el separador Primary Key en el cuadro de diálogo. En ocasiones, puede que desee editar la tabla que ha creado; para ello, haga clic con el botón derecho en la tabla Connections Navigator y seleccione Edit.

## Uso de la Hoja de Trabajo de SQL

- Utilice la hoja de trabajo de SQL para introducir y ejecutar sentencias SQL, PL/SQL y SQL \*Plus.
- Especifique las acciones que se pueden procesar mediante la conexión a la base de datos asociada a la hoja de trabajo.



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

**ORACLE**

### Uso de la Hoja de Trabajo de SQL

Al conectarse a una base de datos, automáticamente se abre una ventana de la hoja de trabajo de SQL para dicha conexión. Puede utilizar la hoja de trabajo de SQL para introducir y ejecutar sentencias SQL, PL/SQL y SQL \*Plus. La hoja de trabajo de SQL soporta sentencias SQL\*Plus hasta un determinado grado. Las sentencias SQL\*Plus no soportadas por la hoja de trabajo de SQL se ignoran y no se transfieren a la base de datos.

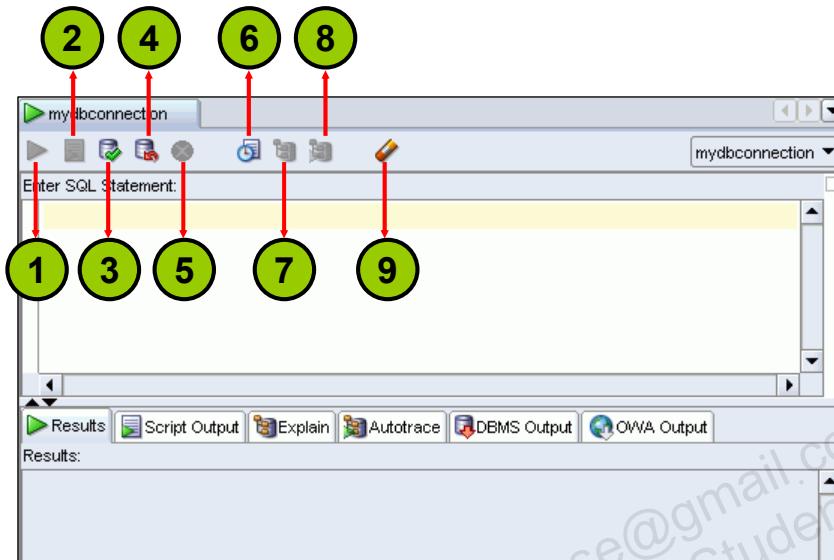
Puede especificar cualquier acción que se pueda procesar mediante la conexión a la base de datos asociada a la hoja de trabajo, como:

- Creación de una tabla
- Inserción de datos
- Creación y edición de un disparador
- Selección de datos de tablas
- Guardado de datos seleccionados en un archivo

Para mostrar una hoja de trabajo de SQL, utilice uno de estos métodos:

- Seleccione Tools > SQL Worksheet.
- Haga clic en el ícono Open SQL Worksheet.

## Uso de la Hoja de Trabajo de SQL



ORACLE

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

### Uso de la Hoja de Trabajo de SQL (continuación)

Puede que desee utilizar teclas o iconos de acceso directo para realizar determinadas tareas como la ejecución de una sentencia SQL o de un script y la visualización del historial de sentencias SQL ejecutadas. Puede utilizar la barra de herramientas de SQL Worksheet que contiene iconos para realizar las siguientes tareas:

1. **Execute Statement:** ejecuta la sentencia donde se encuentra el cursor del cuadro Enter SQL Statement. Puede utilizar variables de enlace en las sentencias SQL pero no variables de sustitución.
2. **Run Script:** ejecuta todas las sentencias en el cuadro Enter SQL Statement mediante Script Runner. Puede utilizar variables de sustitución en las sentencias SQL, pero no variables de enlace.
3. **Commit:** escribe cualquier cambio realizado en la base de datos y finaliza la transacción.
4. **Rollback:** desecha cualquier cambio realizado en la base de datos, sin escribirlos en la base de datos y finaliza la transacción.
5. **Cancel:** para la ejecución de cualquier sentencia que se esté ejecutando actualmente.
6. **SQL History:** muestra un cuadro de diálogo con información sobre las sentencias SQL ejecutadas.
7. **Execute Explain Plan:** genera el plan de ejecución, que puede ver haciendo clic en el separador Explain.
8. **Autotrace:** genera información de rastreo para la sentencia.
9. **Clear:** borra la sentencia o sentencias del cuadro Enter SQL Statement.

## Uso de la Hoja de Trabajo de SQL

- Utilice la hoja de trabajo de SQL para introducir y ejecutar sentencias SQL, PL/SQL y SQL \*Plus.
- Especifique las acciones que se pueden procesar mediante la conexión a la base de datos asociada a la hoja de trabajo.



### Uso de la Hoja de Trabajo de SQL (continuación)

Al conectarse a una base de datos, automáticamente se abre una ventana de la hoja de trabajo de SQL para dicha conexión. Puede utilizar la hoja de trabajo de SQL para introducir y ejecutar sentencias SQL, PL/SQL y SQL \*Plus. Todos los comandos SQL y PL/SQL están soportados, ya que se transfieren directamente desde la hoja de trabajo de SQL hasta Oracle Database. La hoja de trabajo de SQL tiene que interpretar los comandos SQL\*Plus utilizados en SQL Developer antes de transferirlos a la base de datos.

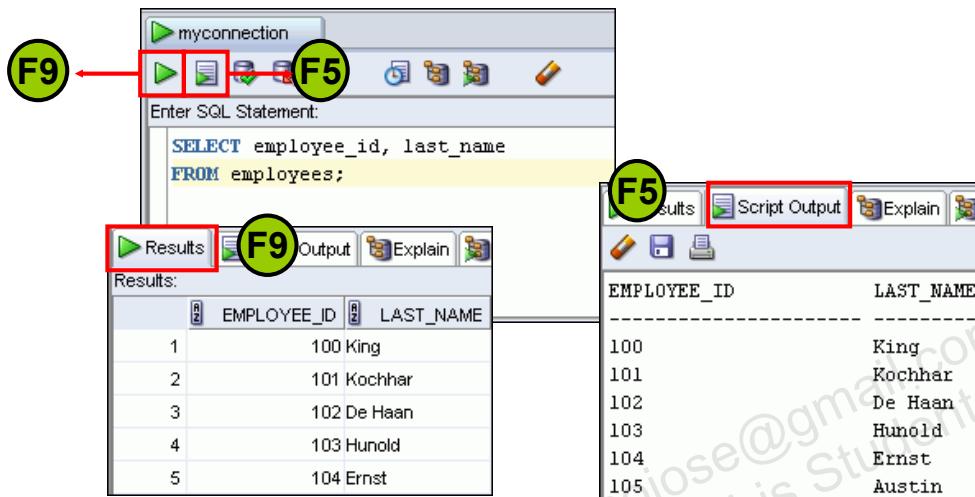
La hoja de trabajo de SQL soporta actualmente una serie de comandos de SQL\*Plus. Los comandos no soportados por la hoja de trabajo de SQL se ignoran y no se envían a la base de datos Oracle. Mediante la hoja de trabajo de SQL, puede ejecutar sentencias SQL y algunos comandos de SQL\*Plus.

Puede mostrar una hoja de trabajo de SQL mediante alguna de las siguientes dos opciones:

- Seleccione Tools > SQL Worksheet.
- Haga clic en el ícono Open SQL Worksheet.

## Ejecución de Sentencias SQL

Utilice el cuadro Enter SQL Statement para introducir una o varias sentencias SQL.

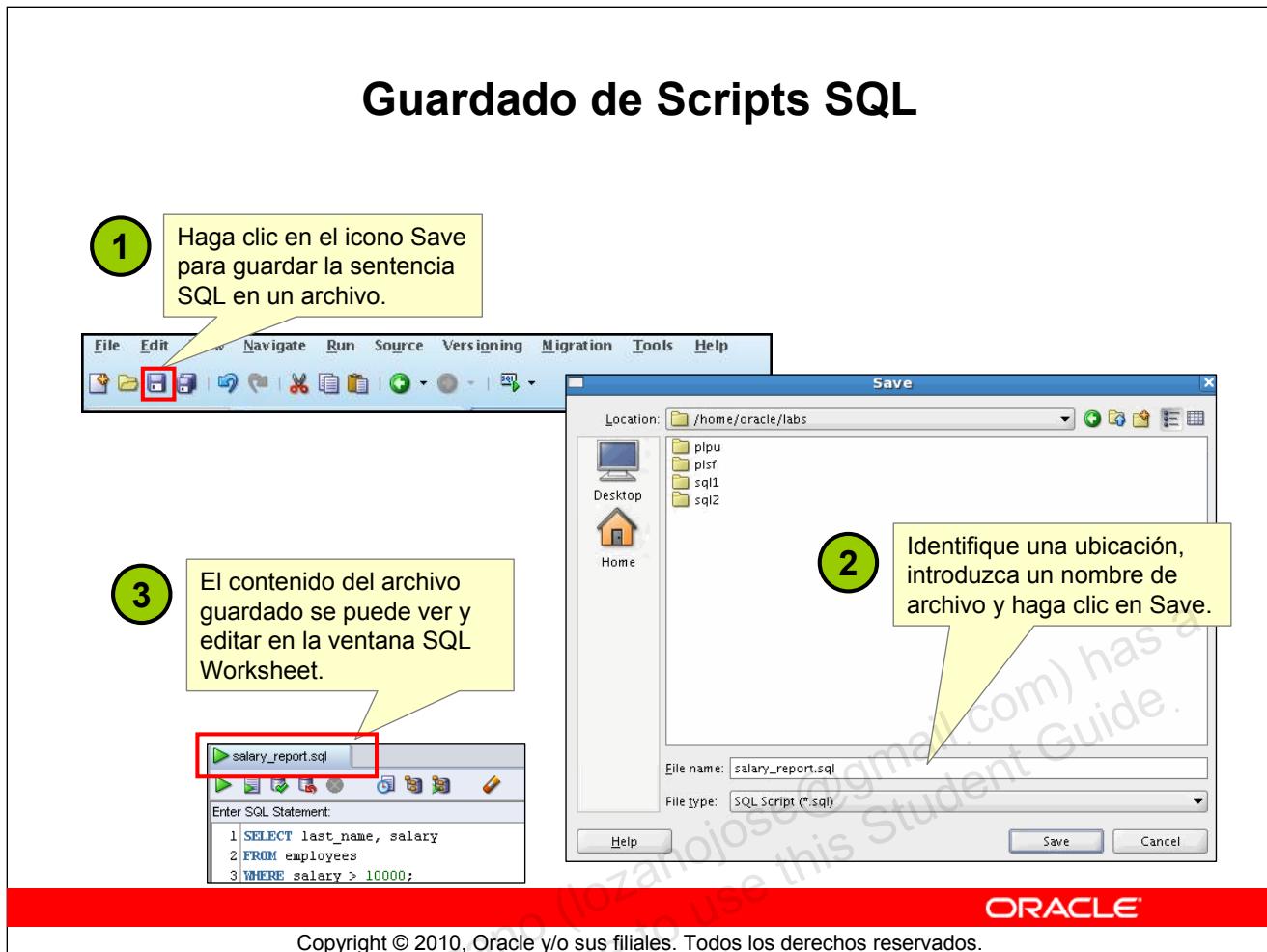


ORACLE

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

### Ejecución de Sentencias SQL

En el ejemplo de la diapositiva se muestra la diferencia de la salida de la misma consulta cuando se utiliza la tecla F9 o Execute Statement, frente a la salida cuando se utiliza F5 o Run Script.



## Guardado de scripts SQL

Puede guardar las sentencias SQL desde la hoja de trabajo de SQL en un archivo de texto. Para guardar el contenido del cuadro Enter SQL Statement, siga estos pasos:

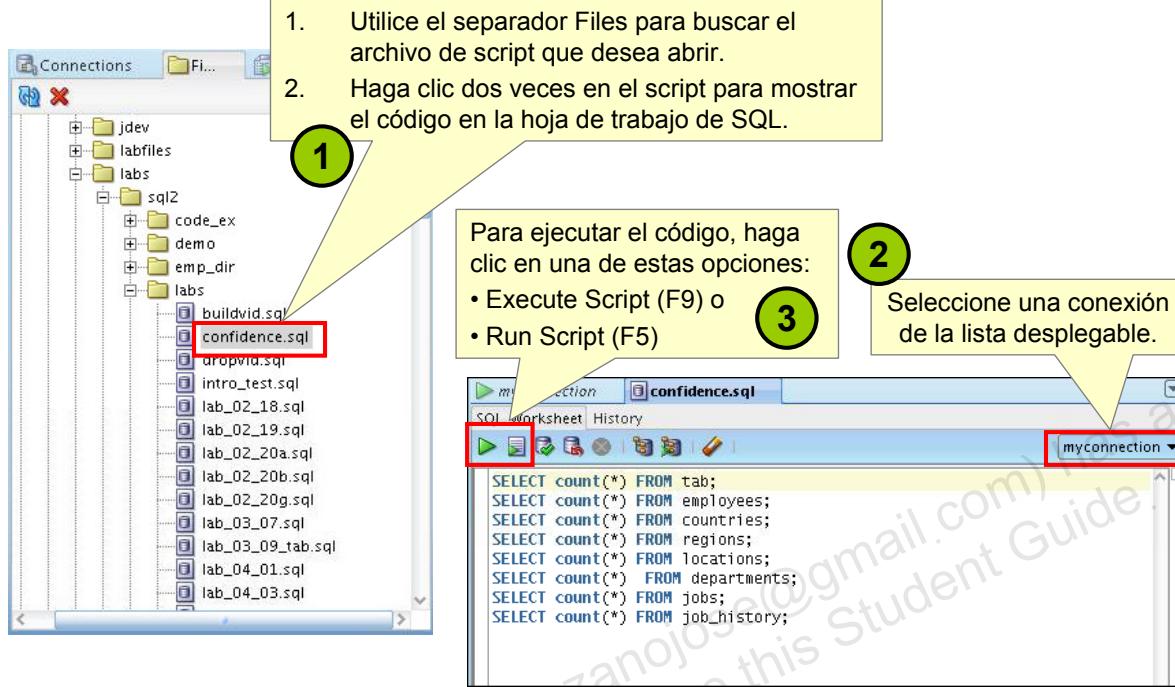
1. Haga clic en el icono Save o utilice el elemento de menú File > Save.
2. En el cuadro de diálogo Windows Save, introduzca un nombre de archivo y la ubicación donde desea guardar el archivo.
3. Haga clic en Save.

Una vez guardado el contenido en un archivo, la ventana Enter SQL Statement muestra una página con separadores con el contenido del archivo. Puede tener varios archivos abiertos a la vez. Cada archivo muestra una página con separadores.

### Rutas de Acceso de los Scripts

Puede seleccionar una ruta de acceso por defecto para buscar los scripts y guardarlos. En Tools > Preferences > Database > Worksheet Parameters, introduzca un valor en el campo “Select default path to look for scripts”.

## Ejecución de Archivos de Script Guardados: Método 1



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

### Ejecución de Archivos de Script Guardados: Método 1

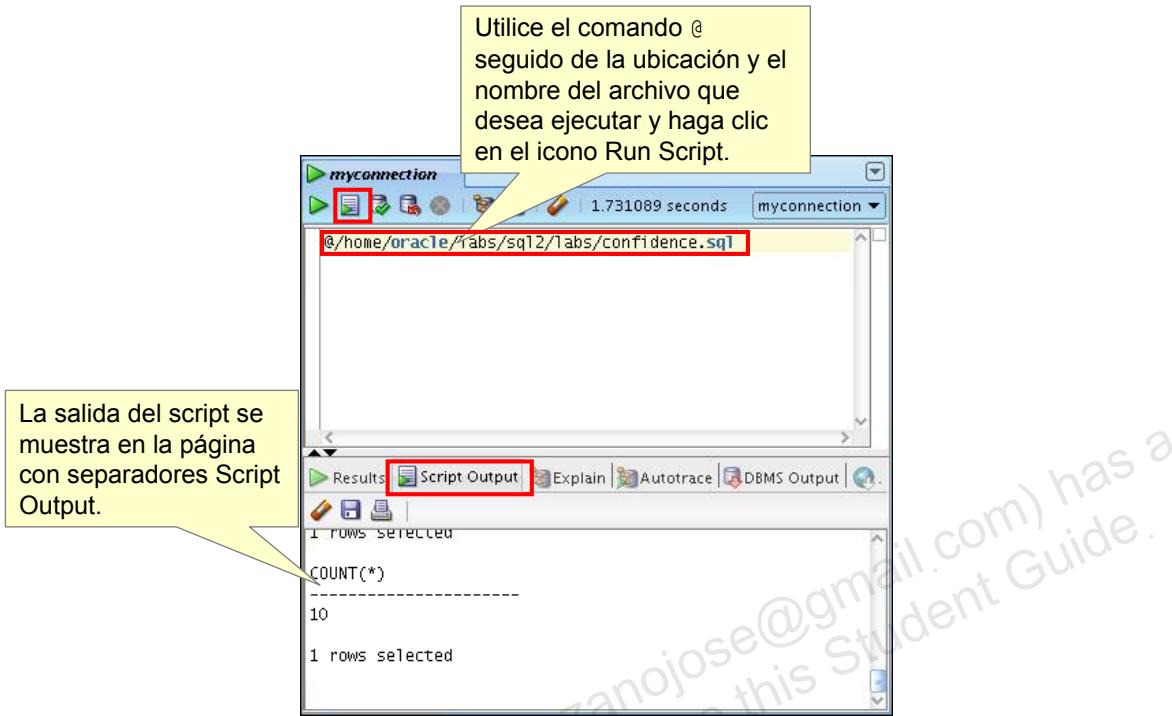
Para abrir un archivo de script y mostrar el código en el área SQL Worksheet, realice lo siguiente:

- En el navegador de archivos, seleccione (o acceda al) archivo de script que desea abrir.
- Haga clic dos veces para abrirlo. El código del archivo de script se muestra en el área SQL Worksheet.
- Seleccione una conexión de la lista desplegable Connection.
- Para ejecutar el código, haga clic en el ícono Run Script (F5) en la barra de herramientas de SQL Worksheet. Si no ha seleccionado una conexión de la lista desplegable Connection, aparecerá un cuadro de diálogo Connection. Seleccione la conexión que desea utilizar para ejecutar el script.

También puede:

- Seleccionar File > Open. Aparece un cuadro de diálogo Open.
- En el cuadro de diálogo Open, seleccionar (o acceder al) archivo de script que desea abrir.
- Hacer clic en Open. El código del archivo de script se muestra en el área SQL Worksheet.
- Seleccione una conexión de la lista desplegable Connection.
- Para ejecutar el código, haga clic en el ícono Run Script (F5) en la barra de herramientas de SQL Worksheet. Si no ha seleccionado una conexión de la lista desplegable Connection, aparecerá un cuadro de diálogo Connection. Seleccione la conexión que desea utilizar para ejecutar el script.

## Ejecución de Archivos de Script Guardados: Método 2



ORACLE

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

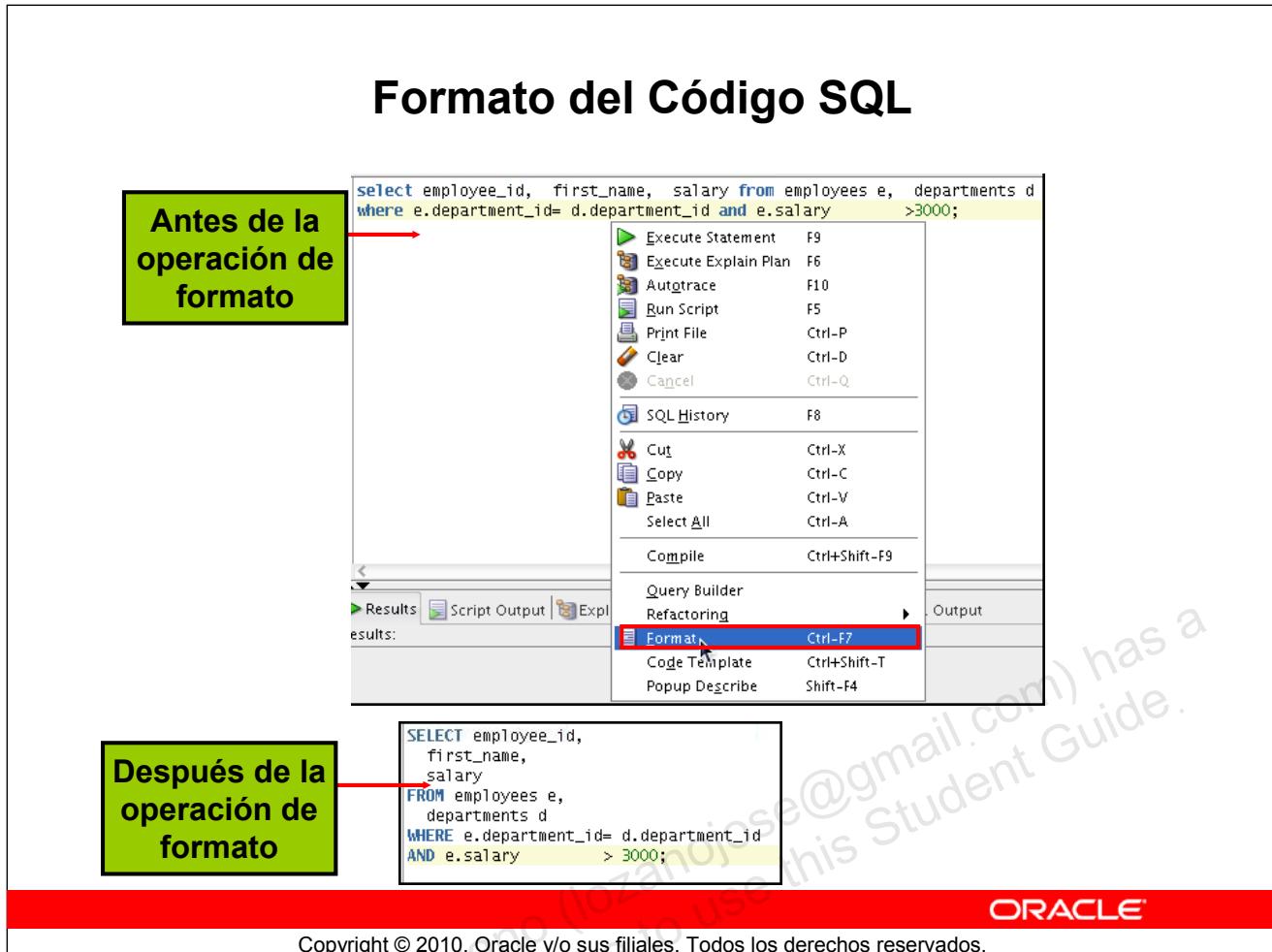
### Ejecución de Archivos de Script Guardados: Método 2

Para ejecutar un script SQL guardado, realice lo siguiente:

1. Utilice el comando @, seguido de la ubicación y el nombre del archivo que desea ejecutar, en la ventana Enter SQL Statement.
2. Haga clic en el ícono Run Script.

El resultado de la ejecución del archivo se muestra en la página con separadores Script Output.

También puede guardar la salida del script haciendo clic en el ícono Save de la página con separadores Script Output. Aparece el cuadro de diálogo Windows Save, donde puede identificar un nombre y una ubicación para el archivo.



## Formato del Código SQL

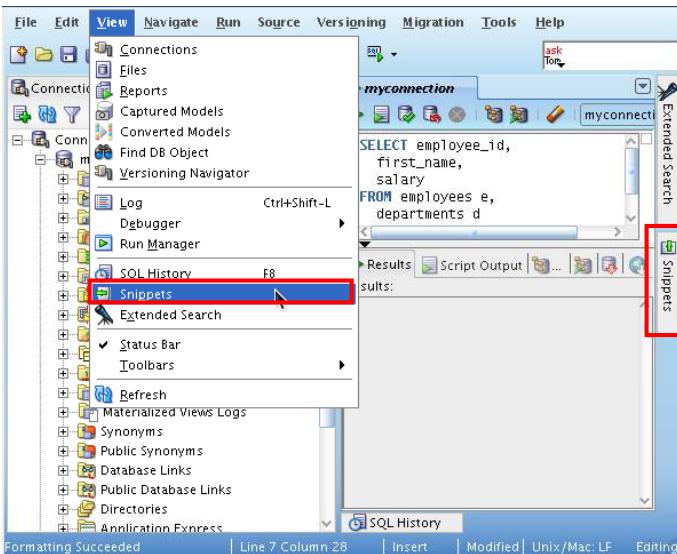
Puede que desee mejorar la apariencia del sangrado, espaciado, uso de mayúsculas y separación de líneas del código SQL. SQL Developer cuenta con una función para dar formato al código SQL.

Para dar formato al código SQL, haga clic con el botón derecho en el área de la sentencia y seleccione Format SQL.

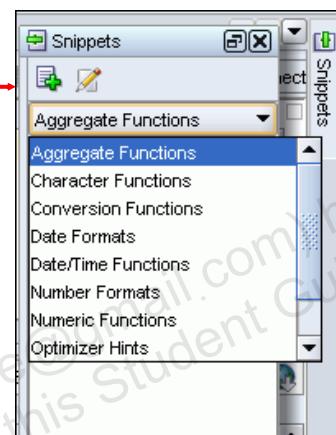
En el ejemplo de la diapositiva, antes de dar formato, las palabras clave del código SQL no están en mayúscula y la sentencia no está sangrada correctamente. Después de dar formato, el código SQL mejora la apariencia con las palabras clave en mayúscula y la sentencia sangrada correctamente.

## Uso de Fragmentos

Los fragmentos son fragmentos de código que puede ser simplemente sintaxis o ejemplos.



Al colocar el cursor aquí, aparece la ventana Snippets. En la lista desplegable, puede seleccionar la categoría de funciones que deseé.



ORACLE

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

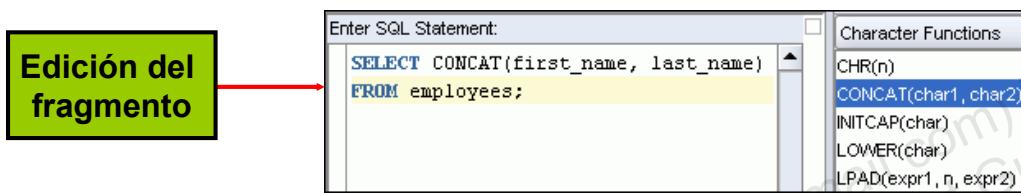
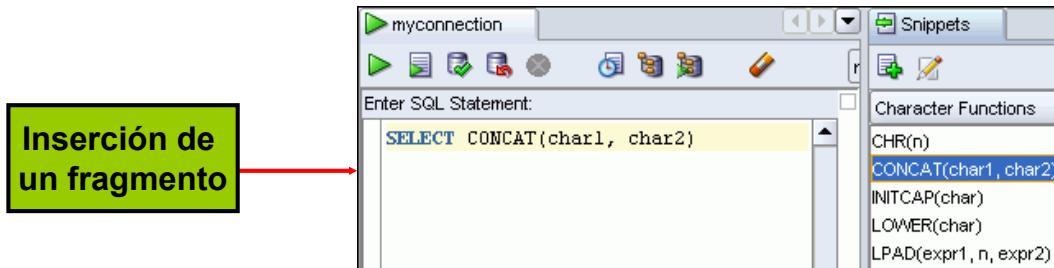
### Uso de Fragmentos

Es posible que desee utilizar determinados fragmentos de código al utilizar la hoja de trabajo de SQL o al crear o editar un procedimiento o función de PL/SQL. SQL Developer dispone de la función denominada Snippets. Los fragmentos son partes de código, como funciones SQL, indicaciones del optimizador y otras técnicas de programación de PL/SQL. Puede arrastrar los fragmentos en la ventana Editor.

Para visualizar los fragmentos, seleccione View > Snippets.

Aparece la ventana Snippets a la derecha. Puede utilizar la lista desplegable para seleccionar un grupo. En el margen de la ventana derecha se encuentra un botón Snippets, para poder acceder a la ventana Snippets si se oculta.

## Uso de Fragmentos: Ejemplo



ORACLE

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

### Uso de Fragmentos: Ejemplo

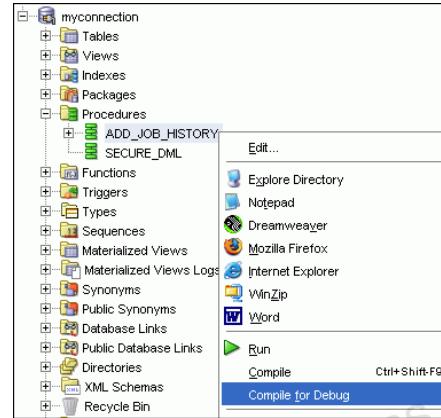
Para insertar un fragmento en el código en la hoja de trabajo de SQL o en una función o procedimiento, arrastre el fragmento desde la ventana Snippets y suéltelo en el lugar deseado del código. A continuación, puede editar la sintaxis para que la función SQL sea válida en el contexto actual. Para ver una descripción breve de una función SQL en una ayuda de burbuja, coloque el cursor sobre el nombre de la función.

En el ejemplo de la diapositiva se muestra que `CONCAT (char1, char2)` se arrastra desde el grupo Character Functions a la ventana Snippets. A continuación, se edita la sintaxis de la función `CONCAT` y el resto de la sentencia se agrega como se describe a continuación:

```
SELECT CONCAT(first_name, last_name)
FROM employees;
```

## Depuración de Procedimientos y Funciones

- Utilice SQL Developer para depurar funciones y procedimientos PL/SQL.
- Utilice la opción “Compile for Debug” para realizar una compilación PL/SQL para que se pueda depurar el procedimiento.
- Utilice las opciones del menú Debug para definir puntos de división y para ejecutar Step Into y Step Over.



ORACLE

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

### Depuración de Procedimientos y Funciones

En SQL Developer, puede depurar procedimientos y funciones PL/SQL. Con las opciones del menú Debug, puede realizar las siguientes tareas de depuración:

- **Find Execution Point** permite ir al siguiente punto de ejecución.
- **Resume** permite continuar la ejecución.
- **Step Over** permite omitir el siguiente método e ir a la siguiente sentencia después del método.
- **Step Into** permite ir a la primera sentencia del siguiente método.
- **Step Out** permite salir del método actual e ir a la siguiente sentencia.
- **Step to End of Method** permite ir a la última sentencia del método actual.
- **Pause** permite detener la ejecución, pero no salir, con lo que se permite reanudar la ejecución.
- **Terminate** permite detener y salir de la ejecución. No se puede reanudar la ejecución desde este punto; en su lugar, para iniciar la ejecución o depuración desde el principio de la función o procedimiento, haga clic en el icono Run o Debug de la barra de herramientas con separadores Source.
- **Garbage Collection** permite eliminar objetos no válidos de la caché en favor de objetos a los que se acceda más frecuentemente y más válidos.

Estas opciones también están disponibles como iconos en la barra de herramientas de depuración.

## Informes de Bases de Datos

SQL Developer proporciona un número predefinido de informes sobre la base de datos y sus objetos.

The screenshot shows the SQL Developer interface. On the left, the 'Connections' sidebar has a red box around the 'Reports' icon. The main pane displays a 'Dependencies' report titled 'myconnection'. The report table lists various database objects and their dependencies:

| Owner  | Name                     | Type    | Referenced Owner | Referenced Name          |
|--------|--------------------------|---------|------------------|--------------------------|
| CTXSYS | CTX_CLASSES              | VIEW    | CTXSYS           | DR\$CLASS                |
| CTXSYS | CTX_CLS                  | PACKAGE | SYS              | STANDARD                 |
| CTXSYS | CTX_DOC                  | PACKAGE | SYS              | STANDARD                 |
| CTXSYS | CTX_INDEX_SETS           | VIEW    | CTXSYS           | DR\$INDEX_SET            |
| CTXSYS | CTX_INDEX_SETS           | VIEW    | SYS              | USER\$                   |
| CTXSYS | CTX_INDEX_SET_INDEXES    | VIEW    | CTXSYS           | DR\$INDEX_SET            |
| CTXSYS | CTX_INDEX_SET_INDEXES    | VIEW    | CTXSYS           | DR\$INDEX_SET_INDEX      |
| CTXSYS | CTX_INDEX_SET_INDEXES    | VIEW    | SYS              | USER\$                   |
| CTXSYS | CTX_OBJECTS              | VIEW    | CTXSYS           | DR\$CLASS                |
| CTXSYS | CTX_OBJECTS              | VIEW    | CTXSYS           | DR\$OBJECT               |
| CTXSYS | CTX_OBJECT_ATTRIBUTES    | VIEW    | CTXSYS           | DR\$CLASS                |
| CTXSYS | CTX_OBJECT_ATTRIBUTES    | VIEW    | CTXSYS           | DR\$OBJECT               |
| CTXSYS | CTX_OBJECT_ATTRIBUTES    | VIEW    | CTXSYS           | DR\$OBJECT_ATTRIBUTE     |
| CTXSYS | CTX_OBJECT_ATTRIBUTE_LOV | VIEW    | CTXSYS           | DR\$CLASS                |
| CTXSYS | CTX_OBJECT_ATTRIBUTE_LOV | VIEW    | CTXSYS           | DR\$OBJECT               |
| CTXSYS | CTX_OBJECT_ATTRIBUTE_LOV | VIEW    | CTXSYS           | DR\$OBJECT_ATTRIBUTE     |
| CTXSYS | CTX_OBJECT_ATTRIBUTE_LOV | VIEW    | CTXSYS           | DR\$OBJECT_ATTRIBUTE_LOV |
| CTXSYS | CTX_PARAMETERS           | VIEW    | CTXSYS           | DR\$PARAMETER            |

At the bottom right, the Oracle logo is visible.

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

### Informes de Bases de Datos

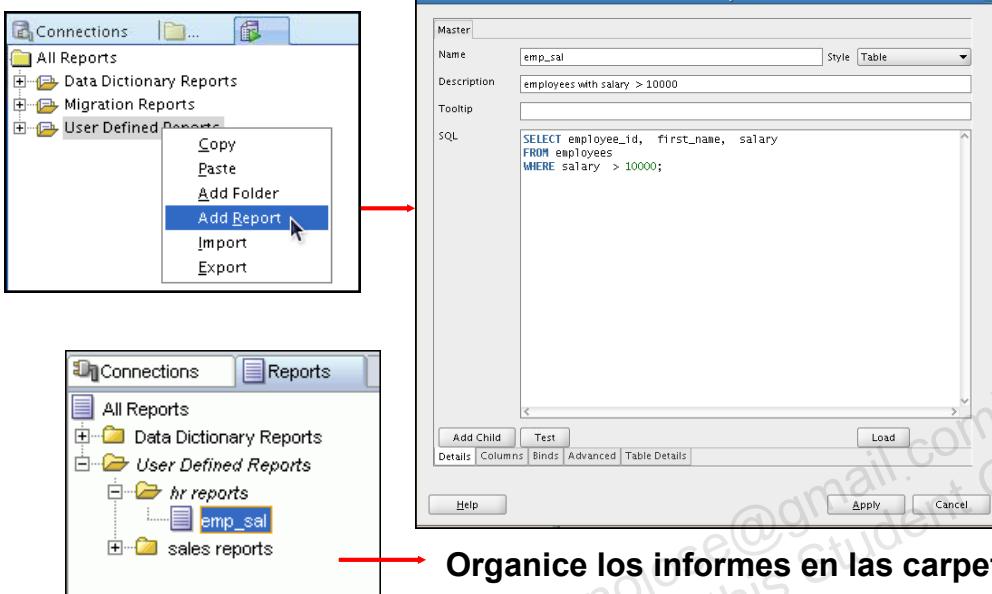
SQL Developer proporciona diferentes informes sobre la base de datos y sus objetos. Estos informes se pueden agrupar en las siguientes categorías:

- Informes About Your Database
- Informes Database Administration
- Informes Table
- Informes PL/SQL
- Informes Security
- Informes XML
- Informes Jobs
- Informes Streams
- Informes All Objects
- Informes Data Dictionary
- Informes User Defined

Para visualizar los informes, haga clic en el separador Reports situado a la izquierda de la ventana. Los informes individuales se muestran en los paneles con separadores situados a la derecha de la ventana y, para cada informe, puede seleccionar (mediante una lista desplegable) la conexión a la base de datos para la que desea mostrar el informe. Para los informes sobre objetos, sólo se muestran aquellos objetos que sean visibles para el usuario de la base de datos asociado a la conexión a la base de datos seleccionada y las filas ordenadas por propietario. También puede crear sus propios informes definidos por el usuario.

## Creación de un Informe Definido por el Usuario

Cree y guarde informes definidos por el usuario para un uso repetido.



ORACLE

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

### Creación de un Informe Definido por el Usuario

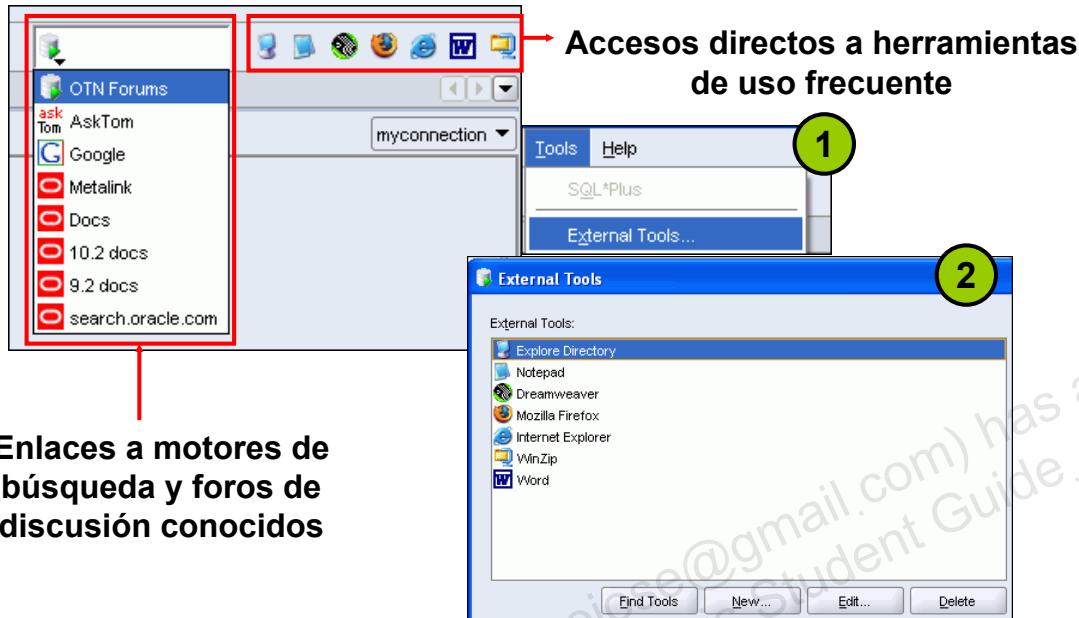
Los informes definidos por el usuario son los informes creados por los usuarios de SQL Developer. Para crear un informe definido por el usuario, realice los siguientes pasos:

1. Haga clic con el botón derecho en el nodo User Defined Reports en Reports y seleccione Add Report.
2. En el cuadro de diálogo Create Report, especifique el nombre del informe y la consulta SQL para recuperar la información para el informe. A continuación, haga clic en Apply.

En el ejemplo de la diapositiva, se especifica el nombre del informe como `emp_sal`. Se proporciona una descripción opcional que indica que el informe contiene los detalles de los empleados con `salary >= 10000`. La sentencia SQL completa para la recuperación de la información que se mostrará en el informe definido por el usuario se especifica en el cuadro SQL. También puede incluir una ayuda de burbuja opcional que se muestre al colocar el cursor brevemente sobre el nombre del informe en la pantalla del navegador Reports.

Puede organizar los informes definidos por el usuario en carpetas y puede crear una jerarquía de carpetas y subcarpetas. Para crear una carpeta para los informes definidos por el usuario, haga clic con el botón derecho en el nodo User Defined Reports o en cualquier nombre de carpeta de dicho nodo y seleccione Add Folder. La información sobre los informes definidos por el usuario, incluidas las carpetas de dichos informes, se almacena en un archivo denominado `UserReports.xml` en el directorio de información específica del usuario.

## Motores de Búsqueda y Herramientas Externas



ORACLE

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

### Motores de Búsqueda y Herramientas Externas

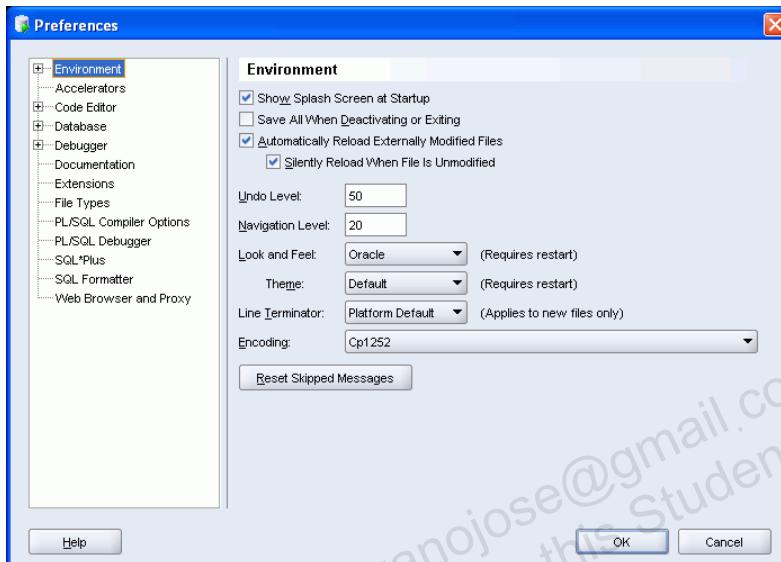
Para mejorar la productividad de los desarrolladores SQL, SQL Developer ha agregado enlaces rápidos a motores de búsqueda y foros de discusión conocidos, como AskTom, Google, etc. Asimismo, existen iconos de acceso directo a algunas de las herramientas de uso más frecuente, como Bloc de Notas, Microsoft Word y Dreamweaver.

Puede agregar herramientas externas a la lista existente o incluso suprimir los accesos directos a herramientas que no utilice frecuentemente. Para ello, realice los siguientes pasos:

1. En el menú Tools, seleccione External Tools.
2. En el cuadro de diálogo External Tools, seleccione New para agregar nuevas herramientas. Seleccione Delete para eliminar las herramientas de la lista.

## Definición de Preferencias

- Personalice la interfaz y el entorno de SQL Developer.
- En el menú Tools, seleccione Preferences.



ORACLE

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

### Definición de Preferencias

Puede personalizar muchos aspectos de la interfaz y del entorno de SQL Developer mediante la modificación de las preferencias de SQL Developer según sus preferencias y necesidades. Para modificar las preferencias de SQL Developer, seleccione Tools y, a continuación, Preferences.

Las preferencias se agrupan en las siguientes categorías:

- Entorno
- Aceleradores (accesos directos de teclado)
- Editores de códigos
- Base de datos
- Depurador
- Documentación
- Extensiones
- Tipos de archivo
- Migración
- Compiladores PL/SQL
- Depurador PL/SQL, etc.

## Restablecimiento del Diseño de SQL Developer

```
Terminal
File Edit View Terminal Tabs Help
[oracle@EDRSRSP1 ~]$ locate windowinglayout.xml
/home/oracle/.sqldeveloper/system1.5.4.59.40/o.ide.11.1.1.0.22.49.48/windowinglayout.xml
/home/oracle/.sqldeveloper/system1.5.4.59.41/o.ide.11.1.1.0.22.49.48/windowinglayout.xml
[oracle@EDRSRSP1 ~]$ cd /home/oracle/.sqldeveloper/system1.5.4.59.41/o.ide.11.1.1.0.22.49.48
[oracle@EDRSRSP1 o.ide.11.1.1.0.22.49.48]$ ls
Debugging.layout Editing.layout projects windowinglayout.xml
dtcache.xml preferences.xml settings.xml
[oracle@EDRSRSP1 o.ide.11.1.1.0.22.49.48]$ rm windowinglayout.xml
[oracle@EDRSRSP1 o.ide.11.1.1.0.22.49.48]$
```



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

### Restablecimiento del Diseño de SQL Developer

Mientras trabaja con SQL Developer, si desaparece Connections Navigator o si no puede acoplar la ventana Log en su lugar original, realice los pasos siguientes para corregir el problema:

1. Salga de SQL Developer.
2. Abra una ventana de terminal y utilice el comando `locate` para encontrar la ubicación de `windowinglayout.xml`.
3. Vaya al directorio que contenga `windowinglayout.xml` y suprímalo.
4. Reinicie SQL Developer.

## Resumen

En este apéndice, debe haber aprendido cómo utilizar SQL Developer para realizar las siguientes acciones:

- Examinar, crear y editar objetos de bases de datos
- Ejecutar sentencias SQL y scripts en la hoja de trabajo de SQL
- Crear y guardar informes personalizados



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

### Resumen

SQL Developer es una herramienta gráfica gratuita para simplificar las tareas de desarrollo de la base de datos. Con SQL Developer, puede examinar, crear y editar objetos de bases de datos. Puede utilizar la hoja de trabajo de SQL para ejecutar scripts y sentencias SQL. SQL Developer permite crear y guardar su propio juego especial de informes para un uso repetido.

Jose Enrique Lozano (lozanojose@gmail.com) has a  
non-transferable license to use this Student Guide.

# Uso de SQL\*Plus

ORACLE

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

## Objetivos

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

- Conectarse a SQL\*Plus
- Editar comandos SQL
- Formatear la salida con comandos SQL\*Plus
- Interactuar con archivos de script

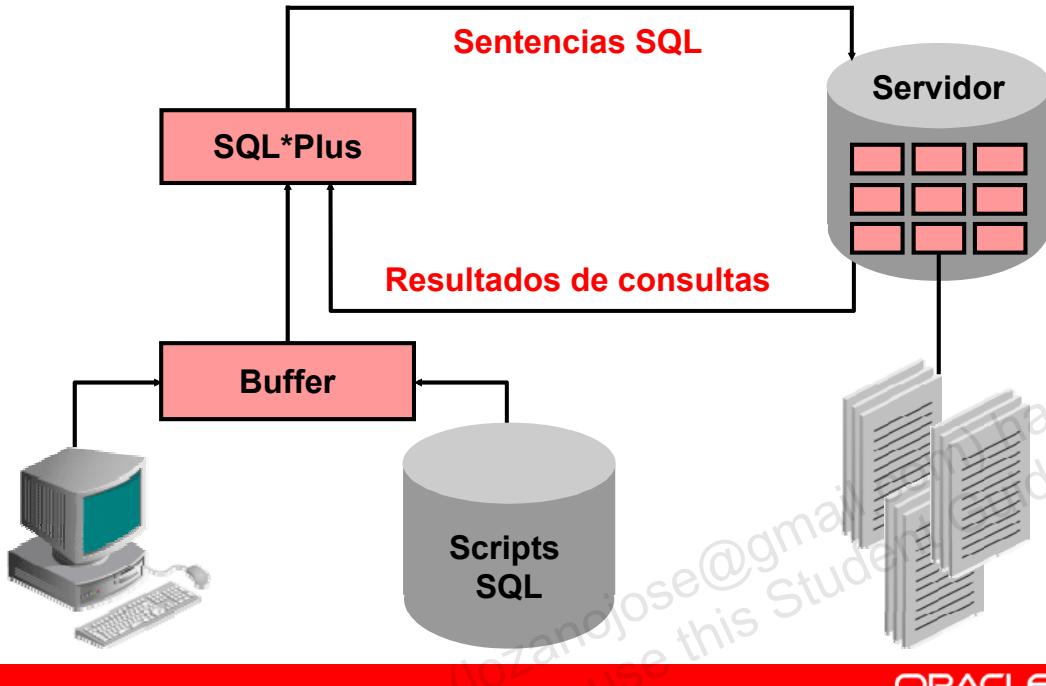


Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

### Objetivos

Puede que desee crear sentencias SELECT que se puedan utilizar repetidamente. En este apéndice también se aborda el uso de comandos SQL\*Plus para ejecutar sentencias SQL. Aprenderá cómo formatear la salida mediante comandos SQL\*Plus, editar comandos SQL y guardar scripts en SQL\*Plus.

## Interacción de SQL y SQL\*Plus



ORACLE

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

### SQL y SQL\*Plus

SQL es un lenguaje de comandos que se utiliza para la comunicación con el servidor de Oracle desde cualquier herramienta o aplicación. Oracle SQL contiene muchas extensiones. Al introducir una sentencia SQL, ésta se almacena en una parte de la memoria denominada *buffer SQL* y permanece allí hasta que introduzca una nueva sentencia SQL. SQL\*Plus es una herramienta de Oracle que reconoce y envía sentencias SQL en Oracle9i para su ejecución. Contiene su propio lenguaje de comandos.

### Funciones de SQL

- Las pueden utilizar una gran variedad de usuarios, incluidos aquéllos con poca o ninguna experiencia de programación.
- Es un lenguaje que no es de procedimientos.
- Reduce la cantidad de tiempo necesario para crear y mantener sistemas.
- Es un lenguaje como el inglés.

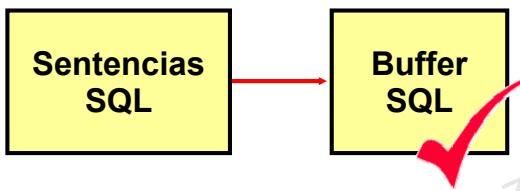
### Funciones de SQL\*Plus

- Acepta la entrada ad hoc de sentencias.
- Acepta la entrada de SQL de los archivos.
- Proporciona un editor de líneas para modificar sentencias SQL.
- Controla la configuración de entorno.
- Formatea resultados de consulta en informes básicos.
- Accede a bases de datos locales y remotas.

## Sentencias SQL frente a Comandos SQL\*Plus

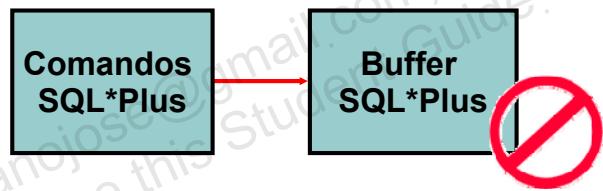
SQL

- Un lenguaje
  - Est醖ar de ANSI
  - Las palabras clave no se pueden abreviar.
  - Las sentencias manipulan definiciones de tablas y datos en la base de datos.



## SQL\*Plus

- Un entorno
  - Propiedad de Oracle
  - Las palabras clave se pueden abbreviar.
  - Los comandos no permiten la manipulación de valores en la base de datos.



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

## **SQL y SQL\*Plus (continuación)**

En la siguiente tabla se compara SQL y SQL\*Plus:

| SQL                                                                               | SQL*Plus                                                                                     |
|-----------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| Lenguaje para la comunicación con el servidor de Oracle para acceder a los datos. | Reconoce las sentencias SQL y las envía al servidor.                                         |
| Se basa en SQL del estándar ANSI (American National Standards Institute)          | Interfaz propiedad de Oracle para ejecutar sentencias SQL.                                   |
| Manipula las definiciones de tablas y datos en la base de datos.                  | No permite la manipulación de valores en la base de datos.                                   |
| Se introduce en el buffer SQL en una o más líneas.                                | Se introduce en una línea al mismo tiempo y no se almacena en el buffer SQL.                 |
| No tiene ningún carácter de continuación.                                         | Utiliza un guión (-) como carácter de continuación si el comando es más largo que una línea. |
| No se puede abbreviar.                                                            | Se puede abbreviar.                                                                          |
| Utiliza un carácter de terminación para ejecutar comandos inmediatamente.         | No necesita caracteres de terminación; ejecuta los comandos inmediatamente.                  |
| Utiliza funciones para realizar algunas tareas de formato.                        | Utiliza comandos para formatear datos.                                                       |

## Visión General de SQL\*Plus

- Conéctese a SQL\*Plus.
- Describa la estructura de la tabla.
- Edite la sentencia SQL.
- Ejecute SQL desde SQL\*Plus.
- Guarde sentencias SQL en archivos y agregue sentencias SQL a los archivos.
- Ejecute archivos guardados.
- Cargue comandos del archivo en el buffer para la edición.



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

### SQL\*Plus

SQL\*Plus es un entorno en el que puede:

- Ejecutar sentencias SQL para recuperar, modificar, agregar y eliminar datos de la base de datos.
- Formatear, realizar cálculos, almacenar e imprimir resultados de consulta como informes.
- Crear archivos de script para almacenar sentencias SQL para un uso repetido en el futuro.

Los comandos SQL\*Plus se pueden dividir en las siguientes categorías principales:

| Categoría                | Objetivo                                                                                                  |
|--------------------------|-----------------------------------------------------------------------------------------------------------|
| Entorno                  | Afectar al comportamiento general de sentencias SQL para la sesión.                                       |
| Formato                  | Formatear resultados de consulta.                                                                         |
| Manipulación de archivos | Guardar, cargar y ejecutar scripts.                                                                       |
| Ejecución                | Enviar sentencias SQL del buffer SQL al servidor de Oracle.                                               |
| Edición                  | Modificar sentencias SQL en el buffer.                                                                    |
| Interacción              | Crear y transferir variables a la sentencia SQL, imprimir valores de variables y mensajes en la pantalla. |
| Otros                    | Conectarse a la base de datos, manipular el entorno SQL*Plus y mostrar definiciones de columnas.          |

## Conexión a SQL\*Plus

```
Terminal
File Edit View Terminal Tabs Help
[oracle@EDRSR5P1 ~]$sqlplus
SQL*Plus: Release 11.2.0.0.2 Beta on Tue May 26 19:59:06 2009
Copyright (c) 1982, 2009, Oracle. All rights reserved.
Enter user-name: ora21@orcl
Enter password:
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.0.2 - Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL>
```

**sqlplus [username[/password[@database]]]**

```
Terminal
File Edit View Terminal Tabs Help
[oracle@EDRSR5P1 ~]$sqlplus ora21/ora21@orcl
SQL*Plus: Release 11.2.0.0.2 Beta on Tue May 26 19:58:06 2009
Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.0.2 - Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL>
```

**ORACLE**

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

### Conexión a SQL\*Plus

La forma de llamar a SQL\*Plus dependerá del tipo de sistema operativo que esté ejecutando con Oracle Database.

Para conectarse desde un entorno Linux:

- Haga clic con el botón derecho en el escritorio de Linux y seleccione el terminal.
- Introduzca el comando `sqlplus` que se muestra en la diapositiva.
- Introduzca el nombre de usuario, la contraseña y el nombre de la base de datos.

En la sintaxis:

|                        |                                                                                  |
|------------------------|----------------------------------------------------------------------------------|
| <code>username</code>  | Nombre de usuario de la base de datos                                            |
| <code>password</code>  | Contraseña de la base de datos (la contraseña será visible si la introduce aquí) |
| <code>@database</code> | Cadena de conexión de la base de datos                                           |

**Nota:** para asegurarse de la integridad de la contraseña, no la introduzca en la petición de datos del sistema operativo. En su lugar, introduzca sólo el nombre de usuario. Introduzca la contraseña en la petición de datos de la contraseña.

## Visualización de la Estructura de la Tabla

Utilice el comando SQL\*Plus `DESCRIBE` para mostrar la estructura de una tabla:

```
DESC[RIBE] tablename
```



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

### Visualización de la Estructura de la Tabla

En SQL\*Plus, puede mostrar la estructura de una tabla mediante el comando `DESCRIBE`. El resultado del comando es una visualización de los nombres de columna y tipos de dato, así como una indicación de si una columna debe contener datos.

En la sintaxis:

`tablename` Nombre de cualquier tabla existente, vista o sinónimo al que puede acceder el usuario

Para describir la tabla `DEPARTMENTS`, utilice este comando:

```
SQL> 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)
```

## Visualización de la Estructura de la Tabla

```
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)    |



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

### Visualización de la Estructura de Tabla (continuación)

En el ejemplo de la diapositiva se muestra la información sobre la estructura de la tabla DEPARTMENTS. En el resultado:

Null?: especifica si una columna debe contener datos (NOT NULL indica que una columna debe contener datos.)

Type: muestra el tipo de dato de una columna

## Comandos de Edición SQL\*Plus

- A [PPEND] *text*
- C [HANGE] / *old* / *new*
- C [HANGE] / *text* /
- CL [EAR] BUFF [ER]
- DEL
- DEL *n*
- DEL *m n*



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

### Comandos de Edición SQL\*Plus

Los comandos SQL\*Plus se introducen en una línea al mismo tiempo y no se almacenan en el buffer SQL.

| Comando                             | Descripción                                                     |
|-------------------------------------|-----------------------------------------------------------------|
| A [PPEND] <i>text</i>               | Agrega texto al final de la línea actual.                       |
| C [HANGE] / <i>old</i> / <i>new</i> | Cambia el texto <i>antiguo</i> por el nuevo en la línea actual. |
| C [HANGE] / <i>text</i> /           | Suprime el <i>texto</i> de la línea actual.                     |
| CL [EAR] BUFF [ER]                  | Suprime todas las líneas del buffer SQL.                        |
| DEL                                 | Suprime la línea actual.                                        |
| DEL <i>n</i>                        | Suprime la línea <i>n</i> .                                     |
| DEL <i>m n</i>                      | Suprime de las líneas <i>m</i> hasta la <i>n</i> , inclusive.   |

### Instrucciones

- Si pulsa Intro antes de que haya terminado la ejecución de un comando, SQL\*Plus le solicitará un número de línea.
- Termine el buffer SQL introduciendo uno de los caracteres de terminación (punto y coma o barra) o pulsando Intro dos veces. A continuación, aparecerá la petición de datos de SQL.

## Comandos de Edición SQL\*Plus

- I [NPUT]
- I [NPUT] *text*
- L[IST]
- L[IST] *n*
- L[IST] *m n*
- R[UN]
- *n*
- *n text*
- 0 *text*

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

### Comandos de Edición SQL\*Plus (continuación)

| Comando              | Descripción                                                    |
|----------------------|----------------------------------------------------------------|
| I [NPUT]             | Inserta un número indefinido de líneas.                        |
| I [NPUT] <i>text</i> | Inserta una línea que consta de <i>texto</i> .                 |
| L[IST]               | Muestra todas las líneas en el buffer SQL.                     |
| L[IST] <i>n</i>      | Muestra una línea (especificada por <i>n</i> ).                |
| L[IST] <i>m n</i>    | Muestra un rango de líneas ( <i>m</i> a <i>n</i> ), inclusive. |
| R[UN]                | Muestra y ejecuta la sentencia SQL actual en el buffer.        |
| <i>n</i>             | Especifica la línea para crear la línea actual.                |
| <i>n text</i>        | Sustituye la línea <i>n</i> con <i>texto</i> .                 |
| 0 <i>text</i>        | Inserta una línea delante de la línea 1.                       |

**Nota:** puede introducir sólo un comando SQL\*Plus para cada petición de datos de SQL. Los comandos SQL\*Plus no se almacenan en el buffer. Para que un comando SQL\*Plus continúe en la siguiente línea, finalice la primera línea con un guión (-).

## Uso de LIST, n y APPEND

```
LIST
 1 SELECT last_name
 2* FROM employees
```

```
1
1* SELECT last_name
```

```
A , job_id
1* SELECT last_name, job_id
```

```
LIST
 1 SELECT last_name, job_id
 2* FROM employees
```

**ORACLE**

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

### Uso de LIST, n y APPEND

- Utilice el comando L [IST] para mostrar el contenido del buffer SQL. El asterisco (\*) situado junto a la línea 2 en el buffer indica que la línea 2 es la línea actual. Cualquier edición que realice se aplica a la línea actual.
- Introduzca el número (n) de la línea que desea editar para cambiar el número de la línea actual. Se muestra la nueva línea actual.
- Utilice el comando A [PPEND] para agregar texto a la línea actual. Se muestra la línea recién editada. Verifique el nuevo contenido del buffer mediante el comando LIST.

**Nota:** muchos de los comandos SQL\*Plus, incluidos LIST y APPEND, se pueden abreviar sólo con su primera letra. LIST se puede abreviar con L; APPEND se puede abreviar con A.

## Uso del Comando CHANGE

```
LIST
```

```
1* SELECT * from employees
```

```
c/employees/departments
```

```
1* SELECT * from departments
```

```
LIST
```

```
1* SELECT * from departments
```



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

### Uso del Comando CHANGE

- Utilice L [ LIST ] para mostrar el contenido del buffer.
- Utilice el comando C [ CHANGE ] para modificar el contenido de la línea actual del buffer SQL. En este caso, sustituya la tabla employees por la tabla departments. Se muestra la nueva línea actual.
- Utilice el comando L [ LIST ] para verificar el nuevo contenido del buffer.

## Comandos de Archivos SQL\*Plus

- `SAVE filename`
- `GET filename`
- `START filename`
- `@ filename`
- `EDIT filename`
- `SPOOL filename`
- `EXIT`



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

### Comandos de Archivos SQL\*Plus

Las sentencias SQL se comunican con el servidor de Oracle. Los comandos SQL\*Plus controlan el entorno, formatean los resultados de la consulta y gestionan archivos. Puede utilizar los comandos descritos en la siguiente tabla:

| Comando                                                    | Descripción                                                                                                                                                                                            |
|------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>SAV[E] filename [.ext]</code><br>[REP[LACE]APP[END]] | Guarda el contenido actual del buffer SQL en un archivo. Utilice APPEND para agregar a un archivo existente; utilice REPLACE para sobrescribir un archivo existente. La extensión por defecto es .sql. |
| <code>GET filename [.ext]</code>                           | Escribe el contenido de un archivo guardado anteriormente en el buffer SQL. La extensión por defecto del nombre de archivo es .sql.                                                                    |
| <code>STA[RT] filename [.ext]</code>                       | Ejecuta el archivo de comandos guardado anteriormente.                                                                                                                                                 |
| <code>@ filename</code>                                    | Ejecuta un archivo de comandos guardado anteriormente (igual que START).                                                                                                                               |
| <code>ED[IT]</code>                                        | Llama al editor y guarda el contenido del buffer en un archivo denominado afiedt.buf.                                                                                                                  |
| <code>ED[IT] [filename[.ext]]</code>                       | Llama al editor para editar el contenido de un archivo guardado.                                                                                                                                       |
| <code>SPO[OL] [filename[.ext]]   OFF OUT</code>            | Almacena los resultados de la consulta en un archivo. OFF cierra el archivo de spool. OUT cierra el archivo de spool y envía los resultados del archivo a la impresora.                                |
| <code>EXIT</code>                                          | Sale de SQL*Plus.                                                                                                                                                                                      |

## Uso de los Comandos SAVE y START

```
LIST
```

```
1 SELECT last_name, manager_id, department_id
2* FROM employees
```

```
SAVE my_query
```

```
Created file my_query
```

```
START my_query
```

| LAST_NAME          | MANAGER_ID | DEPARTMENT_ID |
|--------------------|------------|---------------|
| King               |            | 90            |
| Kochhar            | 100        | 90            |
| ...                |            |               |
| 107 rows selected. |            |               |

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

## Uso de los Comandos SAVE y START

### SAVE

Utilice el comando `SAVE` para almacenar el contenido actual del buffer en un archivo. De esta forma, podrá almacenar los scripts usados con frecuencia para utilizarlos en el futuro.

### START

Utilice el comando `START` para ejecutar un script en SQL\*Plus. También puede utilizar el símbolo `@` para ejecutar un script.

```
@my_query
```

## Comando SERVEROUTPUT

- Utilice el comando `SET SERVEROUT [ PUT ]` para controlar si se debe mostrar la salida de los procedimientos almacenados o de los bloques PL/SQL en SQL\*Plus.
- El límite de la longitud de línea `DBMS_OUTPUT` aumenta de 255 bytes a 32767 bytes.
- El tamaño por defecto ahora es ilimitado.
- Los recursos no están preasignados cuando se define SERVEROUTPUT.
- Debido a que no hay ninguna penalización de rendimiento, utilice `UNLIMITED` a menos que desee conservar la memoria física.

```
SET SERVEROUT[PUT] {ON | OFF} [{SIZE {n | UNLIMTED}}]
[FOR[MAT] {WRA[PPED] | WOR[D_WRAPPED] | TRU[NATED]}]
```



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

### Comando SERVEROUTPUT

La mayoría de los programas PL/SQL realizan la entrada y salida mediante sentencias SQL para almacenar datos en tablas de base de datos o para consultar dichas tablas. Las demás entradas o salidas PL/SQL se realizan mediante API que interactúan con otros programas. Por ejemplo, el paquete `DBMS_OUTPUT` tiene procedimientos como `PUT_LINE`. Para ver el resultado fuera de PL/SQL, se necesita otro programa, como SQL\*Plus, para leer y mostrar los datos transferidos a `DBMS_OUTPUT`.

SQL\*Plus no muestra los datos de `DBMS_OUTPUT` a no ser que antes emita el comando SQL\*Plus `SET SERVEROUTPUT ON`, de la siguiente forma:

```
SET SERVEROUTPUT ON
```

#### Nota

- `SIZE` define el número de bytes de la salida almacenados en buffer en el servidor de Oracle Database. El valor por defecto es `UNLIMITED`. `n` no puede ser menor que 2000 ni mayor que 1.000.000.
- Para obtener más información sobre SERVEROUTPUT, consulte *Oracle Database PL/SQL User's Guide and Reference 11g* (Guía del Usuario y Referencia PL/SQL de Oracle Database 11g).

## Uso del Comando SQL\*Plus SPOOL

```
SPO[OL] [file_name[.ext] [CRE[ATE] | REP[LACE] | APP[END]] | OFF | OUT]
```

| Opción          | Descripción                                                                                       |
|-----------------|---------------------------------------------------------------------------------------------------|
| file_name[.ext] | Envía la salida al nombre de archivo especificado                                                 |
| CRE[ATE]        | Crea un nuevo archivo con el nombre especificado                                                  |
| REP[LACE]       | Sustituye el contenido de un archivo existente. Si el archivo no existe, REPLACE crea el archivo. |
| APP[END]        | Agrega el contenido del buffer al final del archivo especificado                                  |
| OFF             | Para el envío                                                                                     |
| OUT             | Para el envío y manda el archivo a la impresora estándar (por defecto) de la computadora          |

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

### Uso del Comando SQL\*Plus SPOOL

El comando SPOOL almacena el resultado de la consulta en un archivo o bien envía el archivo a una impresora. El comando SPOOL se ha mejorado. Ahora puede agregar o sustituir un archivo existente, donde antes sólo se podía utilizar SPOOL para crear (y sustituir) un archivo. REPLACE es el valor por defecto.

Para enviar la salida generada por comandos en un script sin que la salida aparezca en pantalla, utilice SET TERMOUT OFF. SET TERMOUT OFF no afecta a la salida de comandos que se ejecutan de forma interactiva.

Debe entrecerrar los nombres de archivos que contengan espacio en blanco. Para crear un archivo HTML válido con comandos SPOOL APPEND, debe utilizar el comando PROMPT o uno similar para crear la cabecera y el pie de página HTML. El comando SPOOL APPEND no analiza etiquetas HTML. Defina SQLPLUSCOMPAT [IBILITY] en 9.2 o una versión anterior para desactivar los parámetros CREATE, APPEND y SAVE.

## Uso del Comando AUTOTRACE

- Muestra un informe después de la ejecución correcta de sentencias de manipulación de datos (DML) SQL como SELECT, INSERT, UPDATE o DELETE.
- El informe puede ahora incluir estadísticas de ejecución y la ruta de acceso de ejecución de la consulta.

```
SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]]
[STATISTICS]
```

```
SET AUTOTRACE ON
-- The AUTOTRACE report includes both the optimizer
-- execution path and the SQL statement execution
-- statistics
```



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

### Uso del Comando AUTOTRACE

EXPLAIN muestra la ruta de acceso de ejecución de la consulta realizando un EXPLAIN PLAN. STATISTICS muestra las estadísticas de la sentencia SQL. El formato del informe AUTOTRACE puede variar en función de la versión del servidor al que esté conectado y de la configuración del servidor. El paquete DBMS\_XPLAN proporciona una forma fácil de mostrar la salida del comando EXPLAIN PLAN en varios formatos predefinidos.

#### Nota

- Para obtener más información sobre el paquete y los subprogramas, consulte *Oracle Database PL/SQL Packages and Types Reference 11g* (Referencia de Tipos y Paquetes PL/SQL de Oracle Database 11g).
- Para obtener más información sobre EXPLAIN PLAN, consulte *Oracle Database SQL Reference 11g* (Referencia SQL de Oracle Database 11g).
- Para obtener más información sobre los planes de ejecución y las estadísticas, consulte *Oracle Database Performance Tuning Guide 11g* (Guía de Ajuste de Rendimiento de Oracle Database 11g).

## Resumen

En este apéndice, debe haber aprendido cómo utilizar SQL\*Plus como un entorno para realizar las siguientes acciones:

- Ejecutar sentencias SQL
- Editar sentencias SQL
- Formatear la salida
- Interactuar con archivos de script



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

### Resumen

SQL\*Plus es un entorno de ejecución que puede utilizar para enviar comandos SQL al servidor de la base datos para editar y guardar los comandos SQL. Puede ejecutar los comandos desde la petición de datos de SQL o desde un archivo de script.

# JDeveloper

## Uso de JDeveloper

ORACLE

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

Jose Enrique Lozano (lozanojose@gmail.com) has a  
non-transferable license to use this Student Guide.

## Objetivos

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

- Mostrar las funciones clave de Oracle JDeveloper
- Crear una conexión de base de datos en JDeveloper
- Gestionar objetos de base de datos en JDeveloper
- Utilizar JDeveloper para ejecutar comandos SQL
- Crear y ejecutar unidades de programa PL/SQL

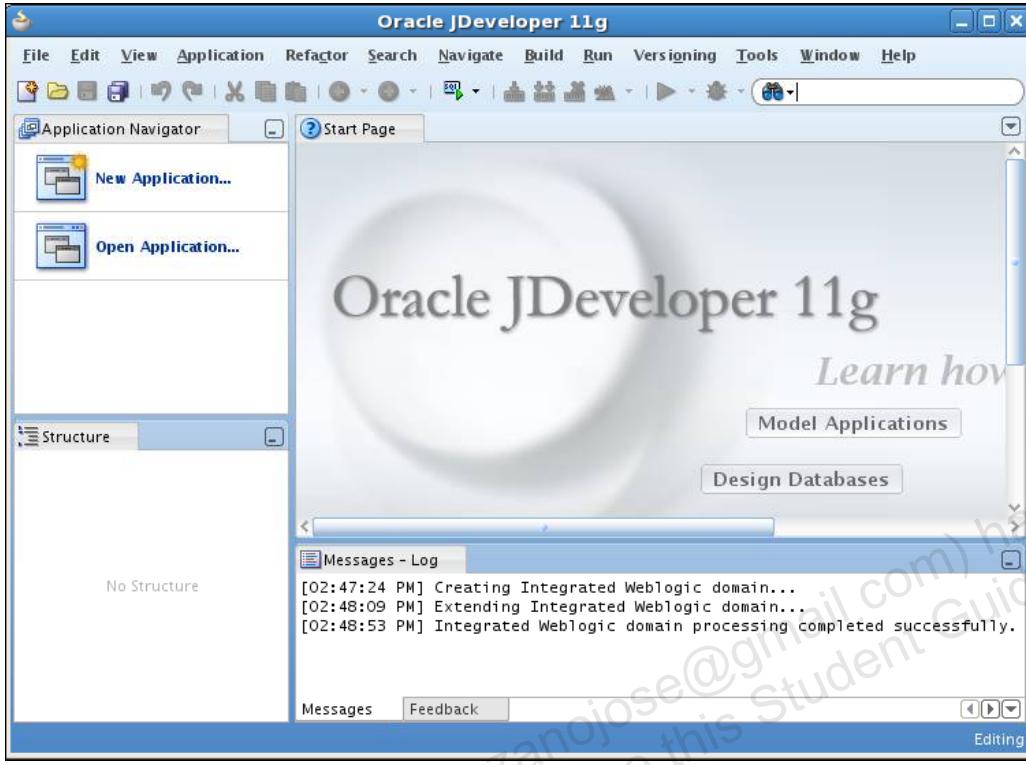


Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

### Objetivos

En este apéndice, se presentará la herramienta JDeveloper. Aprenderá cómo utilizar JDeveloper para las tareas de desarrollo de la base de datos.

## Oracle JDeveloper



ORACLE

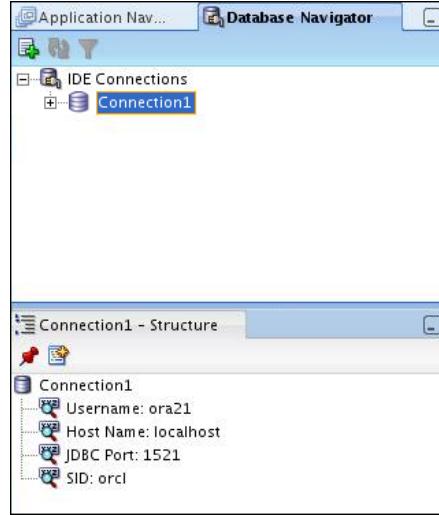
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

### Oracle JDeveloper

Oracle JDeveloper 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 11g 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.

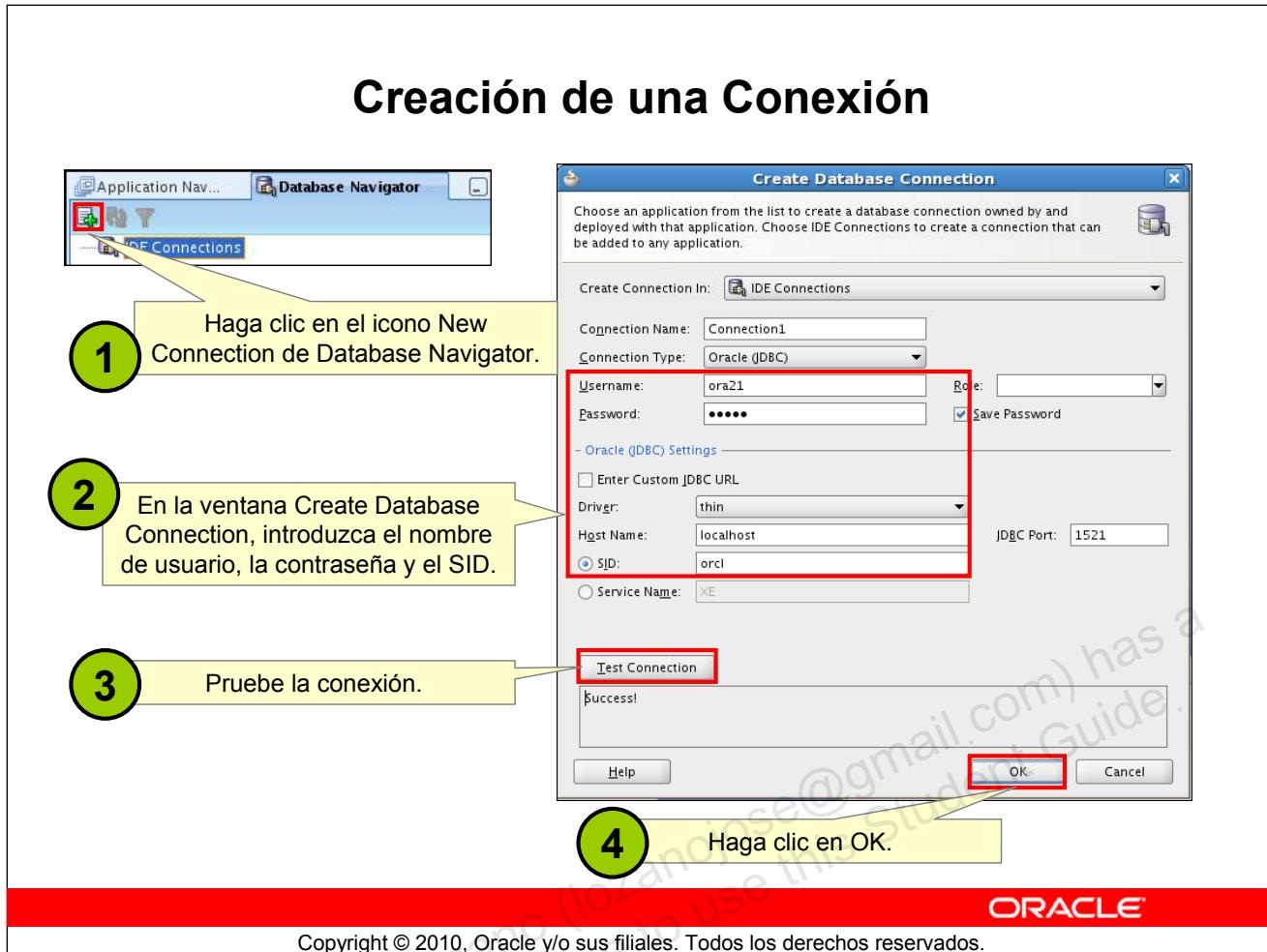
## Database Navigator

The ORACLE logo, which consists of the word "ORACLE" in white capital letters on a red horizontal bar.

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

### Database Navigator

Con Oracle JDeveloper, puede almacenar la información necesaria para conectar a la base de datos en un objeto denominado “conexión”. Una conexión se almacena como parte de la configuración de IDE y se puede exportar e importar para compartirlo 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.



## Creación de una Conexión

Una conexión es un objeto que especifica la información necesaria para conectarse a una base de datos concreta como usuario específico de dicha base de datos. Puede crear y probar conexiones para varias bases de datos y esquemas.

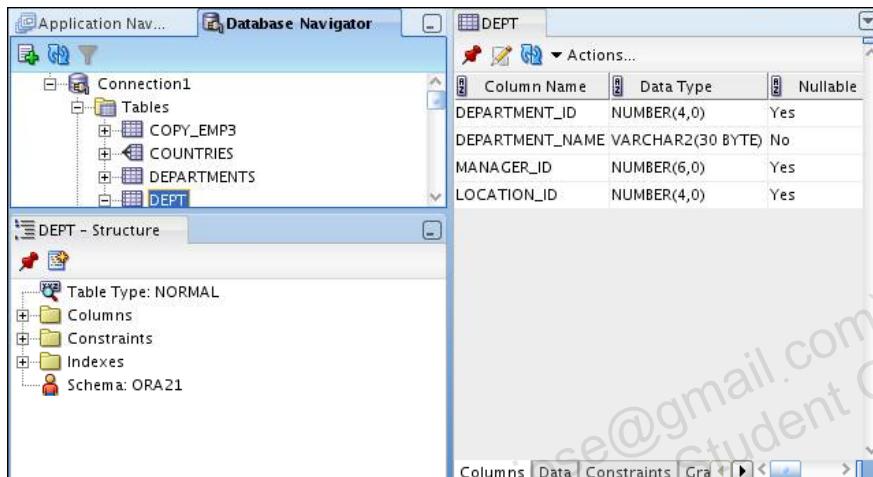
Para crear una conexión a la base de datos, realice los siguientes pasos:

1. Haga clic en el ícono New Connection de Database Navigator.
2. En la ventana Create Database Connection, introduzca el nombre de la conexión. Introduzca el nombre de usuario y la contraseña del esquema al que desea conectarse. Introduzca el SID de la base de datos a la que desea conectarse.
3. Haga clic en Test para asegurarse de que la conexión se ha definido correctamente.
4. Haga clic en OK.

## Exploración de Objetos de Bases de Datos

Utilice Database Navigator para:

- Examinar los objetos de un esquema de base de datos
- Revisar las definiciones de objetos de forma rápida



ORACLE

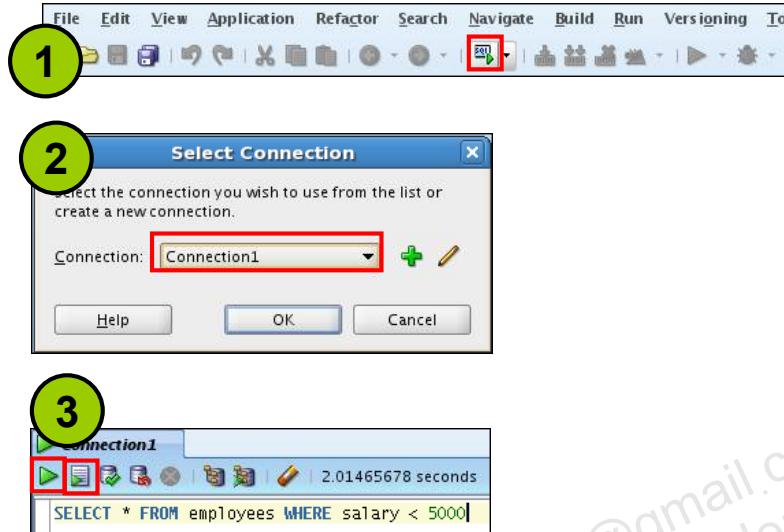
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

### Exploración de Objetos de Bases de Datos

Una vez creada la conexión a la base de datos, puede utilizar Database Navigator para examinar muchos objetos de un esquema de base de datos, entre los que se incluyen tablas, vistas, índices, paquetes, procedimientos, disparadores y tipos.

Puede ver las definiciones de los objetos desglosados en separadores de información que se pasa al diccionario de datos. Por ejemplo, si selecciona una tabla en Navigator, se muestran los detalles sobre las columnas, las restricciones, los permisos, las estadísticas, los disparadores, etc. en una página con separadores fácil de leer.

## Ejecución de Sentencias SQL



ORACLE

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

### Ejecución de Sentencias SQL

Para ejecutar una sentencia SQL, realice los siguientes pasos:

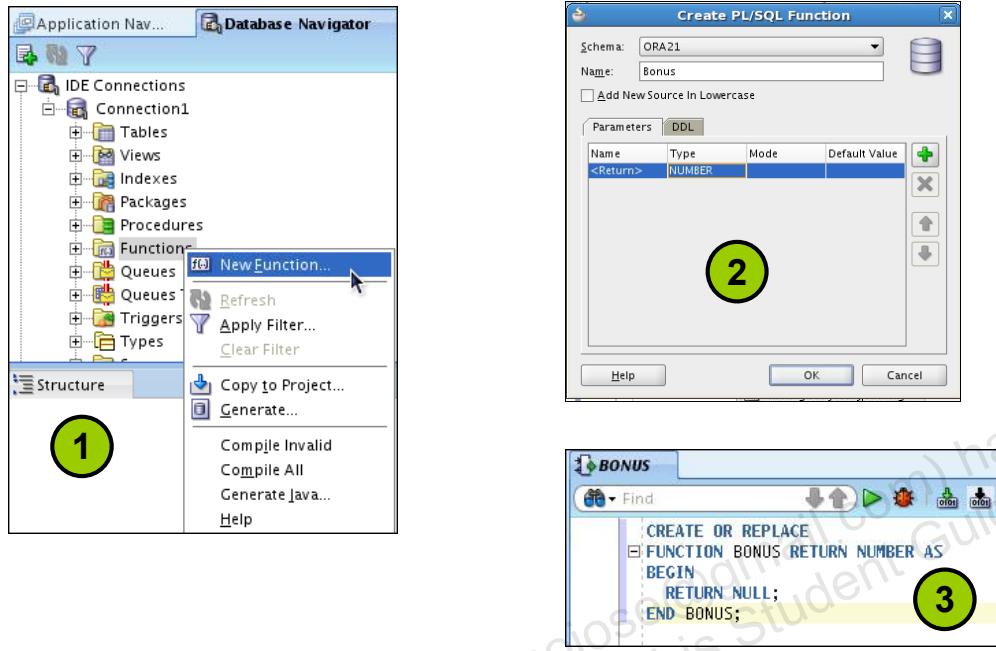
1. Haga clic en el ícono Open SQL Worksheet.
2. Seleccione la conexión.
3. Ejecute el comando SQL haciendo clic en:
  - El botón **Execute statement** o pulsando F9. La salida es la siguiente:

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME |
|-------------|------------|-----------|
| 1           | Steven     | King      |
| 2           | Neena      | Kochhar   |

- El botón **Run Script** o pulsando F5. La salida es la siguiente:

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME |
|-------------|------------|-----------|
| 100         | Steven     | King      |

## Creación de Unidades de Programa



**Esqueleto de la Función**

ORACLE

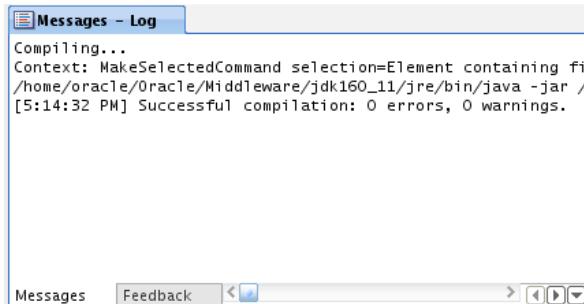
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

### Creación de Unidades de Programa

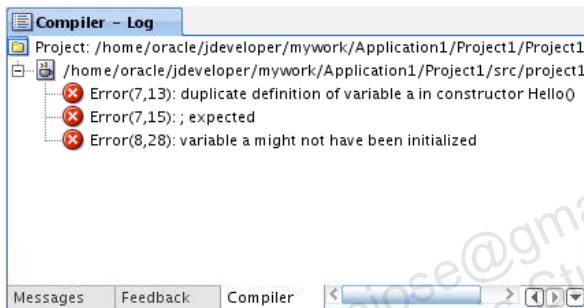
Para crear una unidad de programa PL/SQL:

1. Seleccione View > Database Navigator. Seleccione y amplíe una conexión de base de datos. Haga clic con el botón derecho en una carpeta que corresponda al tipo de objeto (Procedures, Packages, Functions). Seleccione “New [Procedures|Packages|Functions]”.
2. Introduzca un nombre válido para la función, el paquete o el procedimiento y haga clic en OK.
3. Se crea una estructura básica 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

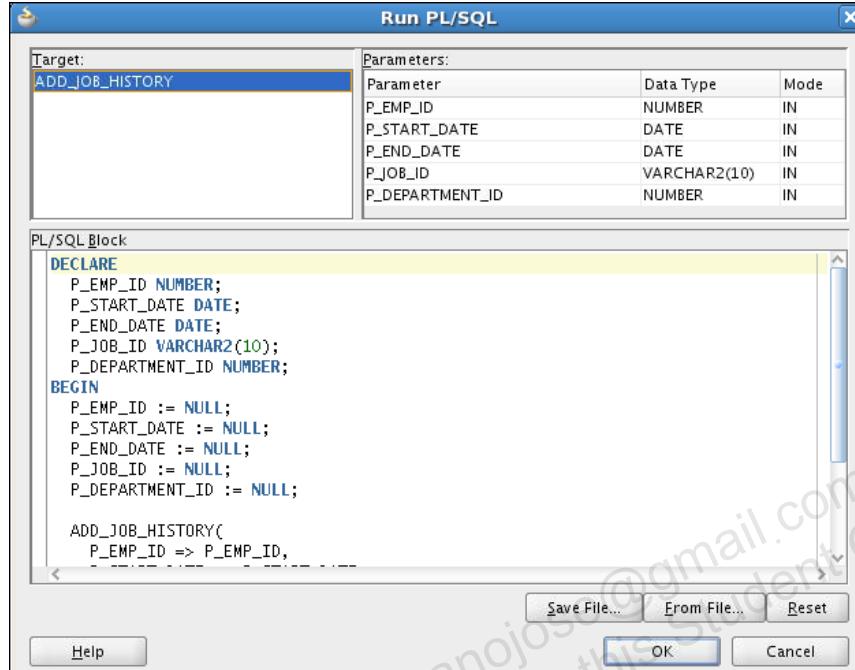
ORACLE

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

### Compilación

Después de editar la definición del esqueleto, debe compilar la unidad de programa. Haga clic con el botón derecho 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



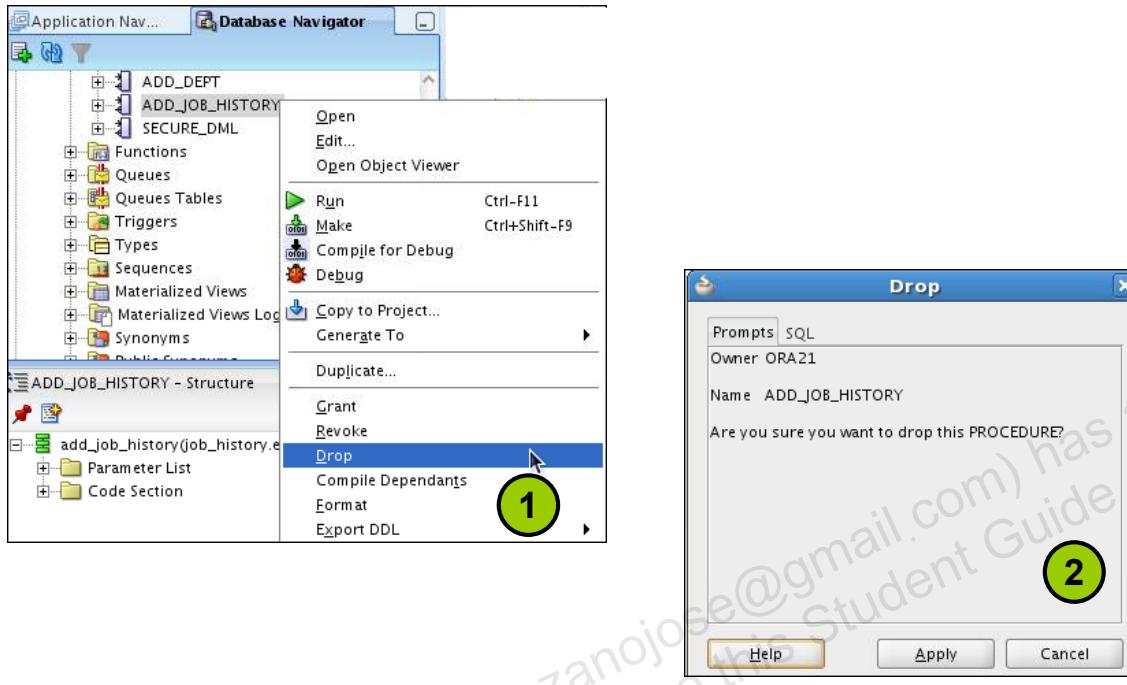
ORACLE

Copyright © 2010, Oracle y/o sus filiales. 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 seleccione Run. Aparece el cuadro 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 muestra en la ventana Message-Log.

## Borrado de una Unidad de Programa



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

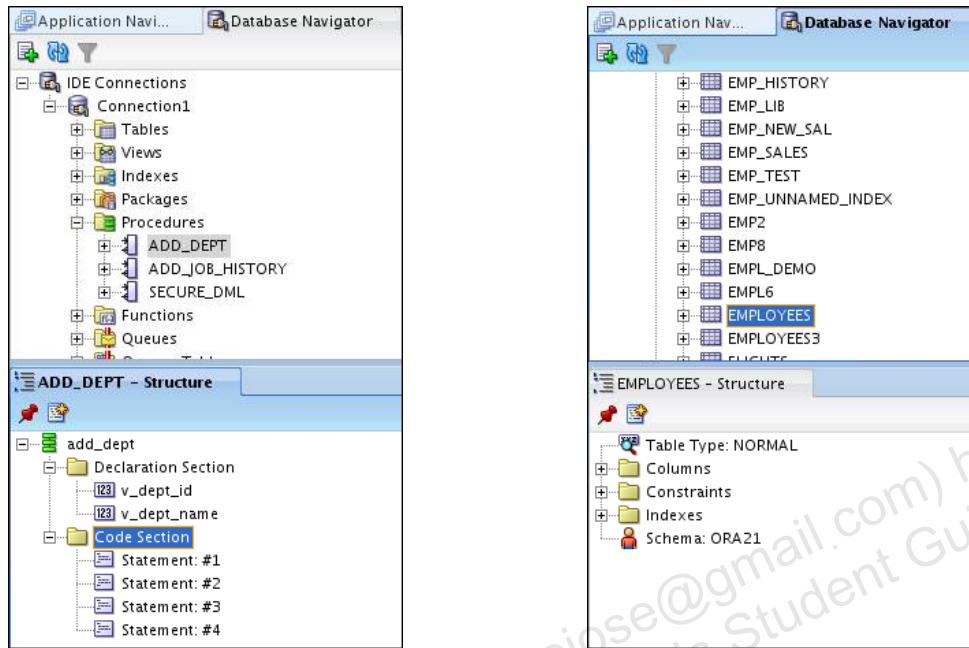
**ORACLE**

### Borrado de una Unidad de Programa

Para borrar una unidad de programa:

1. Haga clic con el botón derecho en el objeto y seleccione Drop.  
Aparece el cuadro de diálogo Drop Confirmation.
2. Haga clic en Apply.  
El objeto se borrará de la base de datos.

## Ventana Structure



ORACLE

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

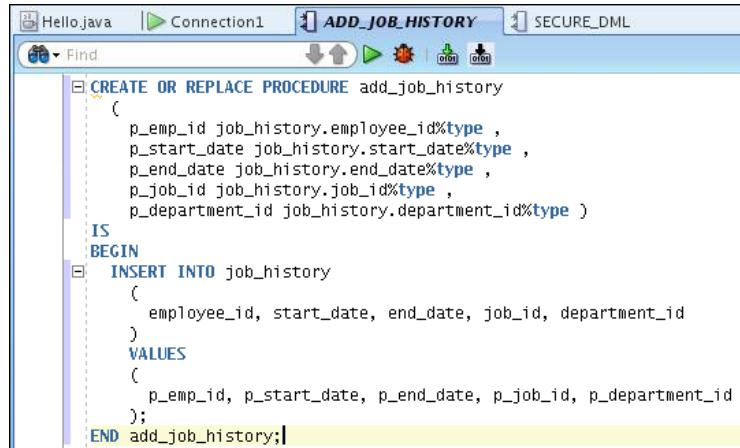
### Ventana Structure

La ventana Structure ofrece una vista estructural de los datos del documento que está 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 que están disponibles para visualización 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



The screenshot shows the Oracle SQL Developer interface. The title bar says "Hello.java | Connection1 ADD\_JOB\_HISTORY SECURE\_DML". The main area contains the following PL/SQL code:

```
CREATE OR REPLACE PROCEDURE add_job_history
(
 p_emp_id job_history.employee_id%type ,
 p_start_date job_history.start_date%type ,
 p_end_date job_history.end_date%type ,
 p_job_id job_history.job_id%type ,
 p_department_id job_history.department_id%type)
IS
BEGIN
 INSERT INTO job_history
 (
 employee_id, start_date, end_date, job_id, department_id
)
 VALUES
 (
 p_emp_id, p_start_date, p_end_date, p_job_id, p_department_id
);
END add_job_history;
```



ORACLE

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

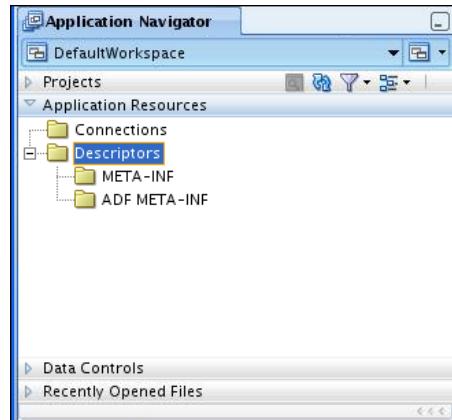
### Ventana Editor

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.

## Application Navigator

The ORACLE logo, consisting of the word "ORACLE" in white capital letters on a red horizontal bar.

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

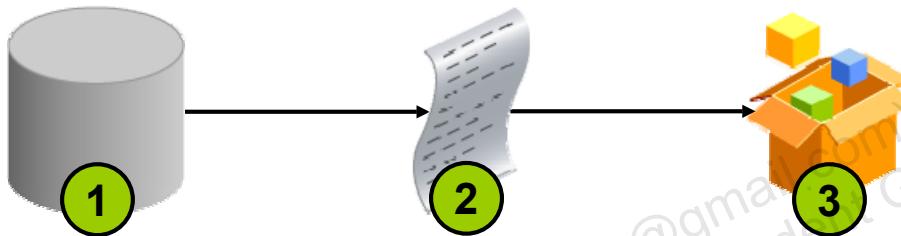
### Application Navigator

Application Navigator 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 Application Navigator 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 Unified Modeling Language (UML), Enterprise JavaBeans (EJB) o servicios web aparecen en el navegador como nodos únicos. Los archivos raw que componen estos nodos abstractos aparecen en la ventana Structure.

## Despliegue de Procedimientos Java Almacenados

Antes de desplegar procedimientos Java almacenados, realice los siguientes pasos:

1. Cree una conexión de base de datos.
2. Cree un perfil de despliegue.
3. Despliegue los objetos.



ORACLE

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

### Despliegue de Procedimientos Java Almacenados

Cree un perfil de despliegue para los procedimientos Java almacenados y, a continuación, despliegue las clases y, si lo desea, cualquier método estático público en JDeveloper con la configuración del perfil.

El despliegue en la base de datos utiliza la información proporcionada en Deployment Profile Wizard y dos utilidades de base de datos Oracle:

- `loadjava` carga la clase Java que contiene los procedimientos almacenados en una base de datos Oracle.
- `publish` genera los encapsuladores 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 screenshot shows two windows side-by-side. The left window, titled 'TrimLob.java', contains Java code for a class named 'TrimLob' with a main method. The right window, titled 'TRIMLOBPROC', contains the PL/SQL code for a procedure named 'TRIMLOBPROC' that calls the Java code. A green circle labeled '1' points to the Java code in the left window, and a green circle labeled '2' points to the PL/SQL code in the right window.

```
public class TrimLob
{
 public static void main (String args []) throws SQLException {
 Connection conn=null;
 if (System.getProperty("oracle.jserver.version") != null)
 {
 conn = DriverManager.getConnection("jdbc:default:connection:");
 }
 else
 {
 DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
 conn = DriverManager.getConnection("jdbc:oracle:thin:scott/tiger");
 }
 }
}
```

```
CREATE OR REPLACE PROCEDURE TRIMLOBPROC
as Language java
name 'TrimLob.main(java.lang.String[])';
```

ORACLE

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

### Publicación de Java en PL/SQL

En la diapositiva se muestra el código Java y se ilustra cómo publicar el código Java en un procedimiento PL/SQL.

## ¿Cómo Puedo Obtener Más Información sobre JDeveloper 11g ?

| Tema                                             | Dirección Web                                                                                                                           |
|--------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Página del Producto Oracle JDeveloper            | <a href="http://www.oracle.com/technology/products/jdev/index.html">http://www.oracle.com/technology/products/jdev/index.html</a>       |
| Tutoriales de Oracle JDeveloper 11g              | <a href="http://www.oracle.com/technology/obe/obe11jdev/11/index.html">http://www.oracle.com/technology/obe/obe11jdev/11/index.html</a> |
| Documentación del Producto Oracle JDeveloper 11g | <a href="http://www.oracle.com/technology/documentation/jdev.html">http://www.oracle.com/technology/documentation/jdev.html</a>         |
| Foro de Discusión de Oracle JDeveloper 11g       | <a href="http://forums.oracle.com/forums/forum.jspa?forumID=83">http://forums.oracle.com/forums/forum.jspa?forumID=83</a>               |



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

## Resumen

En este apéndice, debe haber aprendido cómo utilizar JDeveloper para realizar las siguientes acciones:

- Mostrar las funciones clave de Oracle JDeveloper
- Crear una conexión de base de datos en JDeveloper
- Gestionar objetos de base de datos en JDeveloper
- Utilizar JDeveloper para ejecutar comandos SQL
- Crear y ejecutar unidades de programa PL/SQL



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

## Objetivos

En este apéndice, se presentará la herramienta JDeveloper. Aprenderá cómo utilizar JDeveloper para las tareas de desarrollo de la base de datos.

# R Revisión de PL/SQL

ORACLE

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

## Objetivos

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

- Revisar la estructura en bloque de bloques PL/SQL anónimos
- Declarar variables PL/SQL
- Crear registros y tablas PL/SQL
- Insertar, actualizar y suprimir datos
- Utilizar sentencias IF, THEN y ELSIF
- Utilizar bucles básicos FOR y WHILE
- Declarar y utilizar cursos explícitos con parámetros
- Utilizar bucles FOR de cursor y cláusulas FOR UPDATE y WHERE CURRENT OF
- Detectar excepciones predefinidas y definidas por el usuario



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

### Objetivos de la Lección

Se han tratado los cursos implícitos que PL/SQL crea automáticamente al ejecutar una sentencia SQL SELECT o DML. En esta lección, aprenderá acerca de los cursos explícitos. Aprenderá a diferenciar entre cursos implícitos y explícitos. También aprenderá a declarar y controlar cursos simples y cursos con parámetros.

## Estructura en Bloque para Bloques PL/SQL Anónimos

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



Copyright © 2010, Oracle y/o sus filiales. 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 motor 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, cursoras 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;
```



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



Copyright © 2010, Oracle y/o sus filiales. 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

Declare variables para almacenar el nombre, el cargo y el salario de un nuevo empleado.

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

Copyright © 2010, Oracle y/o sus filiales. 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

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

```
dept_record departments%ROWTYPE;
```

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

```
emp_record employees%ROWTYPE;
```

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

### Ejemplos

En la primera declaración de la diapositiva se 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.

En la segunda declaración de la diapositiva se 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 job\_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;

```

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

## Creación de una Tabla PL/SQL

No existen tipos de dato predefinidos para tablas 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 -2.147.483.647 a 2.147.483.647. Por lo tanto, el valor de la clave puede ser negativo. La indexación no tiene que empezar por 1.

**Nota:** la sentencia *table.EXISTS (i)* devuelve 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: Ejemplo

La cláusula INTO es obligatoria.

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

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

Agregue nueva información de empleado a la tabla EMPLOYEES.

```
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_empid, 'HARDING', 'PU_CLERK', 30);
END;
```



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

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

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



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

Suprima filas que pertenezcan al departamento 190 de la tabla EMPLOYEES.

```
DECLARE
 v_deptid employees.department_id%TYPE := 190;
BEGIN
 DELETE FROM employees
 WHERE department_id = v_deptid;
END ;
```



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

## Supresión de Datos: Ejemplo

Suprima un trabajo concreto:

```
DECLARE
 v_jobid jobs.job_id%TYPE := 'PR_REP';
BEGIN
 DELETE FROM jobs
 WHERE job_id = v_jobid;
END;
```

## Control de Transacciones con las Sentencias **COMMIT y ROLLBACK**

- Inicie una transacción con el primer comando DML después de una sentencia COMMIT o ROLLBACK.
- Utilice las sentencias SQL COMMIT y ROLLBACK para terminar una transacción de forma explícita.



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

### Control de Transacciones con las Sentencias **COMMIT y ROLLBACK**

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 del lenguaje de manipulación de datos (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.

## Sentencias IF, THEN y ELSIF: Ejemplo

Para un valor concreto introducido, se puede devolver un valor calculado.

```
.
.
.
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;
.
.
```

Copyright © 2010, Oracle y/o sus filiales. 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 sentencias 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;
```

En la sentencia de la diapositiva se define más detalladamente 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;
```



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

### Bucle Básico: Ejemplo

El ejemplo de bucle básico que se muestra en la diapositiva 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: Ejemplo

Inserte los primeros 10 elementos de nueva línea para el pedido número 101.

```
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;
```



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

### Bucle FOR: Ejemplo

En la diapositiva se 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;
 ...

```



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

### Bucle WHILE: Ejemplo

En el ejemplo de la diapositiva, la cantidad aumenta con cada iteración del bucle hasta que la cantidad deje de ser inferior al precio máximo permitido para el elemento.

## Atributos de Cursor Implícito SQL

Puede utilizar atributos de cursor SQL para probar el resultado de las sentencias SQL.

| Atributos de Cursor SQL | Descripción                                                                                                                             |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| SQL%ROWCOUNT            | Número de filas afectadas por la sentencia SQL más reciente (valor entero)                                                              |
| SQL%FOUND               | Atributo booleano que se evalúa como TRUE si la sentencia SQL más reciente afecta a una o más filas                                     |
| SQL%NOTFOUND            | Atributo booleano que se evalúa como TRUE si la sentencia SQL más reciente no afecta a ninguna fila                                     |
| SQL%ISOPEN              | El atributo booleano siempre se evalúa como FALSE, ya que PL/SQL cierra los cursos implícitos inmediatamente después de que se ejecuten |

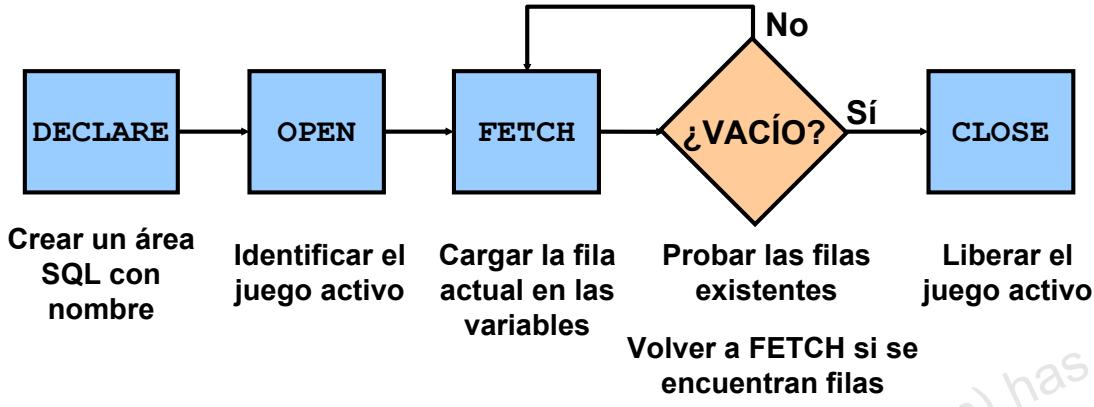
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

### Atributos de Cursor Implícito SQL

Los atributos de cursor SQL le permiten evaluar lo que ha ocurrido 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.

## Control de Cursos Explícitos



ORACLE

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

## Cursos Explícitos

### Control de Cursos 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 diapositiva, 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.

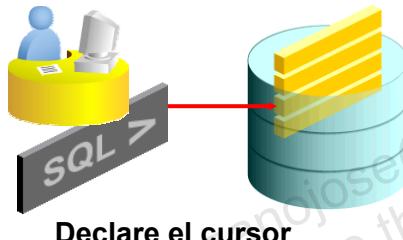
## Control de Cursos Explícitos: Declaración del Cursor

```

DECLARE
 CURSOR c1 IS
 SELECT employee_id, last_name
 FROM employees;

 CURSOR c2 IS
 SELECT *
 FROM departments
 WHERE department_id = 10;
BEGIN
 ...

```



**ORACLE**

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

### Declaración de Cursos 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
 ...

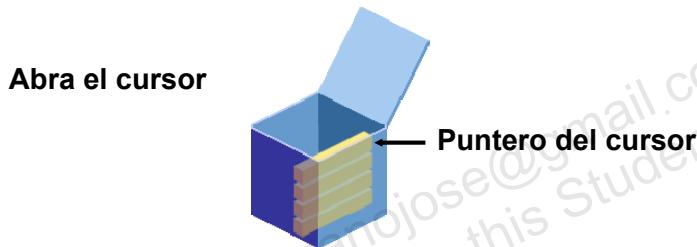
```

**Nota:** puede hacer referencia a las variables en la consulta, pero debe declararlas antes de la sentencia CURSOR.

## Control de Cursores Explícitos: Apertura del Cursor

```
OPEN cursor_name;
```

- Abra el cursor para ejecutar la consulta e identificar el juego activo.
- Si la consulta no devuelve ninguna fila, no se emite ninguna excepción.
- Utilice los atributos de cursor para probar el resultado después de una recuperación.



**ORACLE**

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

## Control de Cursores Explícitos: Recuperación de Datos del Cursor

```

FETCH c1 INTO v_empid, v_ename;

...
OPEN defined_cursor;
LOOP
 FETCH defined_cursor INTO defined_variables
 EXIT WHEN ...;
 ...
 -- Process the retrieved data
 ...
END;

```

Recupere una fila



**ORACLE**

Copyright © 2010, Oracle y/o sus filiales. 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 una variable correspondiente en la lista `INTO`. Asimismo, los tipos de dato deben ser compatibles. Puede recuperar los primeros 10 empleados de uno en uno:

```

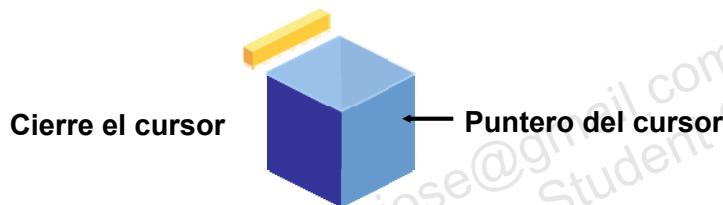
DECLARE
 v_empid employees.employee_id%TYPE;
 v_ename employees.last_name%TYPE;
 i NUMBER := 1;
 CURSOR c1 IS
 SELECT employee_id, last_name
 FROM employees;
BEGIN
 OPEN c1;
 FOR i IN 1..10 LOOP
 FETCH c1 INTO v_empid, v_ename;
 ...
 END LOOP;
END;

```

## Control de Cursores Explícitos: Cierre del Cursor

```
CLOSE cursor_name;
```

- Cierre el cursor al terminar el procesamiento de las filas.
- Vuelva a abrir el cursor si es necesario.
- No intente recuperar datos de un cursor después de cerrarlo.



**ORACLE**

Copyright © 2010, Oracle y/o sus filiales. 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, ya que 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. Por defecto, el número máximo de `OPEN_CURSORS` es 50.

```
...
FOR i IN 1..10 LOOP
 FETCH c1 INTO v_empid, v_ename; ...
END LOOP;
CLOSE c1;
END;
```

## Atributos de Cursor Explícito

Obtenga información sobre el estado de un cursor.

| Atributo  | Tipo    | Descripción                                                                                   |
|-----------|---------|-----------------------------------------------------------------------------------------------|
| ISOPEN    | BOOLEAN | Se evalúa en TRUE si el cursor está abierto                                                   |
| %NOTFOUND | BOOLEAN | Se evalúa en TRUE si la recuperación más reciente no devuelve una fila                        |
| %FOUND    | BOOLEAN | Se evalúa en TRUE 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                              |



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

Recupere empleados de uno en uno hasta que no quede ninguno.

```
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 ;
```



Copyright © 2010, Oracle y/o sus filiales. 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. En el ejemplo de la diapositiva, `emp_record` en el cursor del bucle es un registro declarado implícitamente que se utiliza en la construcción FOR LOOP.

## Cláusula FOR UPDATE: Ejemplo

Recupere los pedidos de cantidades superiores a 1.000 dólares que se han procesado hoy.

```
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;
```



Copyright © 2010, Oracle y/o sus filiales. 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 de 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;
```



Copyright © 2010, Oracle y/o sus filiales. 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

- Haga 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



Copyright © 2010, Oracle y/o sus filiales. 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

```
BEGIN SELECT ... COMMIT;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 statement1;
 statement2;
 WHEN TOO_MANY_ROWS THEN
 statement1;
 WHEN OTHERS THEN
 statement1;
 statement2;
 statement3;
END;
```



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

### Detección de Excepciones Predefinidas del Servidor de Oracle: Ejemplo

En el ejemplo de la diapositiva, se imprime un mensaje para el usuario por cada excepción. Sólo se emite y maneja una excepción cada vez.

## Error No Predefinido

Detecte el error del servidor de Oracle número -2292, que es una violación de la restricción de integridad.

```

DECLARE
 1 e_products_invalid EXCEPTION;
 PRAGMA EXCEPTION_INIT (
 2 e_products_invalid, -2292);
 v_message VARCHAR2(50);
BEGIN
 . . .
 3 EXCEPTION
 WHEN e_products_invalid THEN
 :g_message := 'Product ID
 specified is not valid.';
 . . .
END;

```

ORACLE

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

|                     |                                                       |
|---------------------|-------------------------------------------------------|
| <i>exception</i>    | Es la excepción declarada anteriormente               |
| <i>error_number</i> | 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 diapositiva: 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; 1
 .
 .
 BEGIN
 .
 .
 RAISE e_amount_remaining; 2
 .
 .
EXCEPTION
 WHEN e_amount_remaining THEN
 :g_message := 'There is still an amount
 in stock.';
 .
 .
END;
```

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

**ORACLE**

### 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;  
**dónde:** exception es el nombre de la excepción
2. Utilice la sentencia RAISE para emitir la excepción explícitamente en la sección ejecutable.  
**Sintaxis:** RAISE exception;  
**dónde:** 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 diapositiva: 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 la sentencia RAISE sola en un manejador de excepciones para volver a emitir la misma excepción en el entorno de llamada.

## Procedimiento RAISE\_APPLICATION\_ERROR

```
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



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

### Procedimiento RAISE\_APPLICATION\_ERROR

Utilice el procedimiento RAISE\_APPLICATION\_ERROR para 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 -20.000 y -20.999. *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



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

## Resumen

En este apéndice, debe haber aprendido lo siguiente:

- Revisar la estructura en bloque de bloques PL/SQL anónimos
- Declarar variables PL/SQL
- Crear registros y tablas PL/SQL
- Insertar, actualizar y suprimir datos
- Utilizar sentencias IF, THEN y ELSIF
- Utilizar bucles básicos FOR y WHILE
- Declarar y utilizar cursores explícitos con parámetros
- Utilizar bucles FOR de cursor y cláusulas FOR UPDATE y WHERE CURRENT OF
- Detectar excepciones predefinidas y definidas por el usuario

**ORACLE**

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

### Resumen

El servidor de Oracle utiliza áreas de trabajo para ejecutar sentencias SQL y almacenar la información de procesamiento. Puede utilizar una construcción PL/SQL denominada *cursor* para asignar un nombre a un área de trabajo y acceder a la información almacenada. Hay dos tipos de cursores: implícitos y explícitos. PL/SQL declara implícitamente un cursor para todas las sentencias de manipulación de datos SQL, incluidas las consultas que devuelven sólo una fila. Para las consultas que devuelven más de una fila, debe declarar explícitamente un cursor para procesar las filas de forma individual.

Cada cursor explícito y variable de cursor tiene cuatro atributos: %FOUND, %ISOPEN, %NOTFOUND y %ROWCOUNT. Cuando se agregan al nombre de la variable del cursor, estos atributos devuelven información útil sobre la ejecución de una sentencia SQL. Puede utilizar los atributos de cursor en sentencias de procedimiento, pero no en sentencias SQL.

Utilice bucles simples o bucles FOR de cursor para trabajar en varias filas recuperadas por el cursor. Si utiliza bucles sencillos, tiene que abrir, recuperar y cerrar el cursor; sin embargo, los bucles FOR de cursor lo realizan implícitamente. Si actualiza o suprime filas, bloquee las filas mediante una cláusula FOR UPDATE. Esto garantiza que los datos que utiliza no se actualicen en otra sesión después de que abra el cursor. Utilice una cláusula WHERE CURRENT OF junto con la cláusula FOR UPDATE para hacer referencia a la fila actual recuperada por el cursor.

## **Estudios para Implementación de Disparadores**



**ORACLE**

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

Jose Enrique Lozano (lozanojose@gmail.com) has a  
non-transferable license to use this Student Guide.

## Objetivos

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

- Mejorar la seguridad de la base de datos con disparadores
- 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



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

### Objetivos de la Lección

En esta lección, aprenderá a desarrollar disparadores de base de datos para mejorar las funciones que, de lo contrario, no podría implantar 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;
```



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

- Base los privilegios en el nombre de usuario proporcionado cuando el usuario se conecta a la base de datos.
- Determine el acceso a tablas, vistas, sinónimos y secuencias.
- Determine 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;
/
```



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

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

Desarrolle disparadores para manejar requisitos de seguridad más complejos.

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

## Forzado de Integridad de Datos en el Servidor

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



Copyright © 2010, Oracle y/o sus filiales. 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 diapositiva 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;
/
```



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

- Proporcione valores por defecto variables.
- Fuerce restricciones dinámicas.
- Active y desactive dinámicamente.
- Incorpore restricciones declarativas en la definición de una tabla para proteger la integridad de los datos.

#### Ejemplo

El ejemplo de código de la diapositiva 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;
```



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

- Restrinja las actualizaciones y supresiones.
- Realice supresiones en cascada.
- Active y desactive 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 ;
/
```



Copyright © 2010, Oracle y/o sus filiales. 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 implantar 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;
```



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

En el ejemplo de la diapositiva se 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;

```

Copyright © 2010, Oracle y/o sus filiales. 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);
```



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

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

```

Copyright © 2010, Oracle y/o sus filiales. 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 este apéndice, debe haber aprendido lo siguiente:

- Mejorar la seguridad de la base de datos con disparadores
- 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



Copyright © 2010, Oracle y/o sus filiales. 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 implantar las operaciones de seguridad, auditoría, integridad de los datos, replicación y registro. También trata la forma en que se pueden utilizar los disparadores de base de datos para implantar 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 implantar este tipo de regla de negocio sin algún esfuerzo de programación.

Jose Enrique Lozano (lozanojose@gmail.com) has a  
non-transferable license to use this Student Guide.

# **Uso de los Paquetes DBMS\_SCHEDULER y HTP**

**ORACLE**

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

## Objetivos

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

- Utilizar el paquete `HTP` para generar una página web sencilla
- Llamar al paquete `DBMS_SCHEDULER` para planificar el código PL/SQL para su ejecución



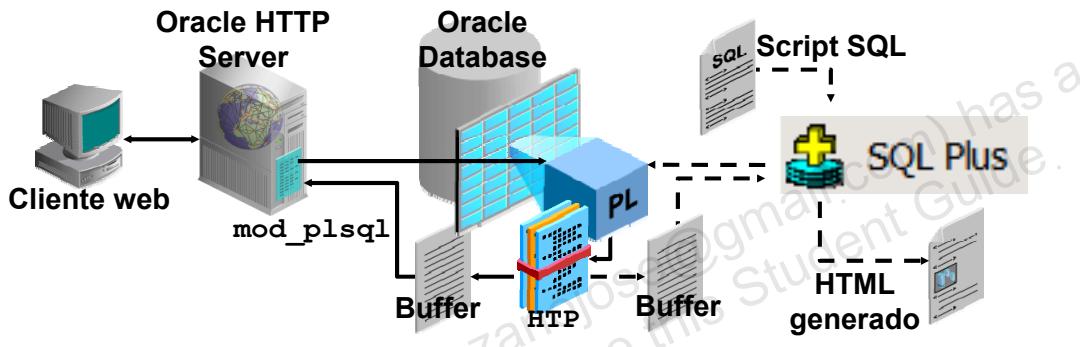
Copyright © 2010, Oracle y/o sus filiales. Todos los derechos reservados.

### Objetivos de la Lección

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

## Generación de Páginas Web con el Paquete HTP

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



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

ORACLE

## Generación de Páginas Web con el Paquete HTP

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

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

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

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

## Uso de los Procedimientos de Paquete HTP

- Genere una o más etiquetas HTML. Por ejemplo:

```
htp.bold('Hello'); -- Hello
htp.print('Hi World'); -- Hi World
```

- Utilizado para crear un documento HTML con formato correcto:

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

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

### Uso de los Procedimientos de Paquete HTP

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

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

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

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

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

## Creación de un Archivo HTML con SQL\*Plus

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

1. Cree un script SQL con los siguientes comandos:

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

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



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

### Creación de un Archivo HTML con SQL\*Plus

En el ejemplo de la diapositiva se muestran los pasos para generar HTML utilizando cualquier procedimiento y guardando la salida en un archivo HTML. Realice los siguientes pasos:

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

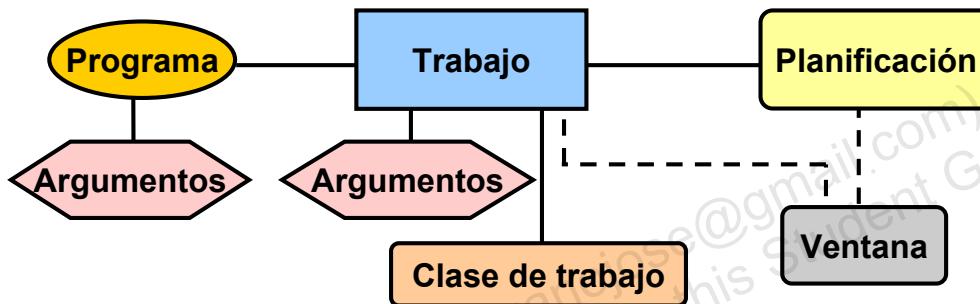
El comando ACCEPT solicita el nombre del procedimiento que se va a ejecutar. La llamada a OWA\_UTIL.SHOWPAGE muestra las etiquetas HTML en la ventana del explorador. A continuación, podrá copiar y pegar las etiquetas HTML generadas desde la ventana del explorador al archivo HTML, normalmente con una extensión .htm o .html.

**Nota:** si se utiliza SQL\*Plus, se puede usar el comando SPOOL para dirigir la salida HTML directamente a un archivo HTML.

## Paquete DBMS\_SCHEDULER

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

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



**ORACLE**

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

### Paquete DBMS\_SCHEDULER

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

El diagrama muestra los siguientes componentes arquitectónicos del planificador:

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

## Paquete DBMS\_SCHEDULER (continuación)

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

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

Para obtener más información sobre el planificador, consulte el curso *Oracle Database 11g: Configure and Manage Jobs with the Scheduler* (Oracle Database 11g: Configuración y Gestión de Trabajos con el Planificador).

## Creación de un Trabajo

- Un trabajo se puede crear de varias maneras utilizando una combinación de parámetros en línea Programs y Schedules con nombre.
- Para crear un trabajo con el procedimiento CREATE\_JOB:
  - Utilice la información en línea con el “qué” y la planificación especificados como parámetros
  - Utilice un programa con nombre (guardado) y especifique la planificación en línea
  - Especifique lo que se debe realizar en línea y utilice una planificación con nombre
  - Utilice componentes de programa y planificación con nombre



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

### Creación de un Trabajo

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

- Proporcione un nombre con el formato [schema.] name
- Necesita el privilegio CREATE JOB

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

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

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

## Creación de un Trabajo con Parámetros en Línea

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

```
-- Schedule a PL/SQL block every hour:

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



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

## Creación de un Trabajo con Parámetros en Línea

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

En el ejemplo de la diapositiva se muestra cómo se pueden especificar todos los detalles del trabajo en línea. Los parámetros del procedimiento CREATE\_JOB definen “qué” se tiene que ejecutar, la planificación y otros atributos del trabajo. Los siguientes parámetros definen lo que se debe ejecutar:

- El parámetro job\_type puede ser uno de los tres siguientes valores:
  - PLSQL\_BLOCK para cualquier bloque PL/SQL o sentencia SQL. Este tipo de trabajo no puede aceptar argumentos.
  - STORED\_PROCEDURE para cualquier procedimiento autónomo o empaquetado almacenado. Los procedimientos pueden aceptar argumentos que se proporcionan con el trabajo.
  - EXECUTABLE para una aplicación ejecutable del sistema operativo de línea de comandos.
- La planificación se especifica utilizando los siguientes parámetros:
  - start\_date acepta un registro de hora y repeat\_interval especifica una cadena como calendario o expresión PL/SQL. Se puede especificar end\_date.

**Nota:** las expresiones de cadena que se especifican para repeat\_interval se describirán más adelante. El ejemplo especifica que el trabajo se debe ejecutar cada hora.

## Creación de un Trabajo Utilizando un Programa

- Utilice CREATE\_PROGRAM para crear un programa:

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

- Utilice el procedimiento sobrecargado CREATE\_JOB con el parámetro program\_name:

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

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

### Creación de un Trabajo Utilizando un Programa

El procedimiento DBMS\_SCHEDULER.CREATE\_PROGRAM define un programa al que se debe asignar un nombre único. Crear el programa por separado para el trabajo le permitirá:

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

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

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

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

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

Para crear un trabajo con un programa, especifique el nombre del programa en el argumento program\_name en la llamada al procedimiento DBMS\_SCHEDULER.CREATE\_JOB, como se muestra en la diapositiva.

## Creación de un Trabajo para un Programa con Argumentos

- Cree un programa:

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

- Defina un argumento:

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

- Cree un trabajo especificando el número de argumentos:

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

**ORACLE**

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

### Creación de un Trabajo para un Programa con Argumentos

Puede que algunos procedimientos PL/SQL o externos necesiten argumentos de entrada. Si utiliza el procedimiento DBMS\_SCHEDULER.DEFINE\_PROGRAM\_ARGUMENT, puede definir un argumento para un programa existente. Entre los parámetros del procedimiento

DEFINE\_PROGRAM\_ARGUMENT se incluyen:

- program\_name especifica un programa existente que hay que modificar.
- argument\_name especifica un nombre de argumento único para el programa.
- argument\_position especifica la posición en la que el argumento se transfiere cuando se llama al programa.
- argument\_type especifica el tipo de dato del valor del argumento que se transfiere al programa llamado.
- default\_value especifica un valor por defecto que se proporciona al programa, si el trabajo que planifica el programa no proporciona un valor.

En la diapositiva se muestra cómo crear un trabajo ejecutando un programa con un argumento. El valor por defecto del argumento del programa es 50. Para cambiar el valor del argumento del programa, utilice:

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

## Creación de un Trabajo Utilizando una Planificación

- Use `CREATE_SCHEDULE` para crear una planificación:

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

- Utilice `CREATE_JOB` haciendo referencia a la planificación en el parámetro `schedule_name`:

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

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

### Creación de un Trabajo Utilizando una Planificación

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

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

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

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

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

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

## Definición del Intervalo de Repetición para un Trabajo

- Mediante una expresión de calendario:

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

- Mediante una expresión PL/SQL:

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



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

## Creación de un Trabajo Utilizando un Programa y una Planificación con Nombre

- Cree un programa con nombre denominado PROG\_NAME utilizando el procedimiento CREATE\_PROGRAM.
- Cree una planificación con nombre denominada SCHED\_NAME utilizando el procedimiento CREATE\_SCHEDULE.
- Cree un trabajo que haga referencia al programa y a la planificación con nombre:

```
BEGIN
 DBMS_SCHEDULER.CREATE_JOB('JOB_NAME',
 program_name => 'PROG_NAME',
 schedule_name => 'SCHED_NAME',
 enabled => TRUE);
END;
/
```



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

### Creación de un Trabajo Utilizando un Programa y una Planificación con Nombre

En el ejemplo de la diapositiva se muestra la última forma de utilizar el procedimiento DBMS\_SCHEDULER.CREATE\_JOB. En este ejemplo, el programa (PROG\_NAME) y la planificación (SCHED\_NAME) con nombre se especifican en los respectivos parámetros en la llamada al procedimiento DBMS\_SCHEDULER.CREATE\_JOB.

Con este ejemplo, puede ver lo fácil que es crear trabajos utilizando un programa y una planificación predefinidos.

Algunos trabajos y planificaciones son demasiado complejos para tratarlos en este curso. Por ejemplo, puede crear ventanas para planes de intervalos periódicos y asociar un plan de recursos a una ventana. Un plan de recursos define atributos sobre los recursos necesarios durante el período definido por la ventana de ejecución.

Para obtener más información, consulte el curso en línea *Oracle Database 11g: Configure and Manage Jobs with the Scheduler* (Oracle Database 11g: Configuración y Gestión de Trabajos con el Planificador).

## Gestión de Trabajos

- Ejecute un trabajo:  

```
DBMS_SCHEDULER.RUN_JOB('SCHEMA.JOB_NAME');
```
- Pare un trabajo:  

```
DBMS_SCHEDULER.STOP_JOB('SCHEMA.JOB_NAME');
```
- Borre un trabajo, aunque se esté ejecutando actualmente:  

```
DBMS_SCHEDULER.DROP_JOB('JOB_NAME' , TRUE);
```



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

### Gestión de Trabajos

Una vez creado un trabajo, podrá:

- Ejecutar el trabajo llamando al procedimiento RUN\_JOB especificando el nombre del trabajo. El trabajo se ejecuta inmediatamente en la sesión actual.
- Pare el trabajo utilizando el procedimiento STOP\_JOB. Si el trabajo se está ejecutando actualmente, se parará de inmediato. El procedimiento STOP\_JOB tiene dos argumentos:
  - **job\_name**: es el nombre del trabajo que se va a parar
  - **force**: intenta terminar el trabajo de forma correcta. Si esto falla y force está definido en TRUE, se termina el esclavo del trabajo. (El valor por defecto es FALSE.) Para utilizar force, debe tener el privilegio del sistema MANAGE SCHEDULER.
- Borre el trabajo con el procedimiento DROP\_JOB. Este procedimiento tiene dos argumentos:
  - **job\_name**: es el nombre de la función que se va a borrar
  - **force**: indica si el trabajo se debe parar y borrar en caso de estar ejecutándose en esos momentos (el valor por defecto es FALSE.)

Si se llama al procedimiento DROP\_JOB y el trabajo especificado se está ejecutando en esos momentos, el comando fallará, a menos que la opción force se haya definido en TRUE. Si la opción force se ha definido en TRUE, se pararán las instancias del trabajo que se estén ejecutando y se borrará el trabajo.

**Nota:** para ejecutar, parar o borrar un trabajo que pertenezca a otro usuario, necesita los privilegios ALTER de ese trabajo o el privilegio del sistema CREATE ANY JOB.

## Vistas del Diccionario de Datos

- [DBA | ALL | USER]\_SCHEDULER\_JOBS
- [DBA | ALL | USER]\_SCHEDULER\_RUNNING\_JOBS
- [DBA | ALL]\_SCHEDULER\_JOB\_CLASSES
- [DBA | ALL | USER]\_SCHEDULER\_JOB\_LOG
- [DBA | ALL | USER]\_SCHEDULER\_JOB\_RUN\_DETAILS
- [DBA | ALL | USER]\_SCHEDULER\_PROGRAMS



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

### Vistas del Diccionario de Datos

La vista DBA\_SCHEDULER\_JOB\_LOG muestra todas las instancias de trabajo terminadas, tanto las correctas como las fallidas.

Para ver el estado de los trabajos, utilice la siguiente consulta:

```
SELECT job_name, program_name, job_type, state
FROM USER_SCHEDULER_JOBS;
```

Utilice la siguiente consulta para determinar en qué instancia se está ejecutando un trabajo:

```
SELECT owner, job_name, running_instance,
resource_consumer_group
FROM DBA_SCHEDULER_RUNNING_JOBS;
```

Para determinar información sobre cómo se ejecutó un trabajo, utilice la siguiente consulta:

```
SELECT job_name, instance_id, req_start_date, actual_start_date,
status
FROM ALL_SCHEDULER_JOB_RUN_DETAILS;
```

Para determinar el estado de los trabajos, utilice la siguiente consulta:

```
SELECT job_name, status, error#, run_duration, cpu_used
FROM USER_SCHEDULER_JOB_RUN_DETAILS;
```

## Resumen

En este apéndice, debe haber aprendido lo siguiente:

- Utilizar el paquete HTP para generar una página web sencilla
- Llamar al paquete DBMS\_SCHEDULER para planificar el código PL/SQL para su ejecución



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

### Resumen

Esta lección abarca un pequeño subjuego de paquetes que se proporciona con la base de datos Oracle. Ya ha utilizado ampliamente DBMS\_OUTPUT para la depuración y para mostrar información generada con procedimientos en la pantalla de SQL\*Plus.

En esta lección, debe haber aprendido a planificar PL/SQL y código externo para su ejecución con el paquete DBMS\_SCHEDULER.

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

Jose Enrique Lozano (lozanojose@gmail.com) has a  
non-transferable license to use this Student Guide.