

2

Creación de procedimientos



ORACLE®

Objetivos

Después de completar esta lección, usted debería ser capaz de:

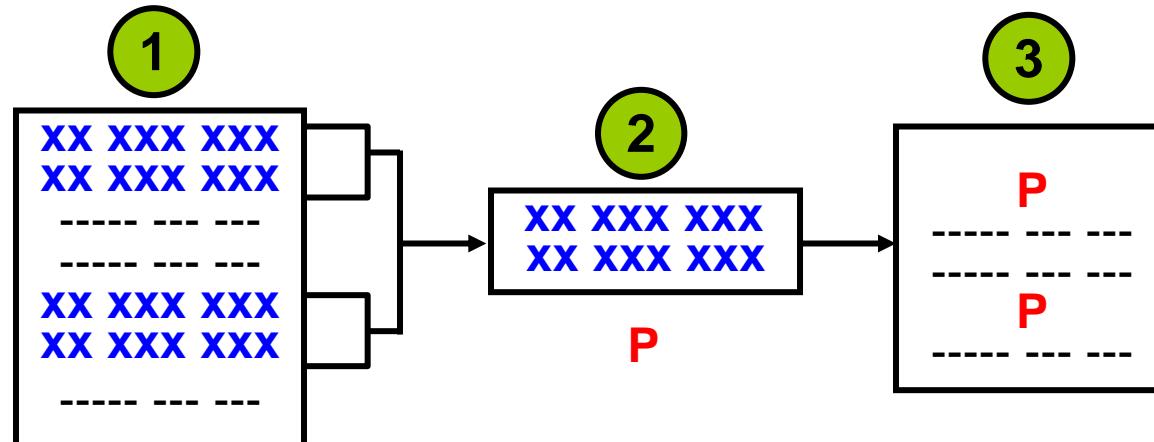
- Identificar los beneficios del diseño de subprogramas modularizados y en capas
- Crear y llamar procedimientos
- Utilizar parámetros formales y reales
- Usar notación posicional, nombrada o mixta para pasar parámetros
- Identificar los modos de paso de parámetros disponibles
- Manejar excepciones en los procedimientos
- Eliminar un procedimiento y mostrar su información

Agenda

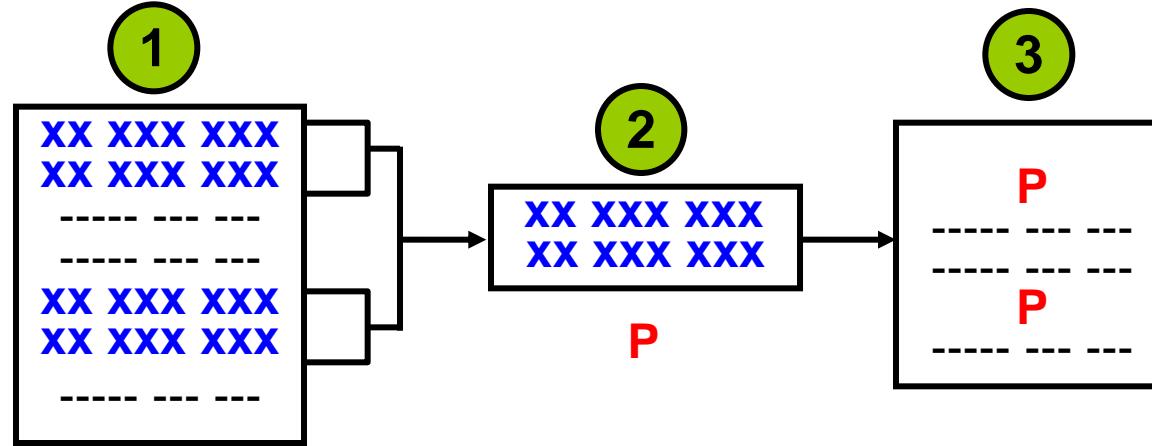
- Utilizando un diseño de subprograma modularizado y en capas e identificando los beneficios de los subprogramas
- Trabajar con procedimientos:
 - Creación y llamada de procedimientos
 - Identificación de los modos de paso de parámetros disponibles
 - Usando parámetros formales y reales
 - Uso de la notación posicional, nombrada o mixta
- Manejo de excepciones en los procedimientos, eliminación de un procedimiento y visualización de la información del procedimiento

Creación de un diseño de subprograma modulado

- El diagrama ilustra el principio de modularización con subprogramas:
 - La creación de piezas manejables más pequeñas de código flexible y reutilizable.
- La flexibilidad se logra mediante el uso de **subprogramas** con parámetros, que a su vez hace que el mismo código sea reutilizable para diferentes valores de entrada.



Creación de un diseño de subprograma modulado



- Para modularizar el código existente, realice los pasos siguientes:
 - 1.- Localizar e identificar secuencias repetitivas de código.
 - 2.- Mueva el código repetitivo a un subprograma PL/SQL.
 - 3.- Reemplace el código repetitivo original por llamadas al nuevo subprograma PL/SQL.

Creación de un diseño de subprograma en capas

- Debido a que PL/SQL permite que las sentencias de SQL se incrusten perfectamente en la lógica, es habitual tener sentencias de SQL extendida por todo el código.
- Sin embargo, se recomienda mantener la lógica SQL separada de la lógica de negocio. Es decir, crear un diseño de aplicación en capas con un mínimo de dos capas:
 - Capa de acceso a datos:
 - Para que las subrutinas accedan a los datos mediante sentencias SQL
 - Lógica de negocio:
 - Para que los subprogramas implementen las reglas de procesamiento de negocios, que pueden o no llamar a las rutinas de la capa de acceso a datos

Modularización del desarrollo con bloques PL/SQL

- Un subprograma se basa en estructuras PL/SQL básicas.
- Contiene:
 - una sección **declarativa**
 - una sección **ejecutable** y
 - una sección opcional de manejo de **excepciones** (por ejemplo, **bloques anónimos, procedimientos, funciones, paquetes y disparadores**).
- Los subprogramas pueden:
 - Ser compilados y almacenados en la base de datos
 - Proporcionar:
 - **Modularidad** Convierte grandes bloques de código en grupos más pequeño
 - **Extensibilidad** Los módulos se pueden ampliar fácilmente
 - **Reutilización** Los módulos pueden ser reutilizados por el mismo programa o compartidos con otros programas
 - **Mantenibilidad.** Es más fácil mantener y depurar código

Bloques anónimos: descripción general

- Los bloques anónimos se usan típicamente para:
 - Escribir código de activación para los componentes de Oracle Forms
 - Iniciando llamadas a **procedimientos, funciones y construcciones de paquetes**
 - Aislar el manejo de excepciones dentro de un bloque de código
Anidamiento dentro de otros bloques PL/SQL para gestionar el control de flujo de código

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

Bloques anónimos: descripción general

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

- La palabra clave `DECLARE` es opcional, pero se requiere si declara variables, constantes y excepciones que se utilizarán dentro del bloque PL/SQL.
- `BEGIN` y `END` son obligatorios y requieren al menos una sentencia entre ellos, SQL, PL/SQL o ambos.
- La sección de excepciones es opcional y se utiliza para manejar errores que se producen dentro del ámbito del bloque PL/SQL.
 - Las excepciones se pueden propagar o ser tratadas en el propio bloque

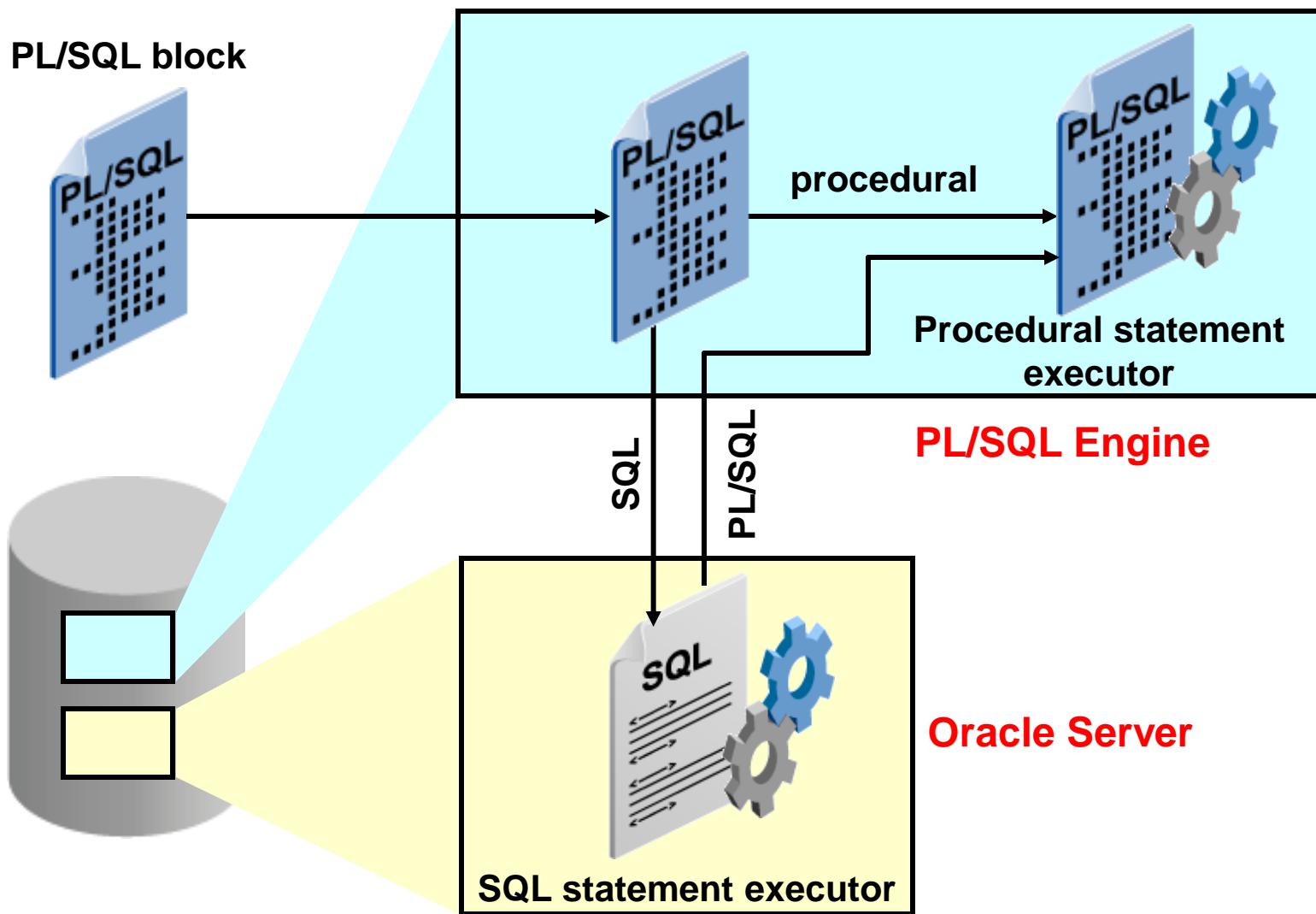
Arquitectura de PL/SQL Runtime

- Todas las ejecuciones de bloques PL/SQL son realizadas dentro del motor de PLSQL.
- Todas las sentencias SQL deben ser enviados al Motor de SQL (SQL Statement Executor)
- El motor PL/SQL es una **máquina virtual** que reside en la memoria y procesa las instrucciones del m-code PL/SQL.
- Cuando el motor PL/SQL encuentra una instrucción SQL, se hace un **cambio de contexto** para pasar la instrucción SQL a los procesos del servidor Oracle

Arquitectura de PL/SQL Runtime

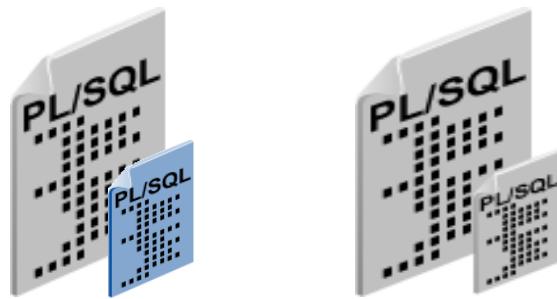
- El motor PL/SQL reside en:
 - La **base de datos Oracle** para ejecutar **subprogramas almacenados**
 - El **cliente Oracle Forms** cuando se ejecutan aplicaciones cliente/servidor o en *Oracle Application Server* cuando se utilizan *Oracle Forms Services* para ejecutar formularios en la Web.
- El motor de SQL reside en:
 - Servidor de Oracle y utiliza los diferentes procesos del Servidor de Oracle para su proceso

Arquitectura de PL/SQL Runtime



¿Qué son los subprogramas PL/SQL?

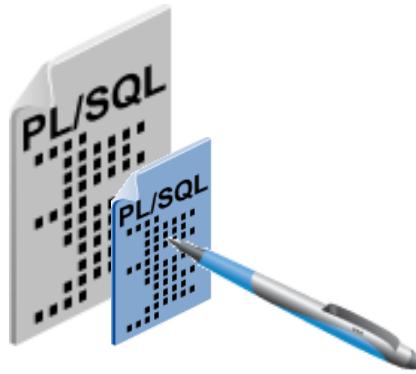
- Un subprograma PL/SQL es un bloque nominado PL/SQL que puede ser llamado con un conjunto de parámetros.
- Puede declarar y definir un subprograma dentro de un bloque PL/SQL u otro subprograma.
- Un subprograma consiste en una especificación y un cuerpo.
- Un subprograma puede ser un [procedimiento](#) o una [función](#).
- Normalmente, se utiliza un [procedimiento](#) para realizar una acción y una [función](#) para calcular y devolver un valor.
- Los subprogramas pueden agruparse en [paquetes PL/SQL](#).



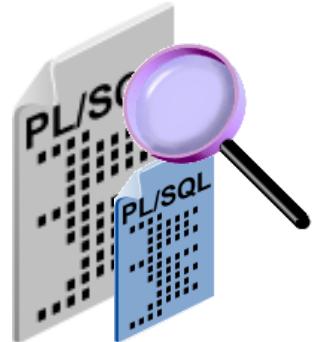
Los beneficios del uso de subprogramas PL/SQL

- *Procedimientos y funciones* tienen muchos beneficios debido a la modularización de código:
 - **Fácil Mantenimiento**
 - Los subprogramas son fáciles de mantener por encontrarse localizados en un lugar.
 - Las modificaciones deben realizarse en un solo lugar para afectar múltiples aplicaciones y minimizar las pruebas excesivas.
 - **Seguridad**
 - Los datos se pueden controlar a través de control accesos de forma indirecta a objetos de base de datos de usuarios no privilegiados con privilegios de seguridad.
 - **Integridad de Datos**
 - La integridad de los datos se gestiona teniendo acciones relacionadas ejecutadas juntas o no
 - **Rendimiento**
 - El código PL / SQL analizado que se encuentra disponible en el área de SQL compartida del servidor
 - Las llamadas subsiguientes al subprograma evitan volver a analizar el código.
 - **Claridad de código**
 - La utilización de nombres y convenciones apropiados para describir la acción de las rutinas, mejoran la claridad del código.

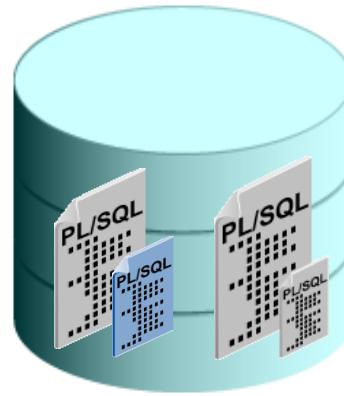
Los beneficios del uso de subprogramas PL/SQL



Fácil
Mantenimiento



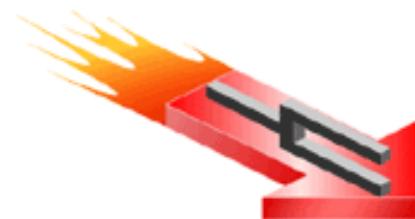
Claridad de código



Subprogramas:
Procedimientos
almacenados y
funciones



Seguridad e Integridad
de Datos



Rendimiento

Diferencias entre bloques y subprogramas anónimos

Bloques anónimos	Subprogramas
Bloques PL/SQL sin nombre	Bloques PL/SQL nombrados
Compilado cada vez	Compilado solo una vez
No almacenado en la base de datos	Almacenado en la base de datos
No puede ser invocado por otras aplicaciones	Nombrado y, por lo tanto, puede ser invocado por otras aplicaciones
No devuelve valores	Las funciones deben devolver algún valor.
No pueden pasarle parámetros	Puede tomar parámetros

Agenda

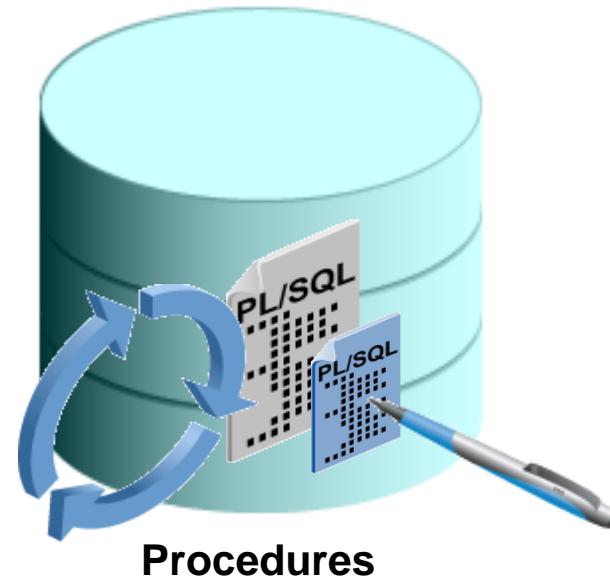
- Utilizando un diseño de subprograma modularizado y en capas e identificando los beneficios de los subprogramas
- Trabajar con procedimientos:
 - Creación y llamada de procedimientos
 - Identificación de los modos de paso de parámetros disponibles
 - Usando parámetros formales y reales
 - Uso de la notación posicional, nombrada o mixta
- Manejo de excepciones en los procedimientos, eliminación de un procedimiento y visualización de la información del procedimiento

¿Qué son los procedimientos?

- Un procedimiento es un bloque denominado PL/SQL que puede aceptar parámetros (a veces denominados argumentos).
- Generalmente, se utiliza un procedimiento para realizar una acción.
 - Tiene:
 - una **cabecera**
 - una sección **declarativa**
 - una sección **ejecutable**
 - una sección **opcional** de manejo de **excepciones**.
- El procedimiento se invoca utilizando el nombre del procedimiento en la sección de ejecución de otro bloque PL/SQL.

¿Qué son los procedimientos?

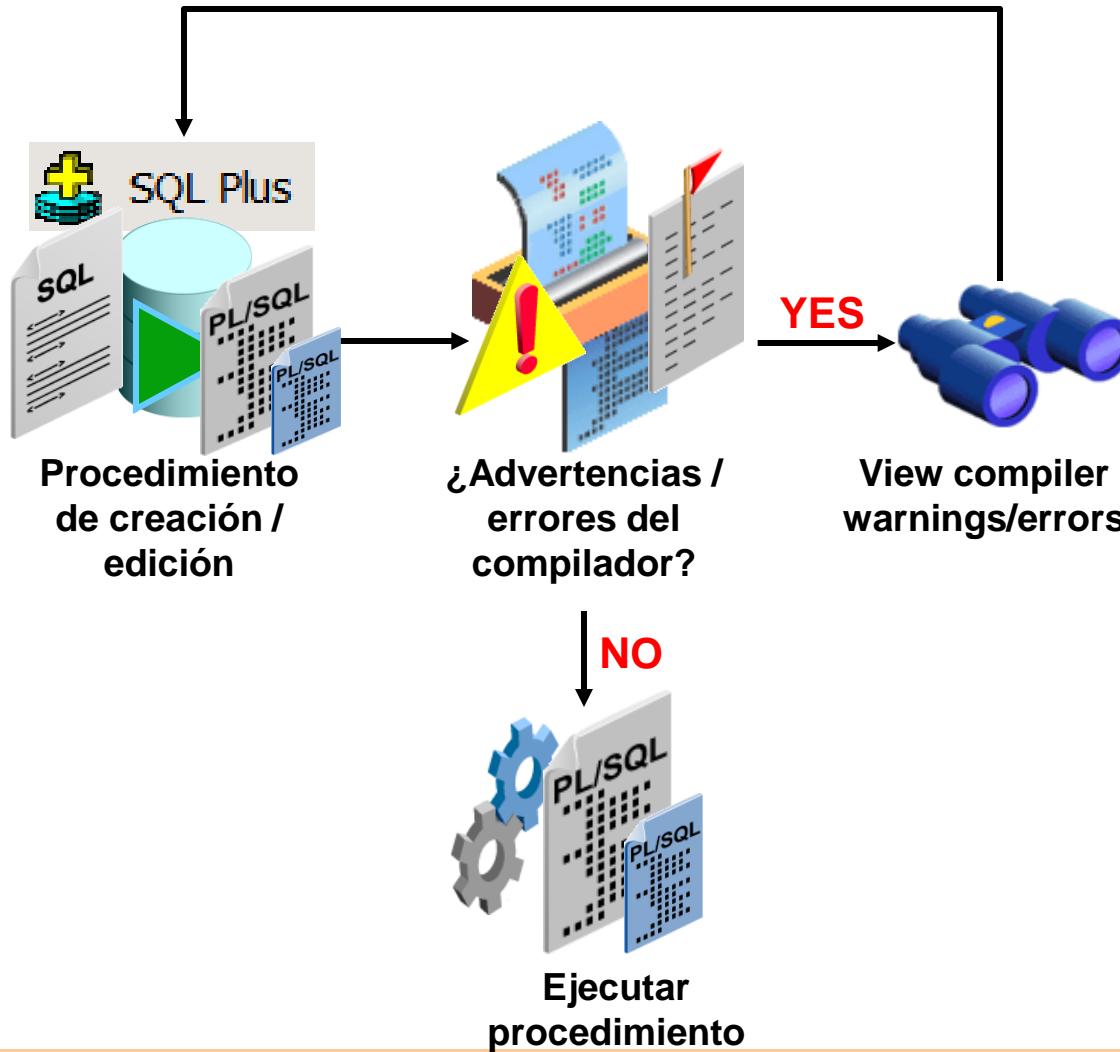
- Un procedimiento se compila y almacena en la base de datos como un **objeto de esquema**.
 - Si está utilizando los procedimientos con Oracle Forms and Reports, se pueden compilar en los ejecutables de Oracle Forms o Oracle Reports.
- Los procedimientos promueven la reutilización y la capacidad de mantenimiento.



Creación de procedimientos: descripción general

- Para desarrollar un procedimiento utilizando una herramienta como SQL Developer, realice los pasos siguientes:
 1. **Cree el procedimiento**
 - Utilizando el árbol del Navegador de objetos de SQL Developer o el área de la hoja de cálculo SQL.
 2. **Compile el procedimiento.**
 - El procedimiento se crea en la base de datos y se compila.
 - La instrucción **CREATE PROCEDURE** crea y almacena el código fuente y el código m compilado en la base de datos.
 3. **Si existen errores de compilación**
 - El **m-code** no se almacena y debe editar el código fuente para realizar correcciones. **SHOW ERRORS**
 - No puede invocar un procedimiento que contenga errores de compilación..
 4. Despues de la compilación exitosa, **ejecute el procedimiento**
 - Puede ejecutar el procedimiento utilizando SQL Developer o utilizar el comando **EXECUTE** en SQL * Plus.

Creación de procedimientos: descripción general



Creación de procedimientos con la instrucción SQL CREATE OR REPLACE

- Puede utilizar la instrucción SQL **CREATE PROCEDURE** para crear procedimientos autónomos que se almacenan en una base de datos Oracle.
- La opción **REPLACE** indica que si el procedimiento existe, se elimina y se reemplaza con la nueva versión creada por la sentencia. La opción **REPLACE** no suprime ningun procedimiento.

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

PL/SQL block

Creación de procedimientos con la instrucción SQL CREATE OR REPLACE

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

PL/SQL block

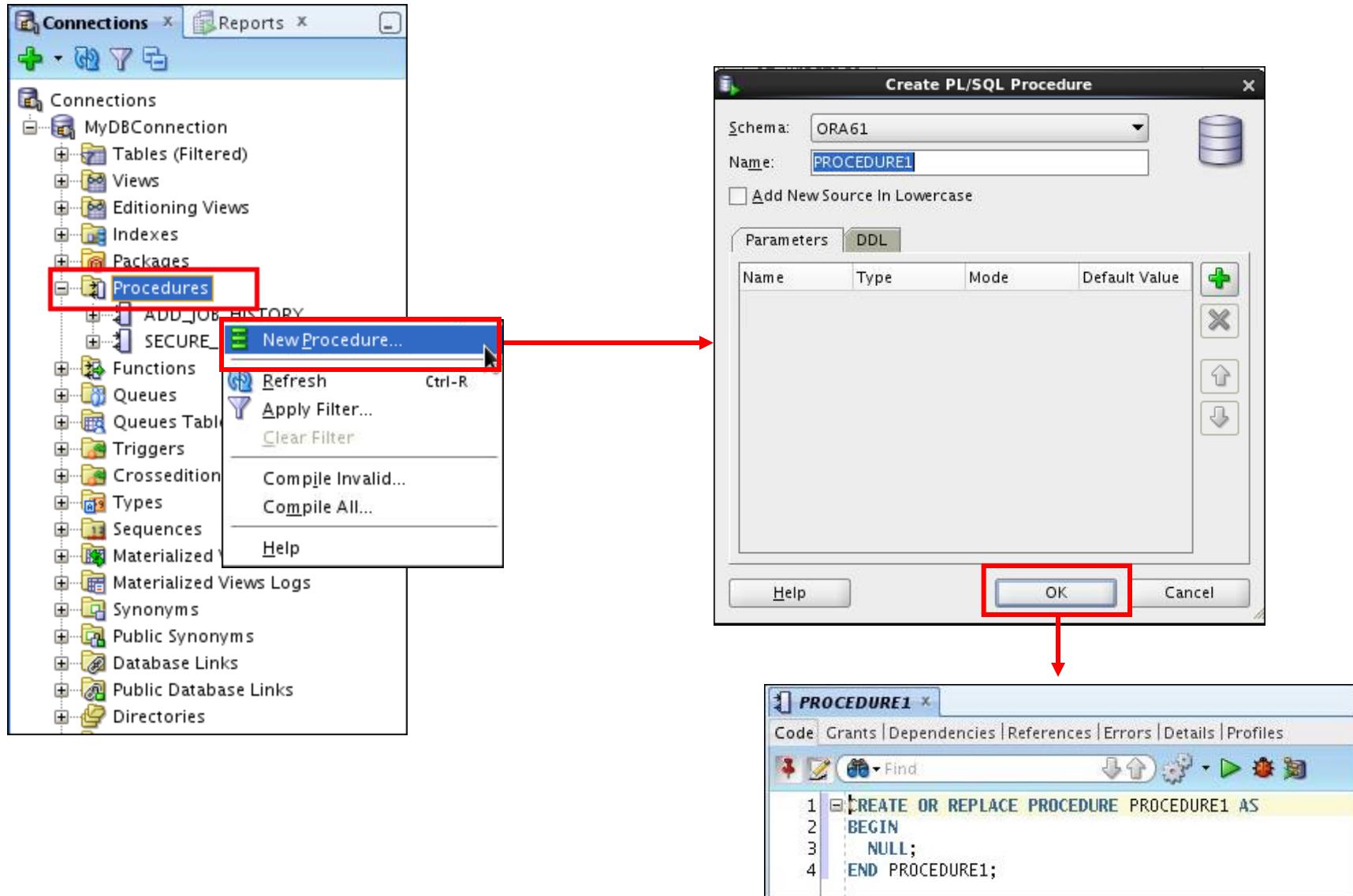
- Otros elementos sintácticos:

- Parámetro Representa el nombre de un parámetro.
- Mode Define cómo se usa un parámetro: IN (predeterminado), OUT o IN OUT.
- Datatype Especifica el tipo de datos del parámetro, **sin ninguna precisión**.
- IS/AS Indica el comienzo del procedimiento

NOTA:

Los parámetros pueden considerarse como variables locales en el procedimiento.

Creación de procedimientos en SQL Developer



Creación de procedimientos en SQL Developer

Para crear un procedimiento utilizando SQL Developer, realice los pasos siguientes:

1. Haga clic con el botón derecho en el nodo Procedimientos en la pestaña Conexiones.
2. Seleccione Nuevo procedimiento en el menú contextual. Aparecerá el cuadro de diálogo Crear procedimiento PL/SQL
3. Especifique la información para el nuevo procedimiento
4. Haga clic en Aceptar para crear el subprograma y hacer que se muestre en la ventana Editor, donde puede introducir los detalles

Los componentes del cuadro de diálogo Crear procedimiento PL/SQL son los siguientes:

- Esquema
- Nombre
- Código
- Parámetros

Procedimientos de compilación en SQL Developer (2 métodos)

The diagram illustrates two methods for compiling a procedure in SQL Developer:

- Method 1 (Left):** Right-click on the procedure node in the Database Navigator (highlighted with a red box) and select "Compile" from the context menu (also highlighted with a red box). A green circle labeled "1" indicates this step.
- Method 2 (Right):** Open the code editor for the procedure (highlighted with a red box) and click the "Compile" button in the toolbar (also highlighted with a red box). A green circle labeled "2" indicates this step.

OR

Procedimientos

- clic con el botón secundario
- Compilar

Edite el procedimiento

- Modificarlo
- Compilar

Compiler - Log x

```
Project: sqldev.temp:/ideConnections%23MyDBConnection.jpr
Procedure HELLO.PUT_LINE@MyDBConnec...
Error(2,1): PLS-00103: Encountered the symbol "BEGIN" when expecting one of the following:  (; is with authid as cluster compress order using com
```

Messages File Compiler < >

Corregir errores de compilación en SQL Developer

MyDBConnection x HELLO x

Code Grants Dependencies References Errors Details Profiles

Find

```
1 Switch to write mode
2 PROCEDURE HELLO
3 BEGIN
4 dbms_output.put_line('Hello Class!');
5 END HELLO;
```

1. Editar el procedimiento

MyDBConnection x HELLO x

Code Grants Dependencies References Errors Details Profiles

Find

```
1 create or replace
2 PROCEDURE HELLO IS
3 BEGIN
4 dbms_output.put_line('Hello Class!');
5 END HELLO;
```

2. Corregir error (agregar palabra IS)

Messages - Log x

Compiled

Procedures

- ADD_JOB_HISTORY
- HELLO
- SECURE_DML

4. Recompilación exitosa

MyDBConnection x HELLO x

Code Grants Dependencies References Errors Details Profiles

Find

```
1 create or replace
2 PROCEDURE HELLO IS
3 BEGIN
4 dbms_output.put_line('Hello Class!');
5 END HELLO;
```

Compile for Debug

Compile

3. Procedimiento de recompilación

¿Qué son los parámetros y los modos de parámetros?

- Los parámetros se utilizan para transferir valores de datos hacia y desde el entorno de llamada y el procedimiento.
- Los parámetros se declaran en el encabezado del subprograma, después del nombre y antes de la sección de declaración para las variables locales.
- Los parámetros están sujetos a uno de los tres modos de paso de parámetros: **IN**, **OUT**, **IN OUT**.

Parámetros formales y reales

- Parámetros **reales** (o argumentos):
 - Valores literales, variables y expresiones utilizados por el programa principal pasados al subprograma
- Parámetros **formales**:
 - Parámetros definidos en la especificación del subprograma

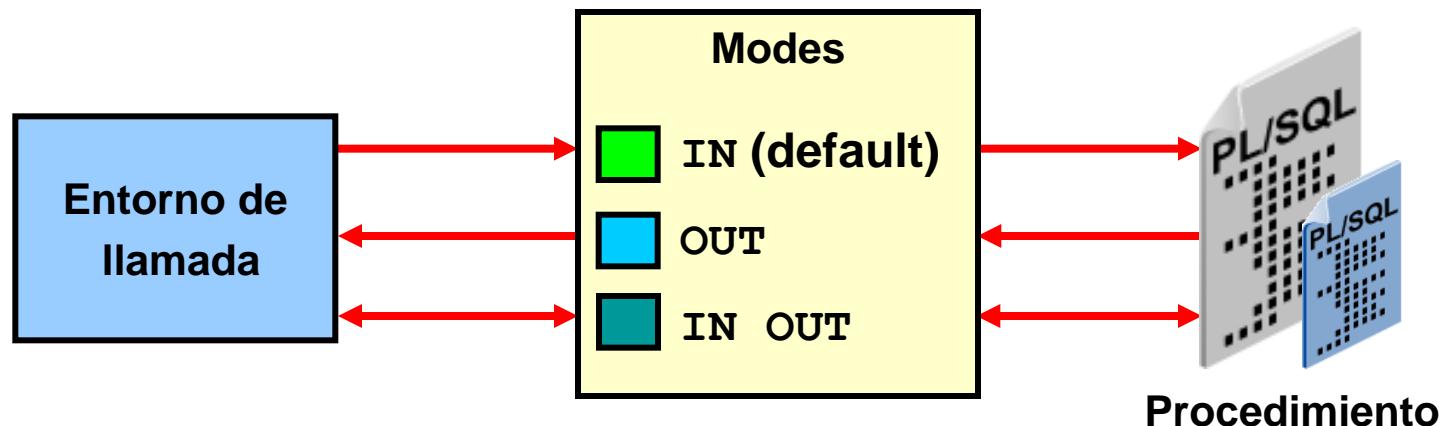
```
-- Procedure definition, Formal parameters
CREATE PROCEDURE raise_sal(p_id NUMBER, p_sal NUMBER) IS
BEGIN
    . . .
END raise_sal;
```

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

Modos de parámetros de procedimiento

- Los modos de parámetro se especifican en la declaración formal de parámetros, después del nombre del parámetro y antes de su tipo de datos.
- El modo `IN` es el predeterminado si no se especifica ningún modo.

```
CREATE PROCEDURE proc_name(param_name [mode] datatype)  
...
```

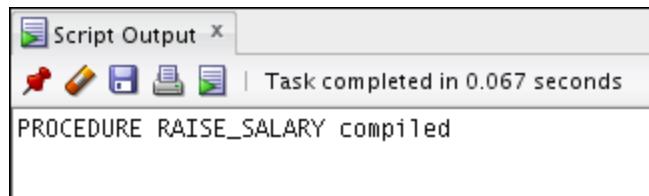


Comparación de los modos de parámetros

IN	OUT	IN OUT
Modo por defecto	Debe especificarse	Debe especificarse
El valor se pasa al subprograma	El valor se devuelve al entorno de llamada	Valor pasado en el subprograma; Valor devuelto al entorno de llamada
El parámetro formal actúa como una constante	Variable no inicializada	Variable no inicializada
El parámetro real puede ser una variable literal, de expresión o constante	Debe ser una variable	Debe ser una variable
Se puede asignar un valor predeterminado	No se puede asignar un valor predeterminado	No se puede asignar un valor predeterminado

Uso del modo de parámetro IN: Ejemplo

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



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

Uso del modo de parámetro OUT: Ejemplo

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

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

Uso del modo de parámetro IN OUT: Ejemplo

Calling environment

p_phone_no (before the call)

'8006330575'

p_phone_no (after the call)

'(800) 633-0575'

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

anonymous block completed
B_PHONE_NO

8006330575

anonymous block completed
B_PHONE_NO

(800) 633-0575

Visualización de los parámetros OUT: Uso de DBMS_OUTPUT.PUT_LINE

Utilice el procedimiento DBMS_OUTPUT.PUT_LINE para visualizar las variables PL/SQL utilizadas

```
SET SERVEROUTPUT ON

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

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

Visualización de parámetros OUT: Uso de variables de host de SQL*Plus

- Las variables SQL*Plus host son creadas mediante el comando **VARIABLE**.
 - Las variables SQL*Plus host son externas al bloque PL/SQL y se conocen como variables de host o bind.
 - Para hacer referencia a las variables de un bloque PL/SQL, debe colocar sus nombres con dos puntos (:).
 - Para mostrar los valores almacenados en las variables de host, debe utilizar el comando SQL*Plus **PRINT** seguido del nombre de la variable

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

anonymous block completed
B_NAME

Smith
B_SAL

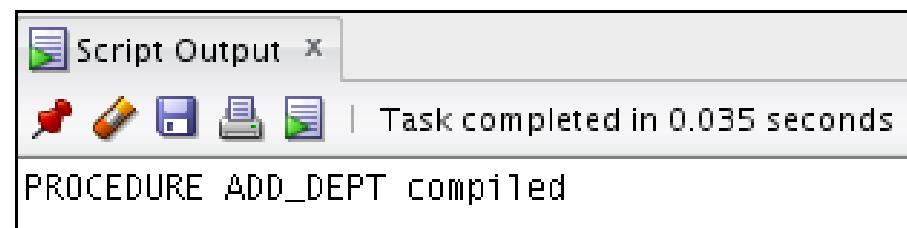
7400

Notaciones disponibles para pasar parámetros reales

- Al llamar a un subprograma, puede escribir los parámetros reales usando las siguientes anotaciones:
 - **Posicional:** Enumera los parámetros reales en el mismo orden que los parámetros formales
 - **Nombrado:** Enumera los parámetros reales en un orden arbitrario y utiliza el operador de asociación (**=>**) para asociar un parámetro formal con su parámetro real
 - **Mixto:** Enumera algunos de los parámetros reales como posicionales y otros como nombrados
- Antes de Oracle Database 11g, sólo se admite la notación de posición en las llamadas desde SQL.
- A partir de Oracle Database 11g, la notación nombrada y mixta se puede utilizar para especificar argumentos en las llamadas a las subrutinas PL/SQL de sentencias SQL.

Pasar parámetros reales: Crear el procedimiento add_dept

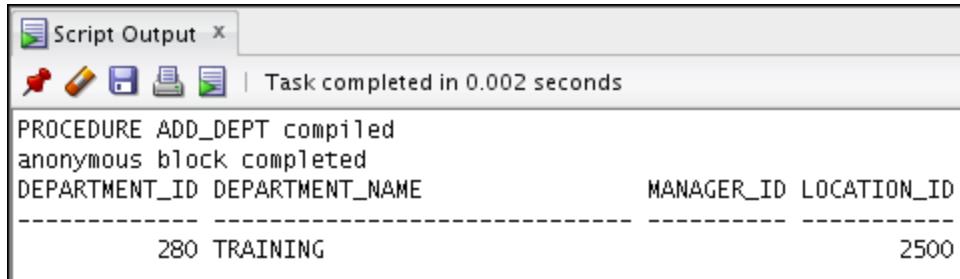
```
CREATE OR REPLACE PROCEDURE add_dept(
    p_name IN departments.department_name%TYPE,
    p_loc  IN departments.location_id%TYPE) IS
BEGIN
    INSERT INTO departments(department_id,
                           department_name, location_id)
    VALUES (departments_seq.NEXTVAL, p_name , p_loc );
END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



El procedimiento **add_dept** declara dos parámetros formales **IN**: **p_name** y **p_loc**. Los valores de estos parámetros se utilizan en la sentencia **INSERT** para establecer las columnas **department_name** y **location_id**, respectivamente.

Pasando parámetros reales: Ejemplos

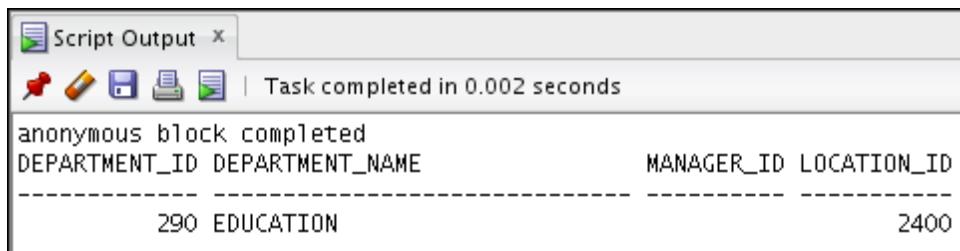
```
-- Pasar parámetros usando la notación posicional.  
EXECUTE add_dept ('TRAINING', 2500)
```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the results of executing the 'add_dept' procedure. The output includes a message about the procedure being compiled, the completion of an anonymous block, and the resulting department data. The data is presented in a table format:

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	TRAINING		2500

```
-- Pasar parámetros usando la notación nombrada.  
EXECUTE add_dept (p_loc=>2400, p_name=>'EDUCATION')
```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the results of executing the 'add_dept' procedure using named parameters. The output includes a message about the completion of an anonymous block and the resulting department data. The data is presented in a table format:

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
290	EDUCATION		2400

Uso de la opción DEFAULT para los parámetros

- Puede asignar un valor predeterminado a un parámetro IN de la siguiente manera:
 - El operador de asignación (`:=`),
 - La opción **DEFAULT**

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

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

Llamando a los Procedimientos

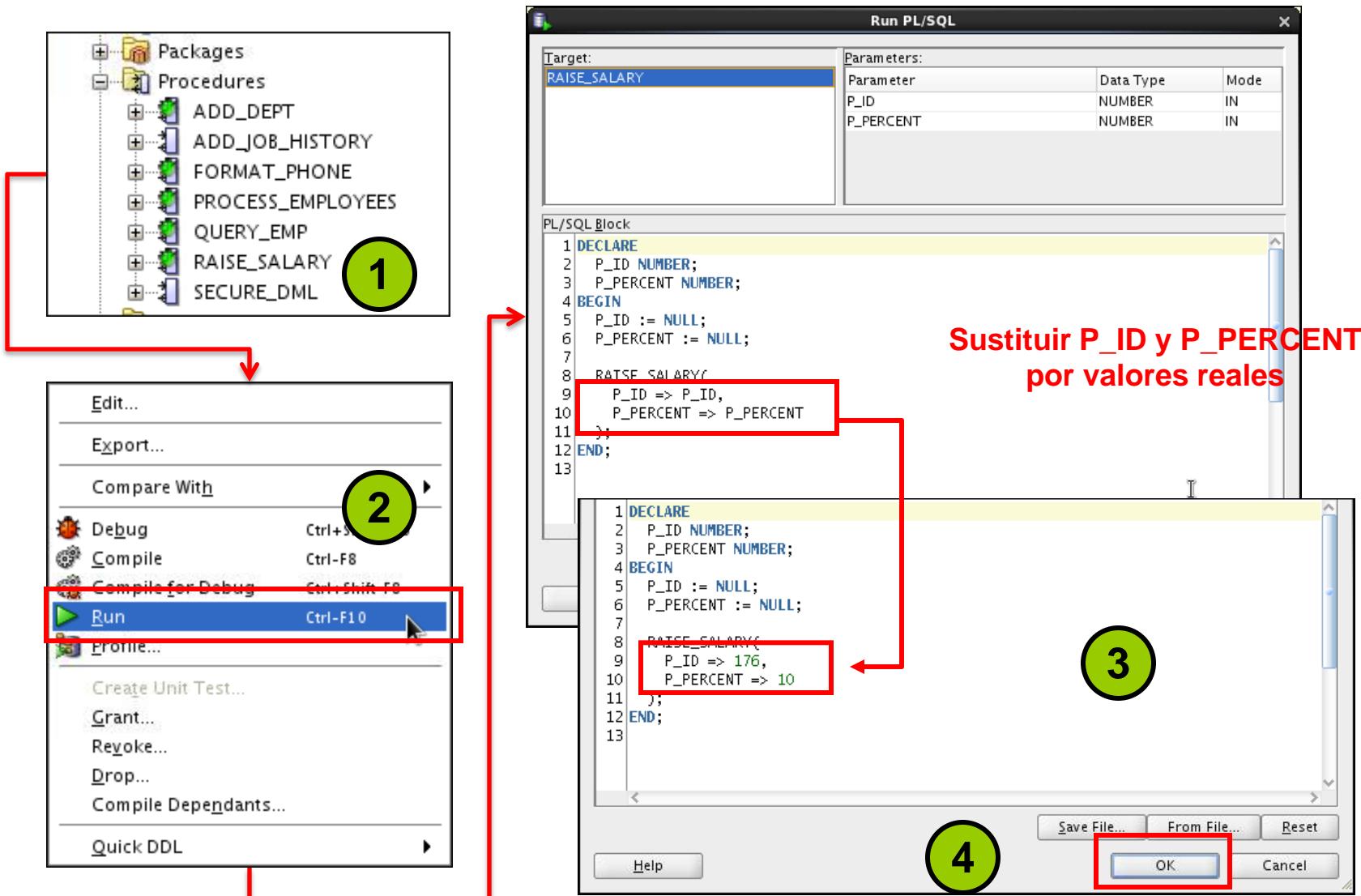
- Puede invocar procedimientos de dos formas diferentes:
 - Bloques anónimos
 - Otro procedimiento o subprograma PL/SQL.
- Debe de ser dueño del procedimiento o tener privilegio **EXECUTE**.

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

PROCEDURE PROCESS_EMPLOYEES compiled

- El ejemplo de la diapositiva le muestra cómo invocar un procedimiento desde otro procedimiento almacenado

Llamando a procedimientos con SQL Developer



Agenda

- Utilizando un diseño de subprograma modularizado y en capas e identificando los beneficios de los subprogramas
- Trabajar con procedimientos:
 - Creación y llamada de procedimientos
 - Identificación de los modos de paso de parámetros disponibles
 - Usando parámetros formales y reales
 - Uso de la notación posicional, nombrada o mixta
- Manejo de excepciones en los procedimientos, eliminación de un procedimiento y visualización de la información del procedimiento

Gestión de Excepciones

- Cuando desarrolla procedimientos que se llaman de otros procedimientos, debe tener en cuenta los efectos que las excepciones.
- Cuando se genera una excepción en un procedimiento llamado, el control pasa inmediatamente a la sección de excepciones de ese bloque.
 - Si la excepción es tratada, se produce el siguiente flujo de código:
 1. Se plantea la excepción.
 2. El control se transfiere al manejador de excepciones.
 3. El bloque se termina.
 4. El programa/bloque que ha llamado continúa ejecutándose como si nada hubiera sucedido.
- Si se inició una transacción, antes de ejecutar el procedimiento en el que se generó la excepción, **la transacción no se verá afectada.**

Gestión de Excepciones

Procedimiento de llamada

```
PROCEDURE  
  PROC1 ...  
IS  
  ...  
BEGIN  
  ...  
  PROC2 (arg1) ;  
  ...  
EXCEPTION  
  ...  
END PROC1;
```

Procedimiento llamado

```
PROCEDURE  
  PROC2 ...  
IS  
  ...  
BEGIN  
  ...  
EXCEPTION  
  ...  
END PROC2;
```

Excepción producida

Excepción tratada

El control vuelve al
procedimiento de llamada

Gestión de Excepciones: Ejemplo

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

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

Excepciones no tratadas

- Si cuando se produce una excepción y ésta no es controlada en el código llamante, se produce el siguiente flujo de código:
 1. Se plantea la excepción.
 2. El bloque termina porque no existe un manejador de excepciones; Cualquier operación **DML realizada dentro del procedimiento se deshace.**
 3. La excepción se propaga a la sección de excepción del procedimiento de llamada, es decir, el control se devuelve a la sección de excepción del bloque de llamada, si existe.
 4. Si no se controla una excepción, todas las sentencias DML en el procedimiento de llamada y el procedimiento llamado se deshacen.
 5. Las sentencias DML que no se ven afectadas son declaraciones que se ejecutaron antes de llamar al código PL / SQL cuyas excepciones no se tratan.

Excepciones no tratadas

Procedimiento de llamada

```
PROCEDURE  
  PROC1 ...  
  IS  
  ...  
 BEGIN  
  ...  
  PROC2 (arg1) ;  
  ...  
 EXCEPTION  
  ...  
 END PROC1 ;
```

Procedimiento llamado

```
PROCEDURE  
  PROC2 ...  
  IS  
  ...  
 BEGIN  
  ...  
 EXCEPTION  
  ...  
 END PROC2 ;
```

Excepción producida

Excepción NO tratada

Control devuelto a la
sección de excepción
del procedimiento de
llamada

Excepciones no tratadas: ejemplo

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

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

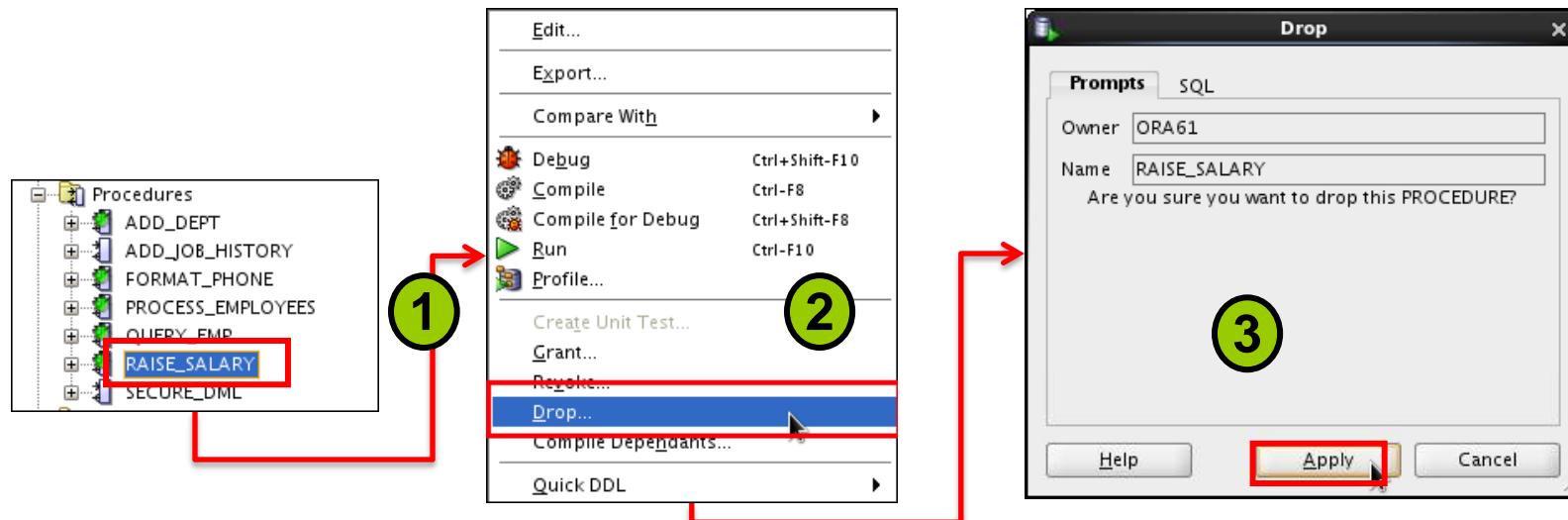
add_department_noex, no tiene
una sección de excepción

Eliminación de Procedimientos: Uso de DROP PROCEDURE o SQL Developer

- Cuando ya no es necesario un procedimiento almacenado, puede utilizar la instrucción **DROP PROCEDURE** seguida por el nombre del procedimiento:

```
DROP PROCEDURE raise_salary;
```

- También puede utilizar SQL Developer para eliminarlo



Información de procedimientos mediante las vistas de diccionario de datos

```
DESCRIBE user_source
```

```
DESCRIBE user_source
Name Null Type
-----
NAME      VARCHAR2(128)
TYPE      VARCHAR2(12)
LINE      NUMBER
TEXT      VARCHAR2(4000)
```

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

TEXT
1 PROCEDURE add_dept(2 p_name departments.department_name%TYPE:='Unknown', 3 p_loc departments.location_id%TYPE DEFAULT 1700) IS 4 5 BEGIN 6 INSERT INTO departments (department_id, department_name, location_id) 7 VALUES (departments_seq.NEXTVAL, p_name, p_loc); 8 END add_dept;

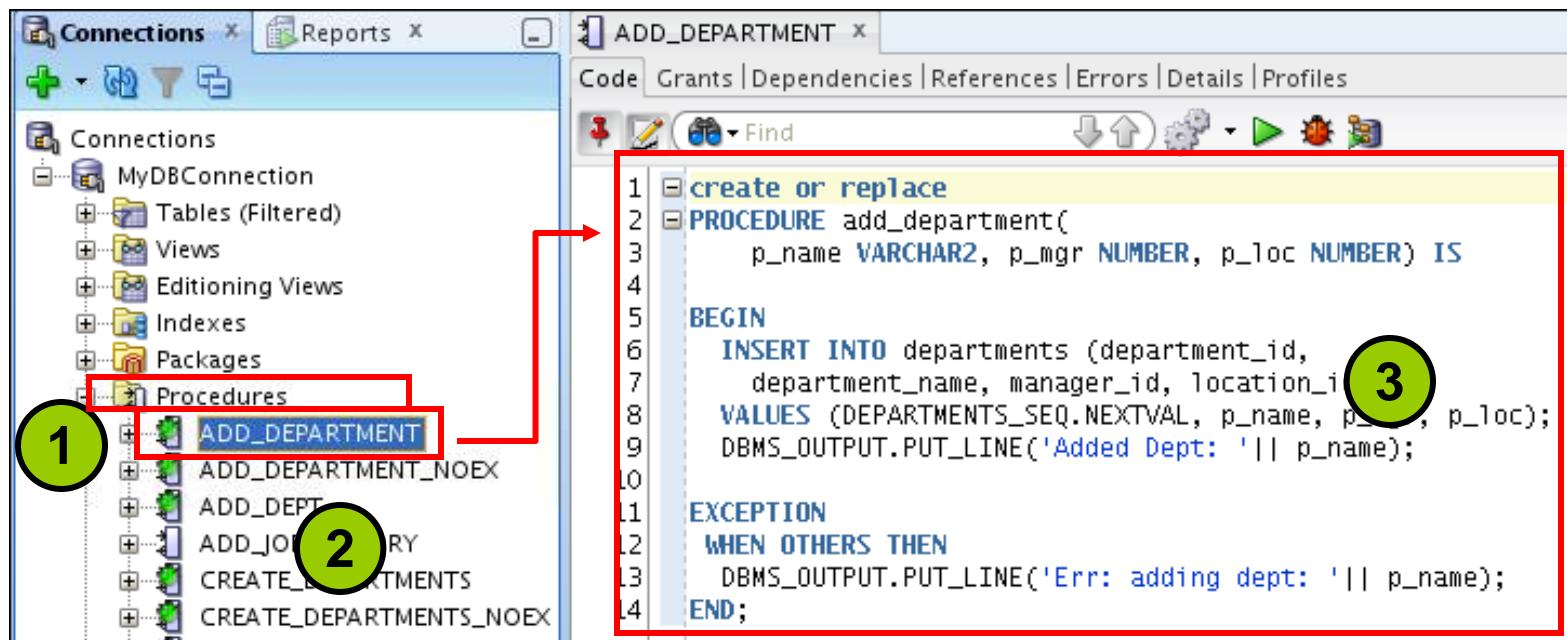
USER_SOURCE

- para mostrar el código PL/SQL que posee

ALL_SOURCE

- para mostrar el código PL/SQL al que se le ha concedido el derecho EXECUTE por el propietario de ese código de subprograma

Visualización de información de procedimientos con SQL Developer



1. Haga clic en el nodo Procedimientos en la pestaña Conexiones.
2. Haga clic en el nombre del procedimiento.
3. El código del procedimiento se muestra en la ficha Código como se muestra en la diapositiva.

Quiz

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

- a. True
- b. False

Resumen

En esta lección, debes haber aprendido a:

- Identificar los beneficios del diseño de subprogramas modularizados y en capas
- Crear y llamar procedimientos
- Utilizar parámetros formales y reales
- Usar notación posicional, nombrada o mixta para pasar parámetros
- Identificar los modos de paso de parámetros disponibles
- Manejar excepciones en los procedimientos
- Eliminar un procedimiento y mostrar su información

Práctica 2: Crear, compilar y llamar procedimientos

En esta lección, realiza las siguientes prácticas:

- Creación de procedimientos almacenados para:
 - Insertar nuevas filas en una tabla utilizando los valores de parámetro suministrados
 - Actualizar datos en una tabla para filas que coincidan con los valores de los parámetros suministrados
 - Eliminar filas de una tabla que coincide con los valores de los parámetros suministrados
 - Consultar una tabla y recuperar datos basados en los valores de los parámetros suministrados
- Manejo de excepciones en los procedimientos
- Procedimientos de compilación e invocación

3

Creación de funciones y depuración de subprogramas



ORACLE®

Objetivos

Después de completar esta lección, usted debería ser capaz de:

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

Agenda

- Trabajo con funciones:
 - Diferenciación entre un procedimiento y una función
 - Describir los usos de las funciones
 - Creación, invocación y eliminación de funciones almacenadas
- Introducción al SQL Developer debugger

Descripción general de las funciones almacenadas

- Una función es un bloque denominado PL/SQL que puede aceptar parámetros, ser invocado y devolver un valor
 - En general, se utiliza una función para calcular un valor.
- Las funciones y los procedimientos se estructuran igualmente.
- Una función **debe devolver un valor** al entorno de llamada, mientras que un procedimiento devuelve cero o más valores a su entorno de llamada.
- Al igual que un procedimiento, una función tiene:
 - Un encabezado
 - Una sección declarativa
 - Una sección ejecutable y
 - Una sección opcional de manejo de excepciones.

Descripción general de las funciones almacenadas

- Una función debe tener una cláusula **RETURN** en el encabezado y al menos una instrucción **RETURN** en la sección ejecutable.
- Las funciones se pueden almacenar en la base de datos como **objetos de esquema** para la ejecución repetida.
 - Una función que se almacena en la base de datos se conoce como una función almacenada.
- Cuando una función es validada, se pueden reutilizar con cualquier número de aplicaciones.
- Una función se puede llamar como parte de:
 - Una expresión SQL
 - Una expresión PL/SQL

Creación de funciones

- Puede crear nuevas funciones con la instrucción **CREATE OR REPLACE FUNCTION**

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

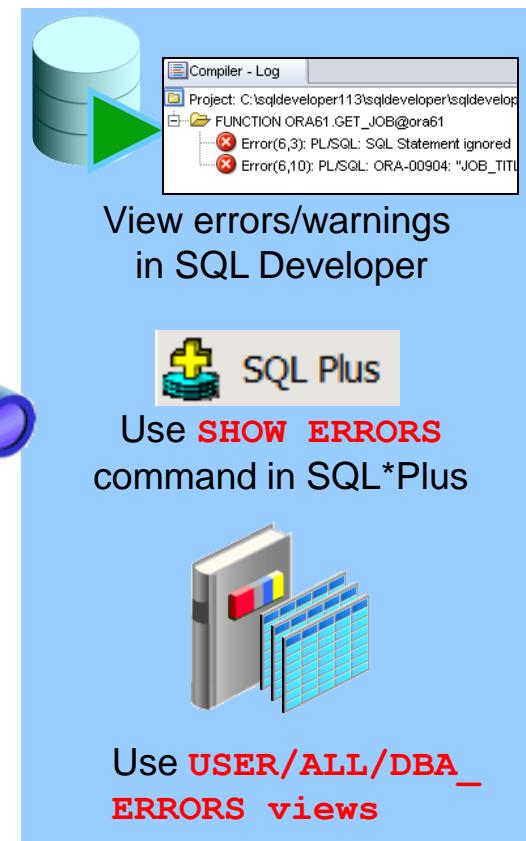
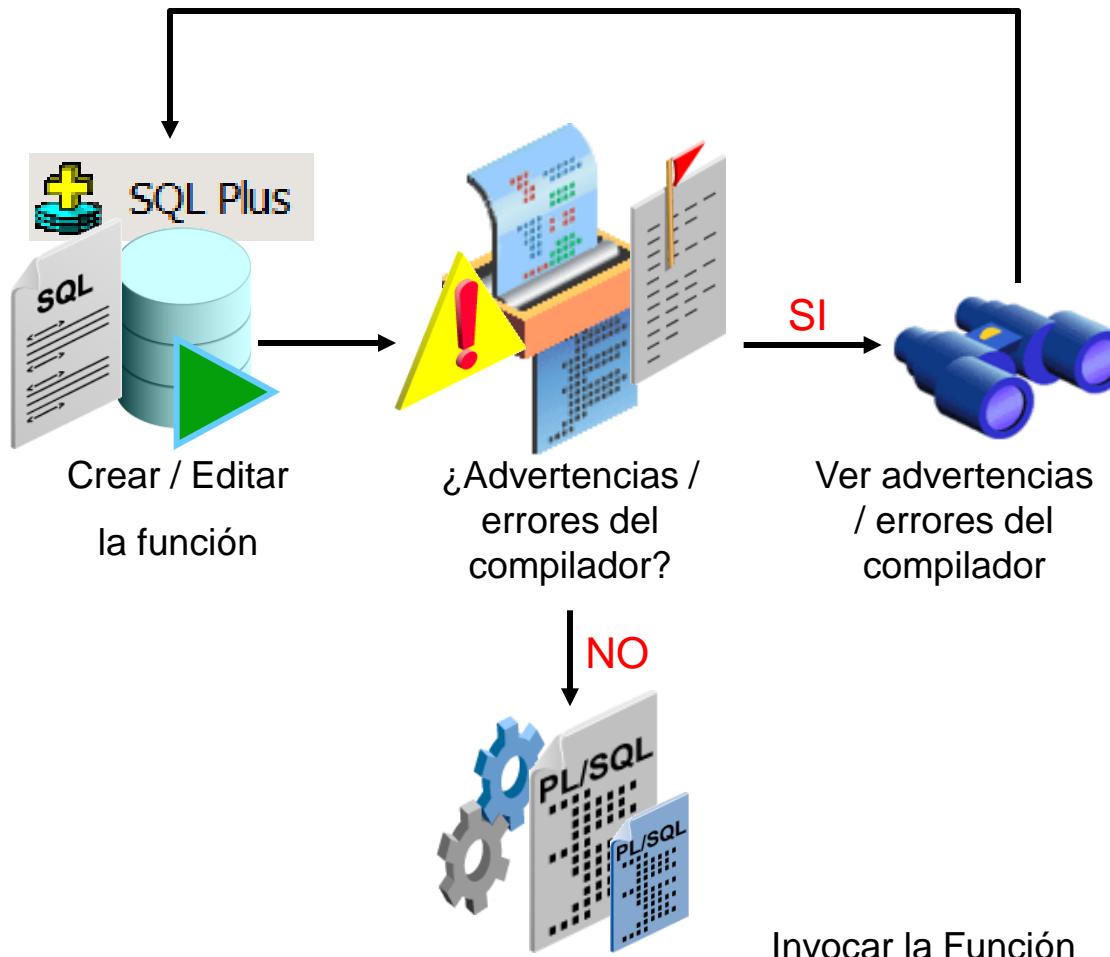
PL/SQL Block

- Podemos indicar una lista de parámetros
- Debe devolver un valor y definir las acciones que debe realizar el bloque PL/SQL estándar.

La diferencia entre procedimientos y funciones

Prodecimientos	Funciones
Se ejecuta como una instrucción PL/SQL	Se invoca como parte de una expresión
No contiene la cláusula RETURN en el encabezado	Debe contener una cláusula RETURN en el encabezado
Puede pasar valores (si los hay) usando parámetros de salida	Debe devolver un solo valor
Puede contener una sentencia RETURN sin valor	Debe contener al menos una declaración RETURN

Creación y ejecución de funciones: descripción general



Creación e invocación de una función mediante la instrucción CREATE FUNCTION: Ejemplo

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

La función **get_sal** se crea con un único parámetro de entrada y devuelve el salario como un número

FUNCTION GET_SAL compiled

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

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

anonymous block completed
24000

Uso de diferentes métodos para ejecutar funciones

```
-- Como una expresión de PL/SQL, obtenga los resultados  
-- usando variables de host
```

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

```
anonymous block completed  
B_SALARY  
-----  
24000
```

```
-- Como expresión PL / SQL, obtenga los resultados utilizando  
-- una variable local
```

```
SET SERVEROUTPUT ON  
DECLARE  
    sal employees.salary%type;  
BEGIN  
    sal := get_sal(100);  
    DBMS_OUTPUT.PUT_LINE('The salary is: ' || sal);  
END;  
/
```

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

Uso de diferentes métodos para ejecutar funciones

```
-- Utilizar como parámetro a otro subprograma
```

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

```
anonymous block completed  
24000
```

```
-- Uso en una sentencia SQL (sujeto a restricciones)
```

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

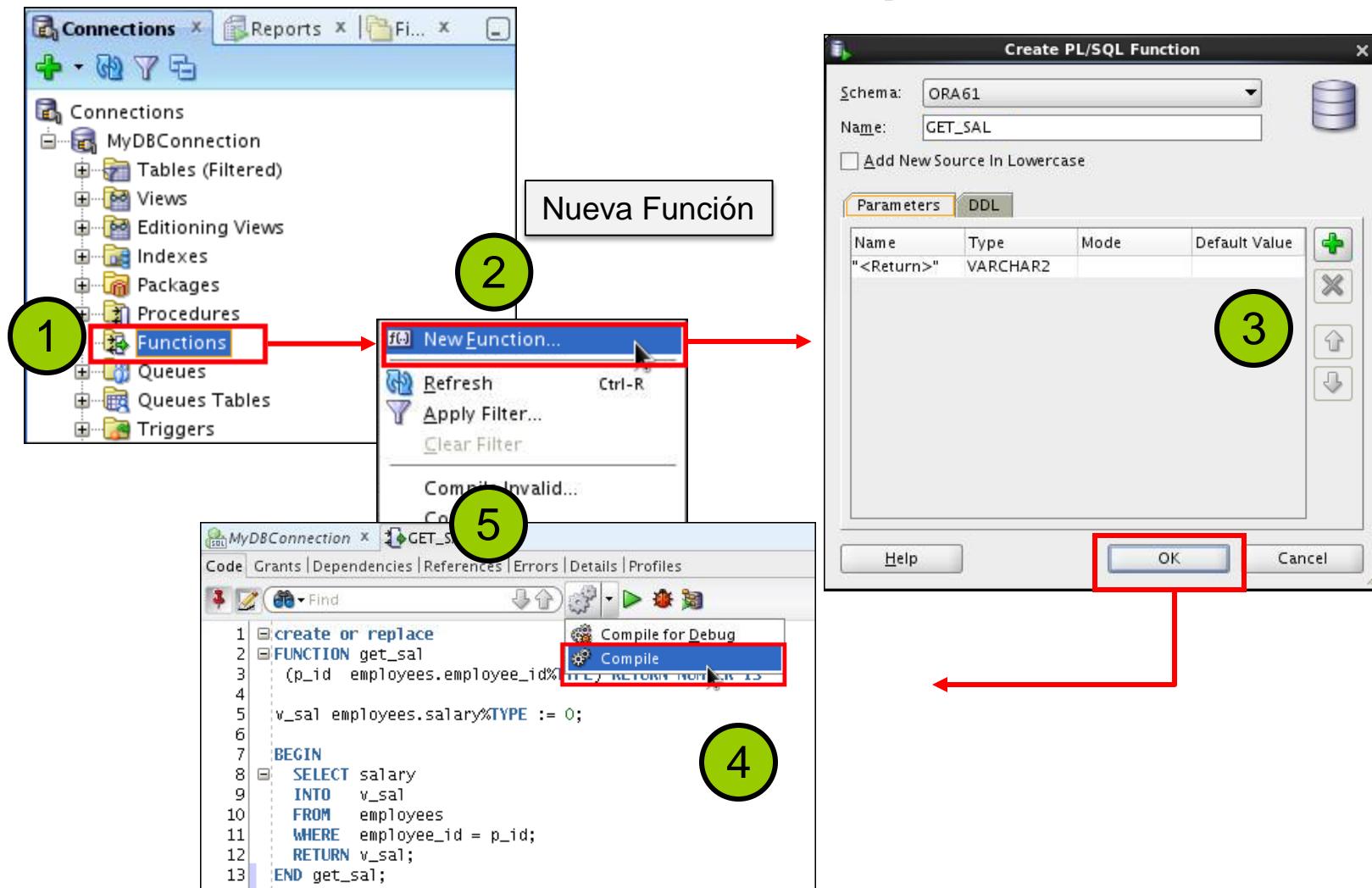
JOB_ID	GET_SAL(EMPLOYEE_ID)
AC_ACCOUNT	8300
AC_MGR	12008
AD_ASST	4400
AD_PRES	24000

...

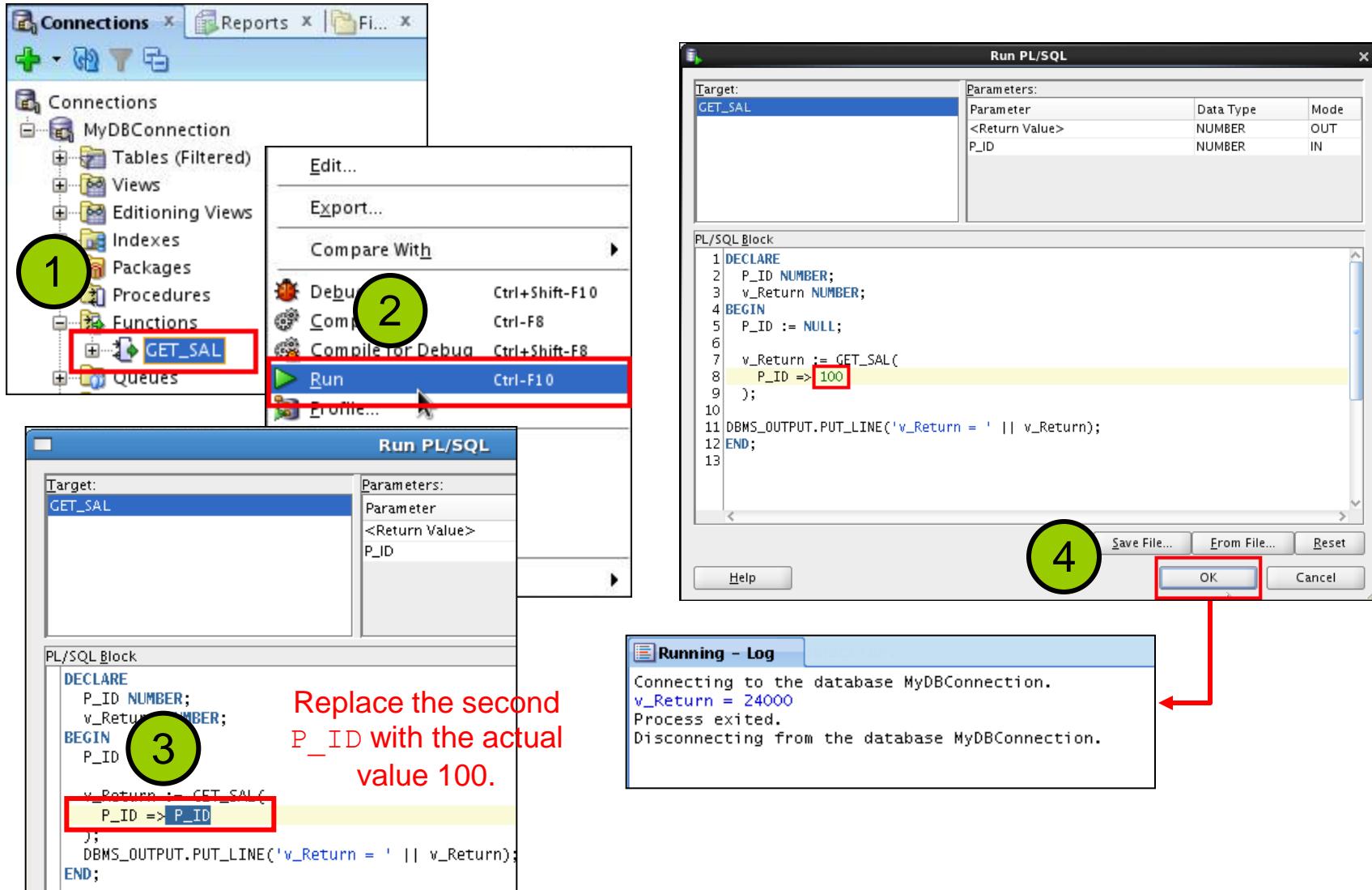
ST_MAN	6500
ST_MAN	5800

```
107 rows selected
```

Creación y compilación de funciones con SQL Developer



Ejecutar funciones con SQL Developer



Ventajas de las funciones definidas por el usuario en las instrucciones SQL

- Las sentencias de SQL pueden hacer referencia a las funciones definidas por el usuario de PL/SQL.
 - Por ejemplo, una función definida por el usuario puede utilizarse en cualquier lugar en el que pueda colocarse una función SQL incorporada, como UPPER ().

Ventajas

- Permite **cálculos complejos**, incómodos o no disponibles con SQL.
 - Las funciones aumentan la independencia de los datos mediante el procesamiento de análisis de datos complejos dentro del servidor Oracle, en lugar de recuperar los datos en una aplicación
- **Aumenta la eficiencia** de las consultas al realizar funciones en la consulta en lugar de en la aplicación
- **Manipula nuevos tipos de datos** y nos da la posibilidad de codificar cadenas de caracteres

Uso de una función en una expresión SQL: Ejemplo

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

FUNCTION TAX compiled	EMPLOYEE_ID	LAST_NAME	SALARY	TAX(SALARY)
	108	Greenberg	12008	960.64
	109	Faviet	9000	720
	110	Chen	8200	656
	111	Sciarra	7700	616
	112	Urman	7800	624
	113	Popp	6900	552
6 rows selected				

Donde podemos llamar a funciones definidas por el usuario en sentencias SQL

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

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

Restricciones al llamar a funciones de SQL Expressions

- Las funciones definidas por el usuario que se pueden llamar desde expresiones SQL deben:
 - Almacenarse en la base de datos
 - Aceptar sólo parámetros `IN` con tipos de datos SQL válidos y tipos de datos específicos de PL/SQL
 - Devolver tipos de datos SQL válidos y tipos de datos específicos de PL/SQL
- Al llamar a funciones en sentencias de SQL:
 - Debe ser dueños de la función o tener el privilegio `EXECUTE`
 - Si se quiere que permita una ejecución paralela, debe de indicarlo mediante la palabra clave `PARALLEL_ENABLE`

Restricciones en las funciones de llamada desde SQL: Ejemplo

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

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

```
FUNCTION DML_CALL_SQL compiled
Error starting at line 127 in command:
UPDATE employees
  SET salary = dml_call_sql(2000)
 WHERE employee_id = 170
Error report:
SQL Error: ORA-04091: table ORA61.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA61.DML_CALL_SQL", line 4
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
*Cause:  A trigger (or a user defined plsql function that is referenced in
        this statement) attempted to look at (or modify) a table that was
        in the middle of being modified by the statement which fired it.
*Action: Rewrite the trigger (or function) so it does not read that table.
```

La instrucción **UPDATE** falla y muestra un error que indica que la tabla está cambiando (La función realiza un **INSERT** en la misma tabla)

Notación Nombrada y Mixta de SQL

- PL/SQL permite que los argumentos de una llamada de subrutina se especifiquen utilizando **notación posicional, nombrada o mixta**.
 - Antes de Oracle Database 11g, sólo se admite la **notación de posición** en las llamadas desde SQL.
 - A partir de Oracle Database 11g, la notación **nombrada y mixta** se puede utilizar para especificar argumentos en las llamadas a las subrutinas PL/SQL de sentencias SQL.
- Para listas de parámetros largos, con la mayoría de los valores por defecto, puede omitir valores de los parámetros opcionales.
- Puede evitar duplicar el valor predeterminado del parámetro opcional en cada sitio de llamada.

Notación Nombrada y Mixta desde SQL: Ejemplo

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

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

```
FUNCTION F compiled
F(P_PARAMETER_5=>10)
-----
```

21

Llamada a la función **f** dentro
de la sentencia SQL
SELECT utiliza la **notación
nombrada**

Visualización de funciones mediante vistas de diccionarios de datos

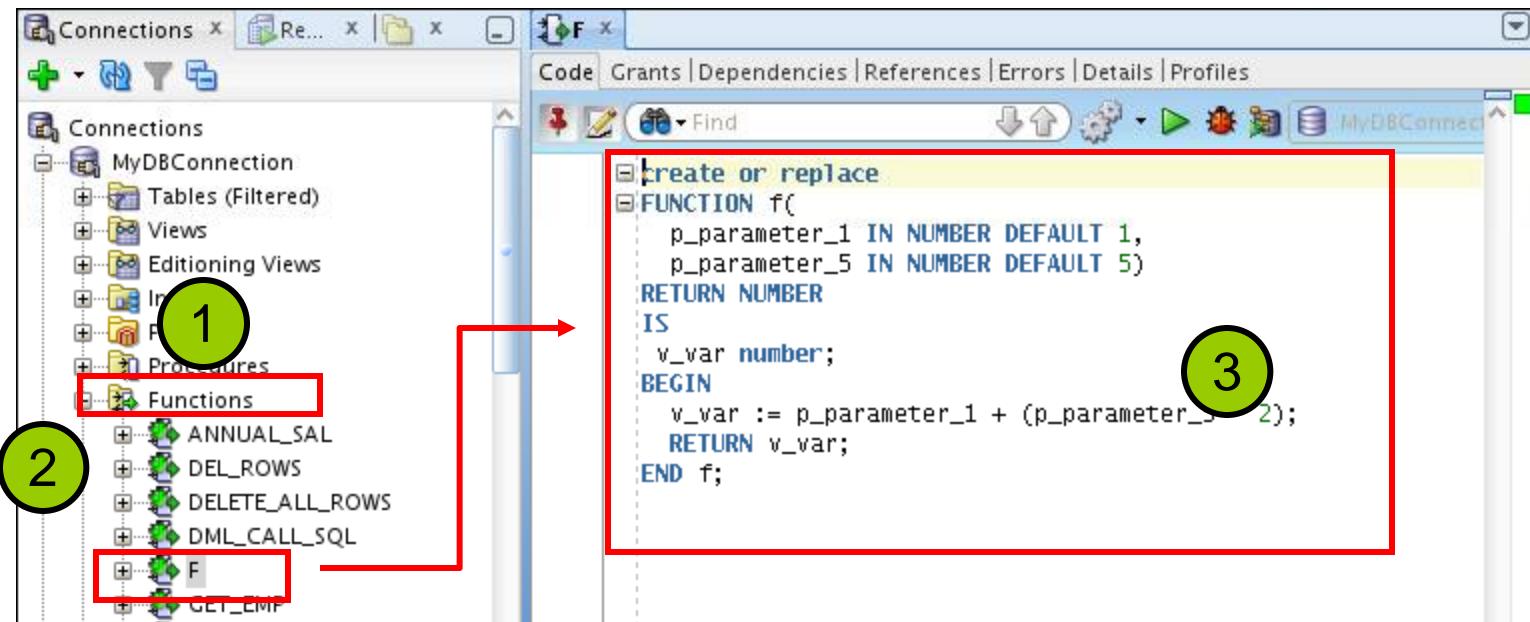
```
DESCRIBE USER_SOURCE
```

```
DESCRIBE user_source
Name Null Type
-----
NAME      VARCHAR2(128)
TYPE      VARCHAR2(12)
LINE      NUMBER
TEXT      VARCHAR2(4000)
```

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

TEXT
1 FUNCTION dml_call_sq1(p_sa1 NUMBER)
2 FUNCTION tax(p_value IN NUMBER)
3 FUNCTION query_call_sq1(p_a NUMBER) RETURN NUMBER IS
4 FUNCTION get_sal
5 RETURN NUMBER IS
6 RETURN NUMBER IS
7 (p_id employees.employee_id%TYPE) RETURN NUMBER IS
8 v_s NUMBER;

Visualización de funciones de información mediante SQL Developer



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

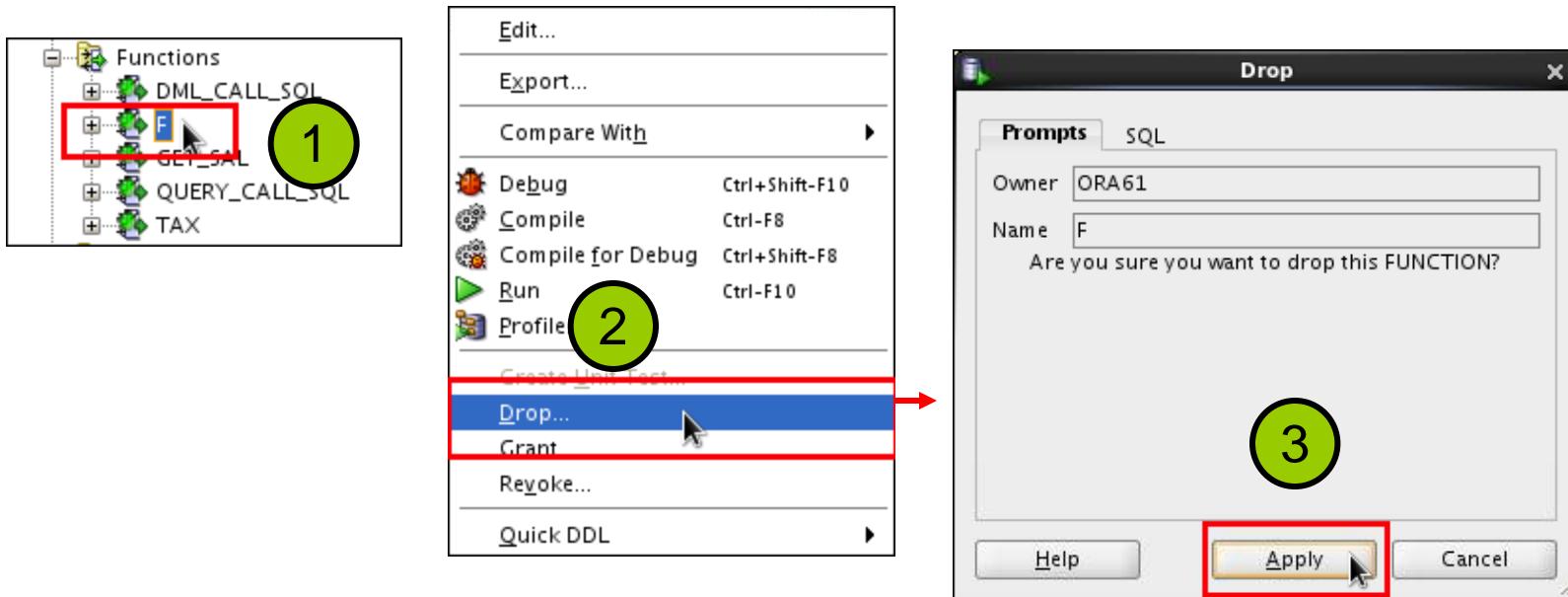
1. Haga clic en el nodo **Funciones** en la pestaña Conexiones.
2. Haga clic en el **nombre de la función**.
3. El código de la función se muestra en la ficha **Código** como se muestra en la diapositiva.

Eliminación de funciones: mediante la instrucción DROP o SQL Developer

- Uso de la instrucción **DROP**:

```
DROP FUNCTION f;
```

- Usando SQL Developer:



Quiz

Una función almacenada PL/SQL: (4)

- a. Se puede invocar como parte de una expresión
- b. Debe contener una cláusula RETURN en el encabezado
- c. Debe devolver un solo valor
- d. Debe contener al menos una declaración RETURN
- e. No contiene una cláusula RETURN en el encabezado

Práctica 3-1

En esta lección, realiza las siguientes prácticas:

- Creación de funciones almacenadas:
 - Para consultar una tabla de base de datos y devolver valores específicos
 - Para ser utilizado en una sentencia SQL
 - Para insertar una nueva fila, con valores de parámetros especificados, en una tabla de base de datos
 - Uso de valores de parámetros predeterminados
- Invocación de una función almacenada de una sentencia SQL
- Invocación de una función almacenada de un procedimiento almacenado

Agenda

- Trabajo con funciones:
 - Diferenciación entre un procedimiento y una función
 - Describir los usos de las funciones
 - Creación, invocación y eliminación de funciones almacenadas
- Introducción al SQL Developer debugger

Depuración de subprogramas PL / SQL con SQL Developer Debugger

- El **depurador de SQL Developer** le permite controlar como se realiza la ejecución de un programa.
 - Cosas como:
 - Controlar una línea determinada de código
 - Controlar si el subprograma se ejecuta completo (procedimiento o función)
 - Controlar si un bloque/sub-bloque es ejecutado
- Para depurar un subprograma PL/SQL, un administrador de seguridad debe conceder los siguientes privilegios al desarrollador de aplicaciones:
 - DEBUG ANY PROCEDURE
 - DEBUG CONNECT SESSION

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

Depuración de un subprograma: descripción general

```
MyDBConnection X ADD_JOB_HISTORY X
Code Grants Dependencies References Errors Data
Switch to write mode
CREATE OR REPLACE PROCEDURE add_job_history
(
    p_emp_id          job.job_id%TYPE,
    p_start_date      job.job_start_date%TYPE,
    p_end_date        job.job_end_date%TYPE,
    p_job_id          job.job_id%TYPE,
    p_department_id   job.department_id%TYPE
);
```

1. Editar el procedimiento

```
1  create or replace
2  PROCEDURE add_job_history
3  (
4      p_emp_id          job.job_id%TYPE,
5      p_start_date      job.job_start_date%TYPE,
6      p_end_date        job.job_end_date%TYPE,
7      p_job_id          job.job_id%TYPE,
8      p_department_id   job.department_id%TYPE
9  );
```

2. Agregar puntos de
interrupción

```
ADD_JOB_HISTORY X
Dependencies References Errors Details Profiles
Replace
Compile for Debug
Compile
```

3. Compilar para depuración

Debugging: IdeConnections%23MyDBConnection.jpr - Log X Breakpoints X

Connecting to the database MyDBConnection.

Executing PL/SQL: ALTER SESSION SET PLSQL_DEBUG=TRUE

Executing PL/SQL: CALL DBMS_DEBUG_JDWP.CONNECT_TCP('127.0.0.1', 5001);

Name	Value	Type
P_EMP_ID	100	NUMBER
P_START_DATE	NULL	DATE
P_END_DATE	NULL	DATE
P_JOB_ID	NULL	VARCHAR2
P_DEPARTMENT_ID	NULL	NUMBER

6. Seleccione **debugging tool**

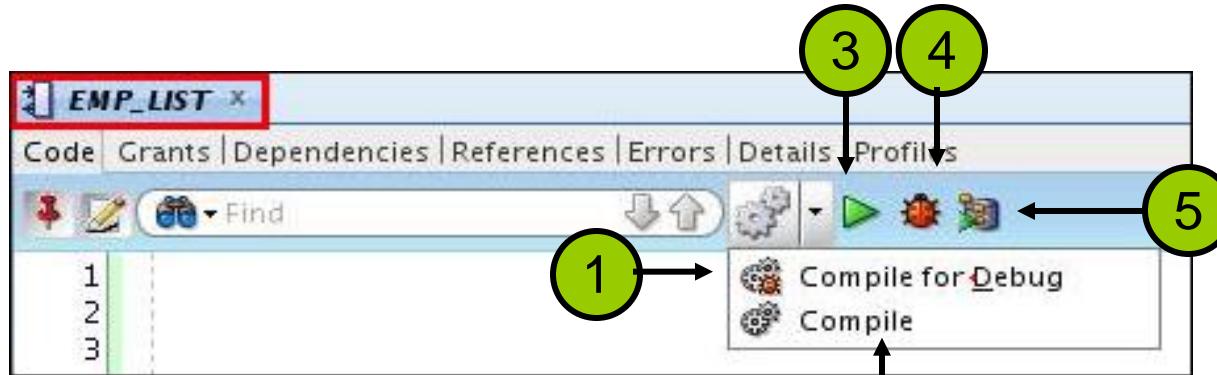
```
10  P_END_DATE := NULL;
11  P_JOB_ID := NULL;
12  P_DEPARTMENT_ID := NULL;
13
14  ADD_JOB_HISTORY(
15      P_EMP_ID = 100,
```



5. Introducir el(los) valor(es)
del parámetro

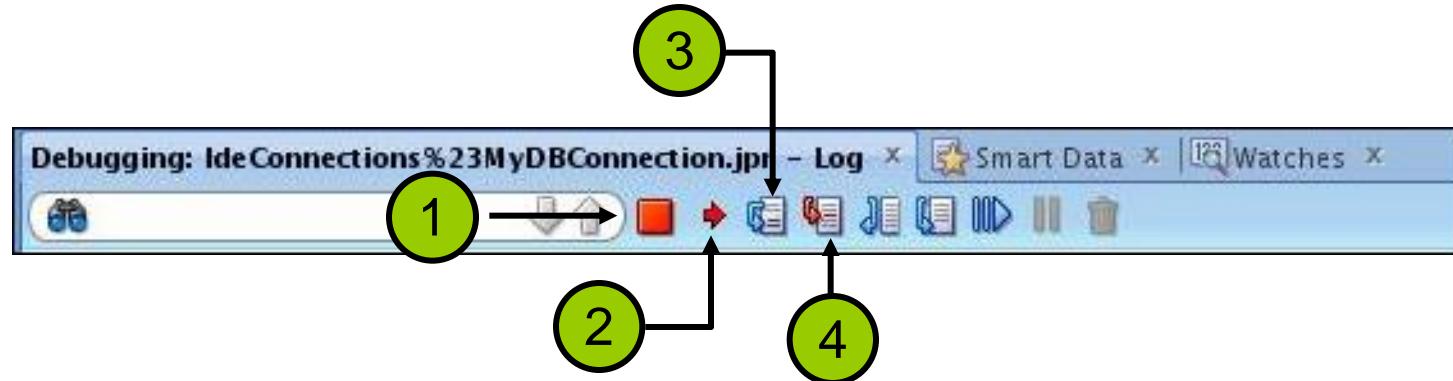
4. Debug

Barra de Herramientas



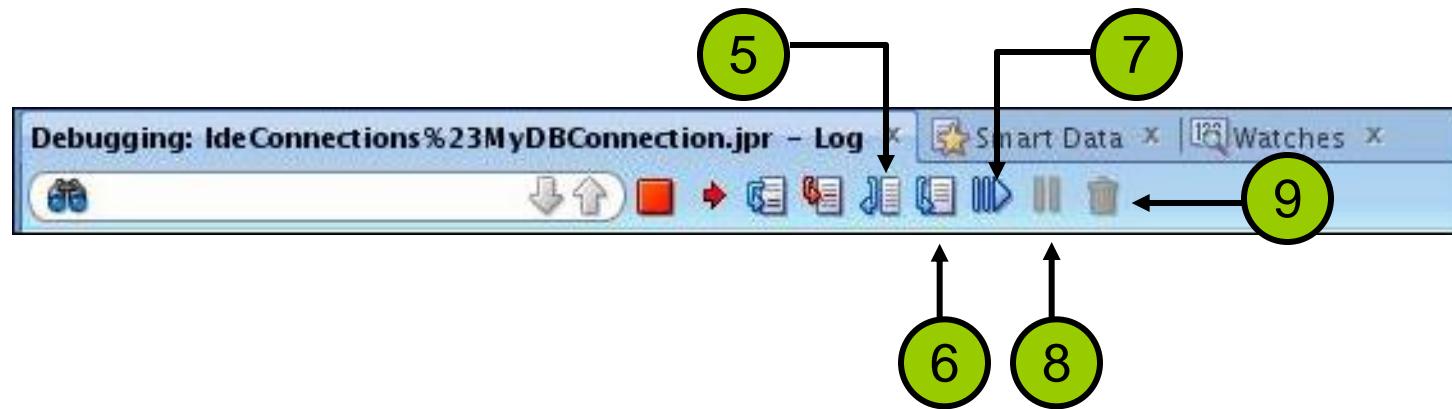
Icon	Descripción
1. Compilar depuración	Compila el subprograma para que se pueda depurar
2. Compilar	Compila el subprograma
3. Ejecutar	Inicia la ejecución normal de la función o procedimiento y muestra los resultados en la ficha Ejecución - Registro
4. Debug	Ejecuta el subprograma en modo de depuración y muestra la ficha Depuración - Registro, que incluye la barra de herramientas de depuración para controlar la ejecución
5. Perfil	Muestra la ventana Perfil que se utiliza para especificar valores de parámetros para ejecutar, depurar o crear perfiles de una función o procedimiento PL / SQL

The Debugging – Barra de registro



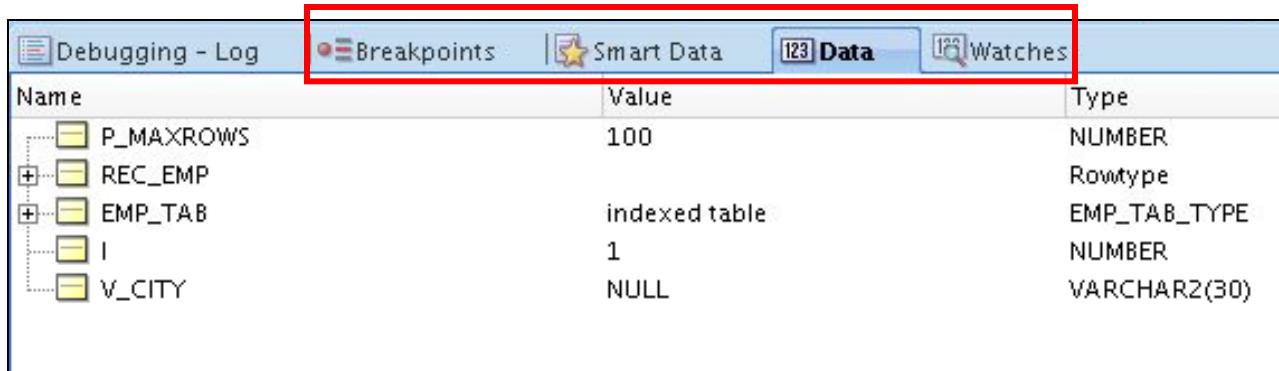
Icon	Descripción
1. Terminar	Detener y salir de la ejecución
2.- Busque el punto de ejecución	Ir al siguiente punto de ejecución
3. Paso más	Desvía el siguiente subprograma y pasa a la siguiente instrucción después del subprograma
4. Entrar	Ejecuta una sola instrucción de programa a la vez. Si el punto de ejecución se encuentra en una llamada a un subprograma, pasa a la primera instrucción en ese subprograma

The Debugging – Barra de registro



Icon	Descripción
5. Paso hacia fuera	Deja el subprograma actual y pasa a la siguiente instrucción con un punto de interrupción
6. Paso al final del método	Va a la última instrucción del subprograma actual
7. Resumen	Continúa la ejecución
8. Pausa	Detiene la ejecución pero no sale
9. Recolección de basura	Elimina objetos no válidos de la caché

Pestañas adicionales



Tab	Descripción
Breakpoints	Muestra puntos de interrupción, tanto definidos por el sistema como definidos por el usuario.
Smart Data	Muestra información sobre variables. Puede especificar estas preferencias haciendo clic con el botón derecho del ratón en la ventana Smart Data y seleccionando Preferencias.
Data	Ubicado debajo del área de texto de código; Muestra información sobre todas las variables
Watches	Muestra información sobre los watches Cambios de valores de las variables o expresiones

Depuración de un procedimiento

Ejemplo: Creación de un procedimiento emp_list

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

Depuración de un procedimiento

Ejemplo: Creación de una función get_location

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

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

Establecimiento Breakpoints y Compiling emp_list para el modo de depuración

The screenshot shows the Oracle SQL Developer interface with the 'EMP_LIST' procedure open in the editor. Several breakpoints are marked with red circles and red boxes around the corresponding lines of code. The 'Compile for Debug' button is highlighted in the toolbar. The 'Messages - Log' tab at the bottom shows the message 'Compiled'.

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

- Se añaden cuatro puntos de interrupción a varias ubicaciones del código.

Compilación de la función get_location en el modo de depuración

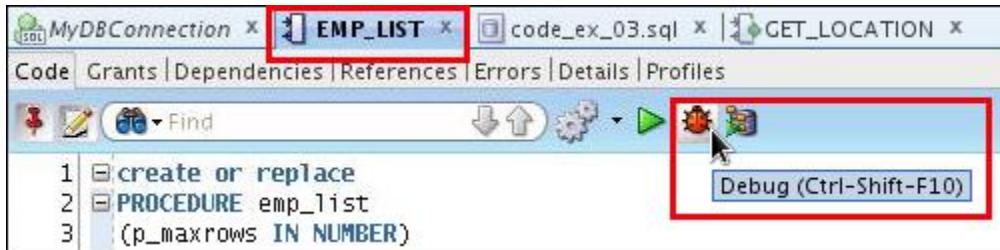
The screenshot shows the Oracle SQL Developer interface. The top menu bar has tabs for 'MyDBConnection', 'EMP_LIST', 'code_ex_03.sql', and 'GET_LOCATION'. The 'GET_LOCATION' tab is active and highlighted with a red box. Below the tabs is a toolbar with various icons. The main area displays a PL/SQL code editor with the following code:

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

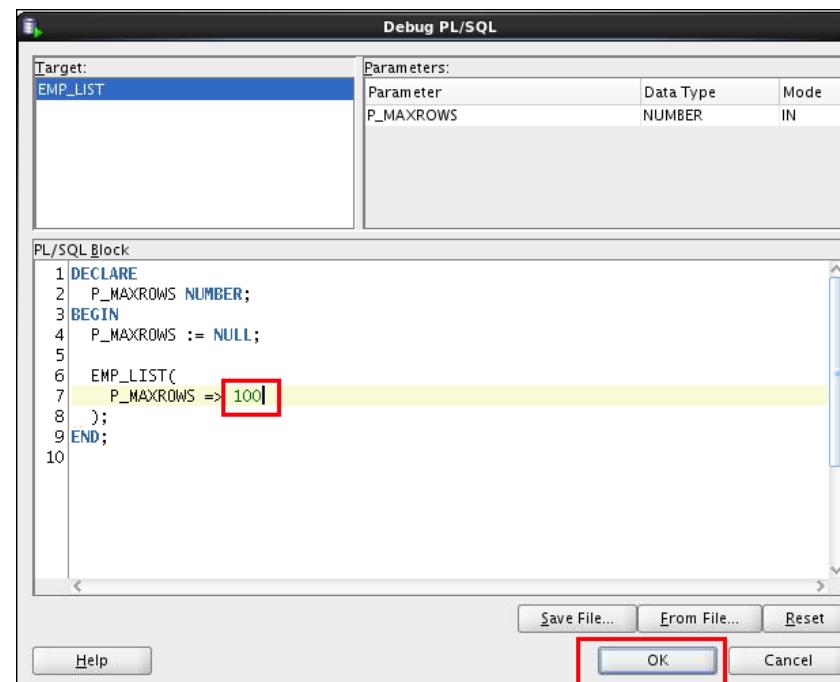
A context menu is open over the code, with the 'Compile for Debug' option highlighted by a red box. At the bottom of the interface, there are tabs for 'Messages - Log' and 'Breakpoints'. The 'Messages - Log' tab is active and contains the message 'Compiled', which is also highlighted with a red box.

- Haga clic con el botón secundario en el código y, a continuación, seleccione **Compilar para depuración** en el menú contextual

Depurando emp_list e introduciendo valores para el parámetro P_MAXROWS



Introduzca el valor del parámetro del procedimiento mediante el bloque anónimo.



PMAXROWS, especifica el número de registros a devolver

Depurando emp_list: Step Into (F7)

The screenshot shows the Oracle SQL Developer interface. On the left, the code editor displays the PL/SQL procedure `emp_list`. A green circle labeled '1' highlights the first line of code, which is a breakpoint. The code is as follows:

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

The bottom window shows the debugger log:

```
Debugging: IdeConnections%23MyDBConnection.jpr - Log X Smart Data X Watches X
Connecting to the database MyDBConnection.
Executing PL/SQL: ALTER SESSION SET PLSQL_DEBUG=TRUE
Executing PL/SQL: CALL DBMS_DEBUG_JDWP.CONNECT_TCP( '127.0.0.1', '15290' )
Debugger accepted connection from database on port 15290.
Source breakpoint occurred at line 16 of EMP_LIST.pls.
```

El control de programa se detiene en el primer punto de interrupción.

Depurando emp_list: Step Into (F7)

The diagram illustrates the step-by-step debugging process for the `emp_list` procedure using the `Step Into (F7)` function.

- Line 1:** The first line of code being executed is highlighted with a red rectangle. A green circle with the number 1 indicates the current execution point at the start of the cursor definition.
- Line 2:** The code has stepped into the cursor loop. A green circle with the number 2 indicates the current execution point at the `OPEN cur_emp;` statement. A callout box shows the step-over window with the `OPEN cur_emp;` and `FETCH cur_emp INTO rec_emp;` statements highlighted.
- Line 3:** The code has stepped into the cursor fetch loop. A green circle with the number 3 indicates the current execution point at the innermost `SELECT` statement of the cursor. A callout box shows the step-over window with the `SELECT d.department_name, e.employee_id, e.salary, e.commission_pct` statement highlighted.

Procedure Code:

```
create or replace
PROCEDURE emp_list
(p_maxrows IN NUMBER)
IS
CURSOR cur
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%ROWTYPE;
TYPE emp_tab_type IS TABLE OF cur%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;
```

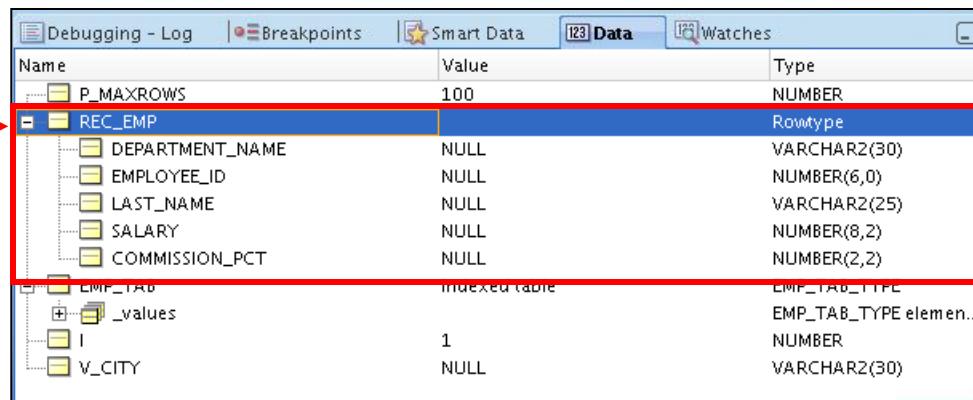
Log Window:

```
Debugging: IdeConnections%23MyDBConnection.jpr - Log x | Breakpoints x | Smart Data x | Data x
Connecting to the database MyDBConnection.
Executing PL/SQL: ALTER SESSION SET PLSQL_DEBUG=TRUE
Executing PL/SQL: CALL DBMS_DEBUG_JDWP.CONNECT_TCP( '127.0.0.1', '63138' )
Debugger accepted connection from database on port 63138.
Source breakpoint occurred at line 16 of EMP_LIST.pls.
```

Visualización de los datos

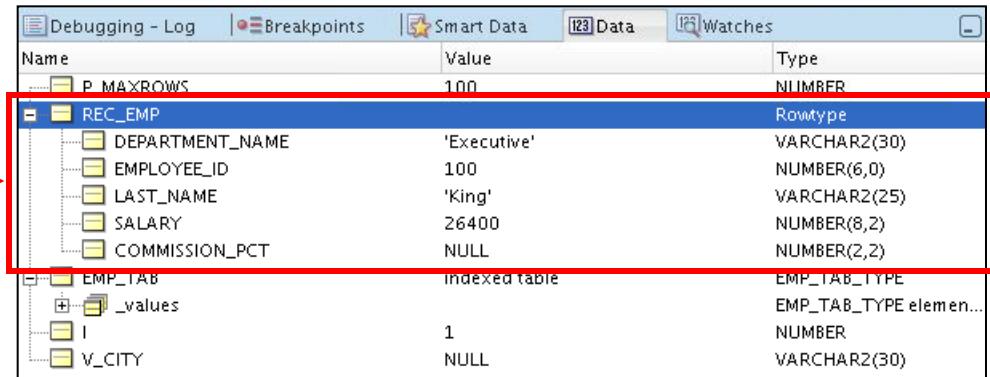
```
18 OPEN emp_cursor;
20 FETCH emp_cursor
```

Mientras está depurando su código, puede utilizar la ficha **Data** para mostrar y modificar las variables (View > Debugger)



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 elemen...
I	1	NUMBER
V_CITY	NULL	VARCHAR2(30)

```
15
16 OPEN cur_emp;
17 FETCH cur_emp INTO rec_emp;
18 emp_tab(i) := rec_emp;
19 WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
```



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

Modificación de las variables Mientras depura el código

The screenshot shows the Oracle Database Navigator interface during a debugging session. A green circle labeled '1' highlights the 'Data' tab in the top navigation bar. A red box highlights the 'I' variable in the 'Value' column of the main table. A green circle labeled '2' highlights the context menu for the 'I' variable, with the 'Modify Value...' option selected. A red box highlights the '1' value in the 'Value' column. A green circle labeled '3' highlights the '3' value entered in the 'New Value' input field of the 'Modify Value' dialog. A red box highlights the 'OK' button in the dialog. A green circle labeled '4' highlights the '3' value in the 'Value' column of the main table after modification. The bottom table shows the updated values for all variables.

Name	Value	Type
P_MAXROWS	100	NUMBER
REC_EMP	'Executive'	Rowtype
EMP_TAB	100	NUMBER(6,0)
I	3	NUMBER
V_CITY	NULL	VARCHAR2(30)

Para modificar el valor de una variable en la ficha Data, Haga clic con el botón secundario en el nombre de la variable y, a continuación, seleccione **Modificar valor**

Depurando emp_list: Step Over (F8)

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

Step Over (F8):
Ejecuta el cursor
(Igual que F7),

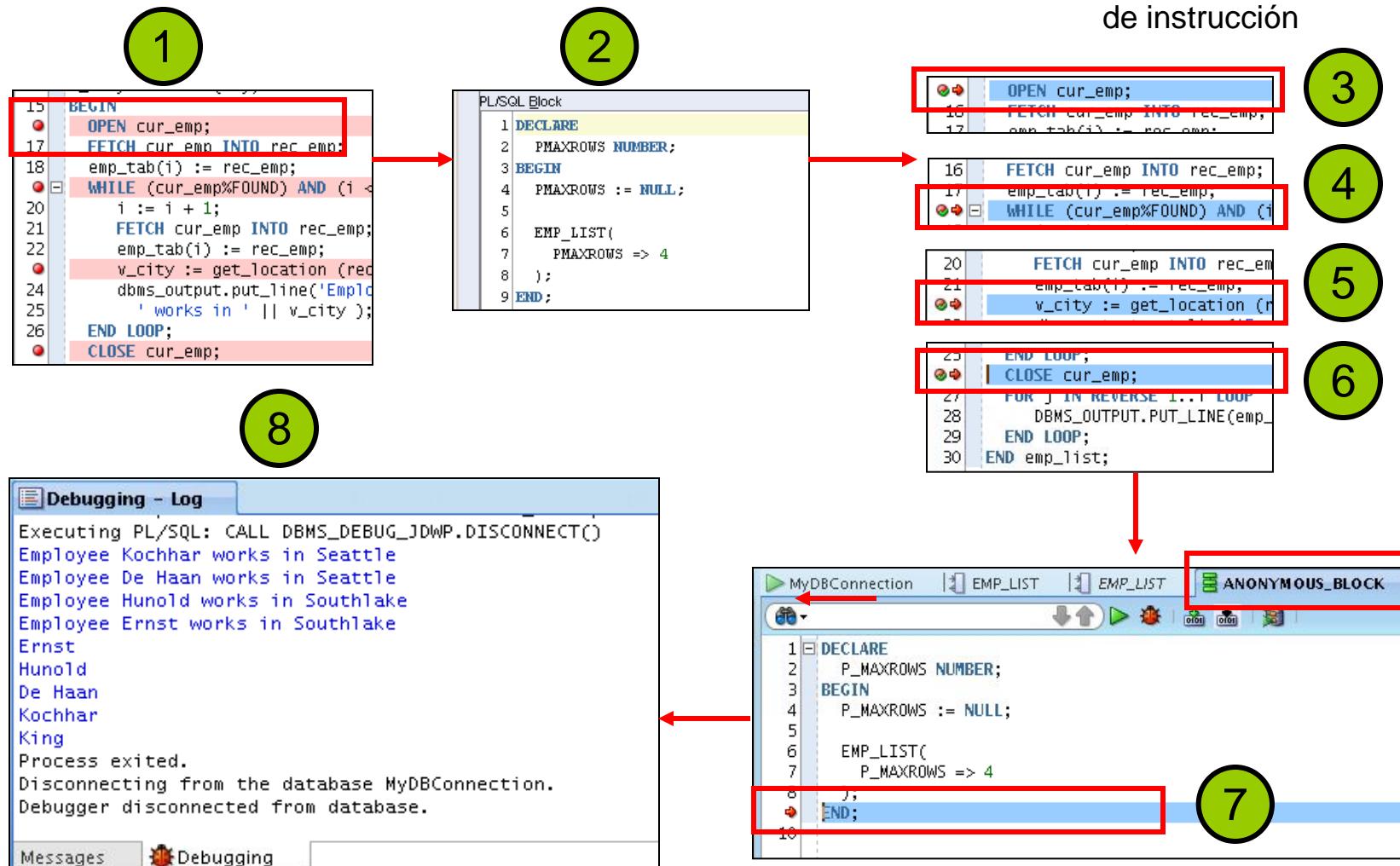
Pero el control no se transfiere
al código de cursor abierto

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

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

Depurando emp_list: Step Out (Shift + F7)

Step Out abandona el subprograma actual y pasa al siguiente subprograma de instrucción



Resumen

En esta lección, debes haber aprendido a:

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

Práctica 3-2 Descripción general: Introducción al SQL Developer Debugger

En esta lección, realiza las siguientes prácticas:

- Creación de un procedimiento y una función
- Inserción de puntos de interrupción en el procedimiento
- Compilación del procedimiento y la función para el modo de depuración
- Depuración del procedimiento y introducción en el código
- Visualización y modificación de las variables de los subprogramas

4

Creando Paquetes



ORACLE®

Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Describir los paquetes y listar sus componentes
- Crear un paquete para agrupar variables relacionadas, cursores, constantes, excepciones, procedimientos y funciones
- Designar una construcción de paquete como público o privado
Invocar una construcción de paquete
- Describir el uso de un paquete sin cuerpo

Agenda

- Identificar los beneficios y los componentes de los paquetes
- Trabajar con paquetes:
 - Creación de la especificación del paquete y del cuerpo
 - Invocación de los subprogramas de paquete
 - Visualización de la información del paquete
 - Eliminación de un paquete

¿Qué son los paquetes PL/SQL?

- Un paquete es un objeto de esquema que agrupa los tipos, variables y subprogramas PL/SQL relacionados lógicamente.
- Los paquetes suelen tener dos partes:
 - Una especificación (**cabecera**) (Pública)
 - Un **cuerpo** (Privada)
- **Cabecera**
 - Declara los tipos, variables, constantes, excepciones, cursores y subprogramas que se pueden hacer referencia desde fuera del paquete.
- **Cuerpo (opcional)**
 - Define el código para los subprogramas, y permite la declaración de tipos, variables, constantes, etc LOCALES
- Cuando se hace referencia a un paquete por primera vez, todo el paquete se carga en la memoria

Ventajas de usar paquetes

- **Modularidad:**
 - Encapsula estructuras de programación relacionadas lógicamente en un módulo con nombre
- **Mantenimiento:**
 - Cada paquete es fácil de entender
- **Diseño de aplicaciones más sencillo:**
 - Codificación y compilación de la especificación y el cuerpo por separado
- **Provisión para ocultar información:**
 - Sólo las declaraciones en la cabecera del paquete son visibles y accesibles para las aplicaciones.
 - Las construcciones **privadas** en el cuerpo del paquete están ocultas e inaccesibles.
 - Toda la codificación está oculta en el cuerpo del paquete.

Ventajas de usar paquetes

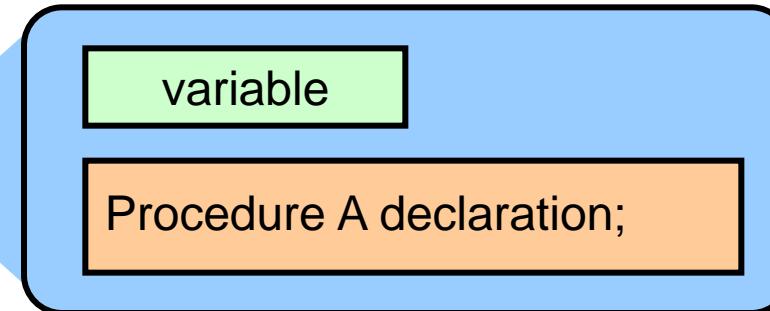
- **Funcionalidad añadida:**
 - Las variables públicas y cursores empaquetadas persisten durante la duración de una sesión.
 - Por lo tanto, pueden ser compartidos por todos los subprogramas que se ejecutan en el entorno
- **Mejor rendimiento:**
 - Todo el paquete se carga en la memoria cuando se hace referencia al paquete por primera vez.
 - Sólo hay una copia en memoria para todos los usuarios.
 - La jerarquía de dependencias se simplifica.
- **Sobrecarga:**
 - Con los paquetes, puede sobrecargar procedimientos y funciones, lo que significa que puede crear varios subprogramas con el mismo nombre en el mismo paquete

Componentes de un paquete PL/SQL

- Un paquete está formado por dos partes:
 - **Cabecera** del paquete
 - Es la parte **pública** para las aplicaciones.
 - Se utiliza para declarar los tipos públicos, variables, constantes, excepciones, cursores y subprogramas disponibles para su uso.
 - La cabecera también puede incluir PRAGMAS, que afecten a todo el paquete.
 - El **cuerpo** del paquete
 - Los componentes **privados** se colocan en el cuerpo del paquete y sólo pueden referenciarse por otras construcciones dentro del mismo cuerpo del paquete.
 - Los componentes privados pueden hacer referencia a los componentes públicos de un paquete.
 - Puede definir sus propios subprogramas y debe implementar completamente los subprogramas declarados en la parte de especificación.
 - El cuerpo del paquete también puede definir construcciones PL/SQL, como tipos, variables, constantes, excepciones y cursores.

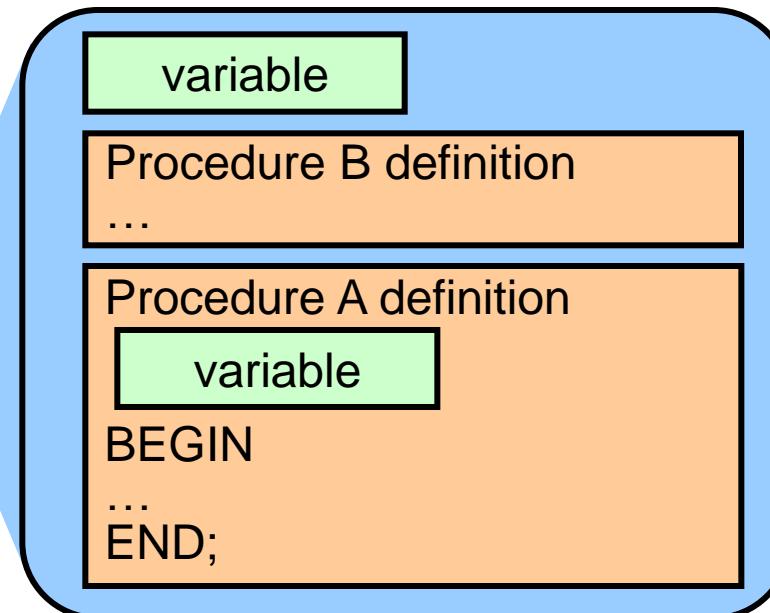
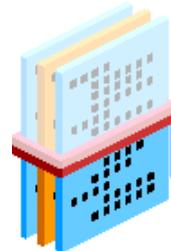
Componentes de un paquete PL/SQL

Cabecera
Paquete



→ Publico

Cuerpo del
Paquete

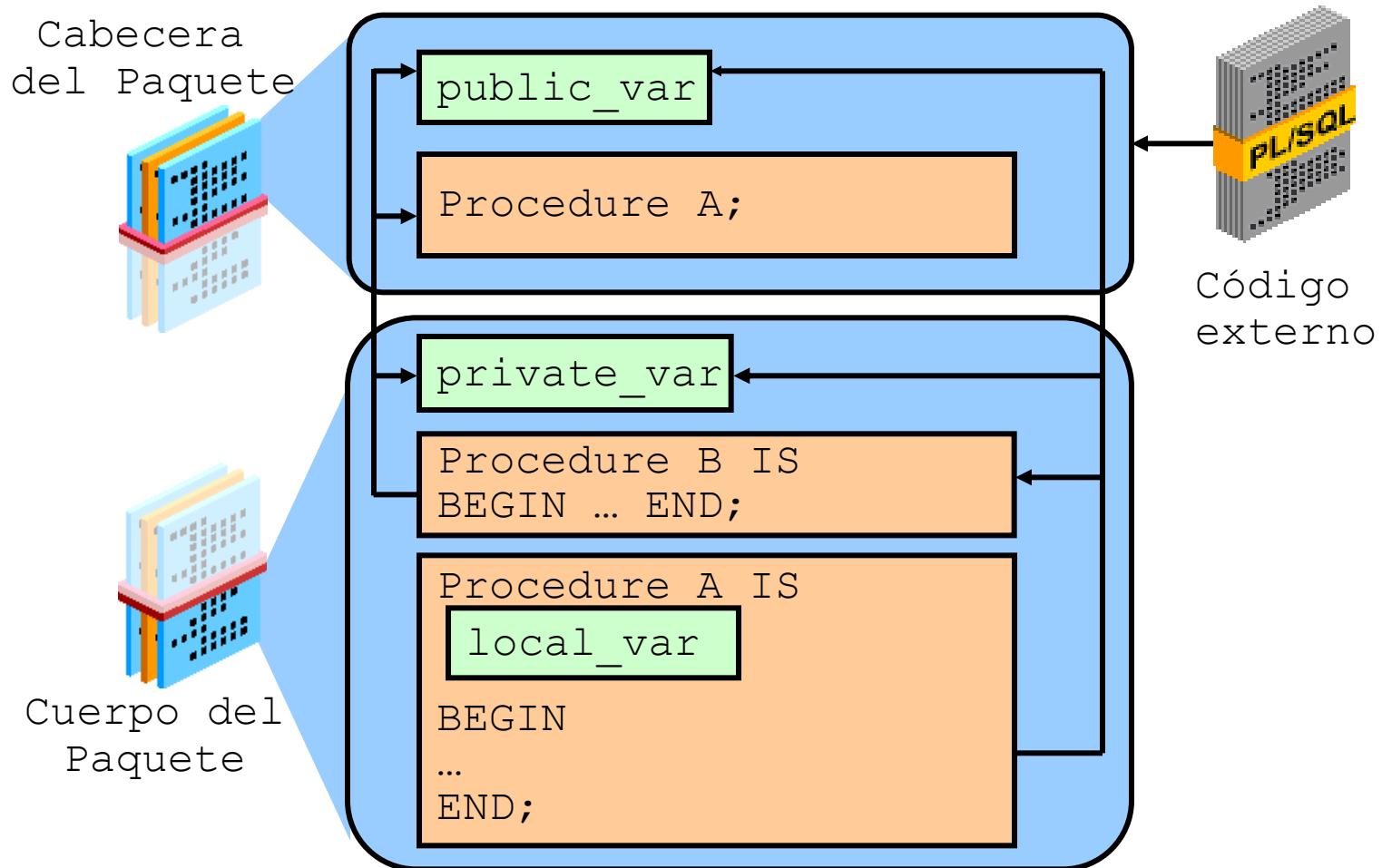


→ Privado

Visibilidad interna y externa de los componentes de un paquete

- La visibilidad de un componente describe si ese componente puede ser referenciado y utilizado por otros componentes u objetos.
 - La visibilidad de los componentes depende de si se declaran local o globalmente.
- Los componentes **locales** son visibles dentro de la estructura en la que se declaran, como por ejemplo:
 - Las variables definidas en un subprograma pueden ser referenciadas dentro de ese subprograma y no son visibles para componentes externos;
- Las variables de paquete que se declaran en el **cuerpo**, pueden ser referenciadas por otros componentes en el **mismo cuerpo de paquete**.
 - No son visibles para ningún subprograma u objeto que esté fuera del paquete

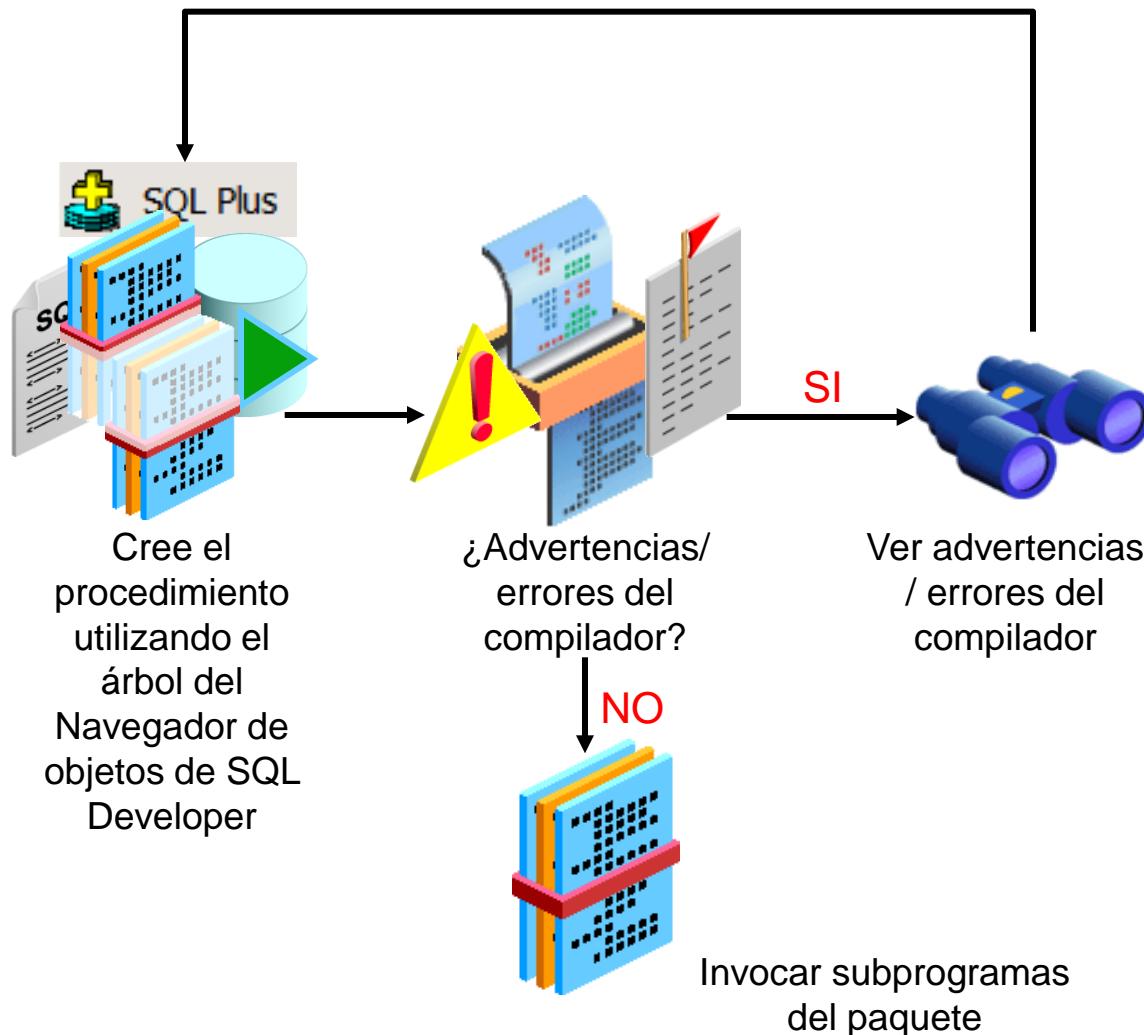
Visibilidad interna y externa de los componentes de un paquete



Visibilidad interna y externa de los componentes de un paquete

- Los componentes declarados globalmente (en la cabecera del paquete) son visibles internamente y externamente al paquete, tales como:
 - Una variable que se declara en la cabecera del paquete, puede referenciarse y cambiarse fuera del paquete.
- Un **subprograma definido en la cabecera del paquete**, puede ser llamado desde código externo al paquete
- Los **subprogramas**, declarados en el cuerpo del paquete, sólo pueden invocarse con subprogramas del mismo paquete

Desarrollo de paquetes PL/SQL: Visión general



Agenda

- Identificar los beneficios y los componentes de los paquetes
- Trabajar con paquetes:
 - Creación de la especificación del paquete y del cuerpo
 - Invocación de los subprogramas de paquete
 - Visualización de la información del paquete
 - Eliminación de un paquete

Creación de la especificación de paquete: Uso de la instrucción CREATE PACKAGE

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

package_name

Nombre del paquete

public type and variable

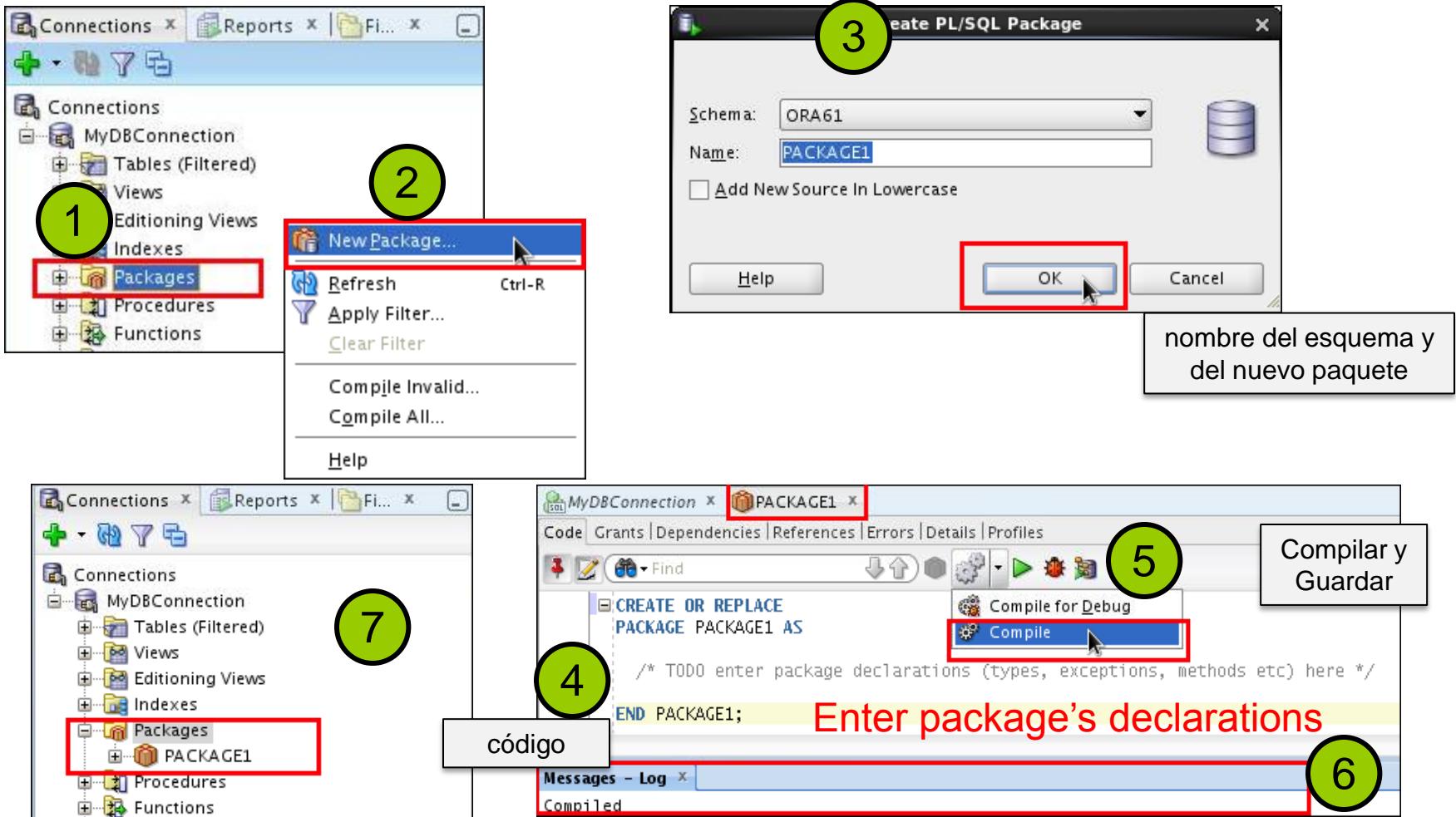
Tipos y variables definidos por el usuario

subprogram specification

Prototipos de las rutinas a implementar

- La opción OR REPLACE borrar y vuelve a crear la especificación del paquete.
- Todas las declaraciones dentro de la cabecera son públicas.
- Las variables declaradas en la especificación del paquete se inicializan en NULL de forma predeterminada.
- Todas las construcciones declaradas en una especificación de paquete son visibles para los usuarios a los que se conceden privilegios en el paquete.

Creación de la cabecera de paquete: Usando SQL Developer



Creación del cuerpo paquete: Con SQL Developer

The screenshot shows the SQL Developer interface. On the left, the Connections tree shows a connection named 'MyDBConnection' with various schema objects like Tables, Views, Procedures, and Functions. A green circle labeled '1' highlights the 'Packages' node, which contains a sub-node 'PACKAGE1'. A red box surrounds this node. On the right, a context menu is open over 'PACKAGE1'. A green circle labeled '2' highlights the 'Create Body...' option, which is also highlighted with a red box. The menu also includes options like Edit..., Export..., Compare With..., Debug, Compile, Run, Prof, Order By, Drop Package..., Grant..., Revoke..., Save Package Spec and Body..., and Quick DDL.

1. Seleccione paquete y boton derecho
2. Seleccione **Crear cuerpo**
3. Introduzca el código para el nuevo cuerpo del paquete.
4. Compile o guarde el cuerpo del paquete.
5. El cuerpo del paquete recién creado se muestra en el árbol
6. Enter package's body code

3

4

5

6

Connections x Reports x Fi... x MyDBConnection x PACKAGE1 x PACKAGE1 Body x

Code Grants Dependencies References Errors Details Profiles

CREATE OR REPLACE PACKAGE BODY PACKAGE1 AS

END PACKAGE1;

Messages - Log x Compiled

Ejemplo de una especificación de paquete: comm_pkg

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

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

- V_STD_COMM es una variable global pública inicializada a 0.10.
- RESET_COMM es un procedimiento público utilizado para restablecer la comisión estándar sobre la base de algunas reglas empresariales. Se implementa en el cuerpo del paquete.

Creación del cuerpo del paquete

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

- La opción OR REPLACE borra y vuelve a crear el cuerpo del paquete.
- Los identificadores definidos en el cuerpo del paquete son **privados** y no son visibles fuera del cuerpo del paquete.
 - Todas las construcciones privadas deben ser declaradas antes de que se hagan referencia.
- Las construcciones públicas son visibles para el cuerpo del paquete.

Ejemplo de un cuerpo de paquete: comm_pkg

```
CREATE OR REPLACE PACKAGE BODY comm_pkg IS
    FUNCTION validate(p_comm NUMBER) RETURN BOOLEAN IS
        v_max_comm          employees.commission_pct%type;
    BEGIN
        SELECT MAX(commission_pct) INTO v_max_comm
        FROM   employees;
        RETURN (p_comm BETWEEN 0.0 AND v_max_comm);
    END validate;

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

Invocación de los subprogramas del paquete: Ejemplos

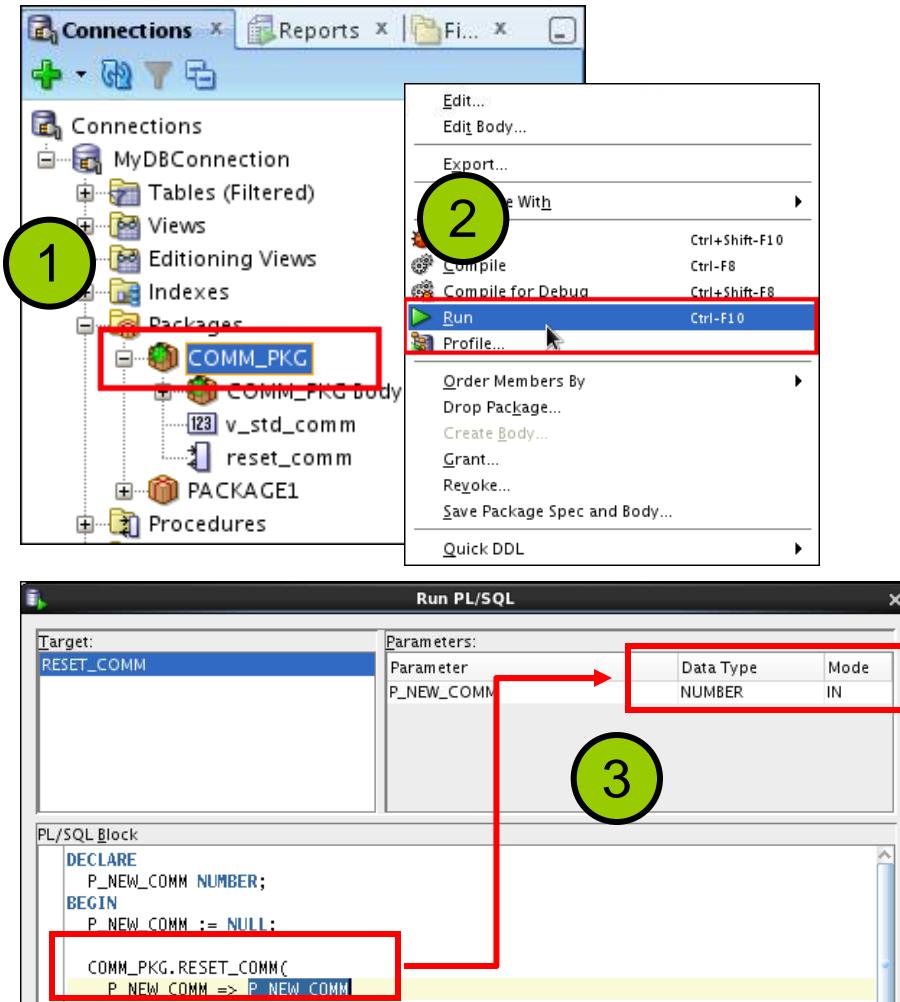
```
-- Invoke a function within the same packages:  
CREATE OR REPLACE PACKAGE BODY comm_pkg IS ...  
  PROCEDURE reset_comm(p_new_comm NUMBER) IS  
    BEGIN  
      IF validate(p_new_comm) THEN  
        v_std_comm := p_new_comm;  
      ELSE ...  
      END IF;  
    END reset_comm;  
END comm_pkg;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

Invoca la función de validación
del procedimiento reset_comm
dentro del mismo paquete.

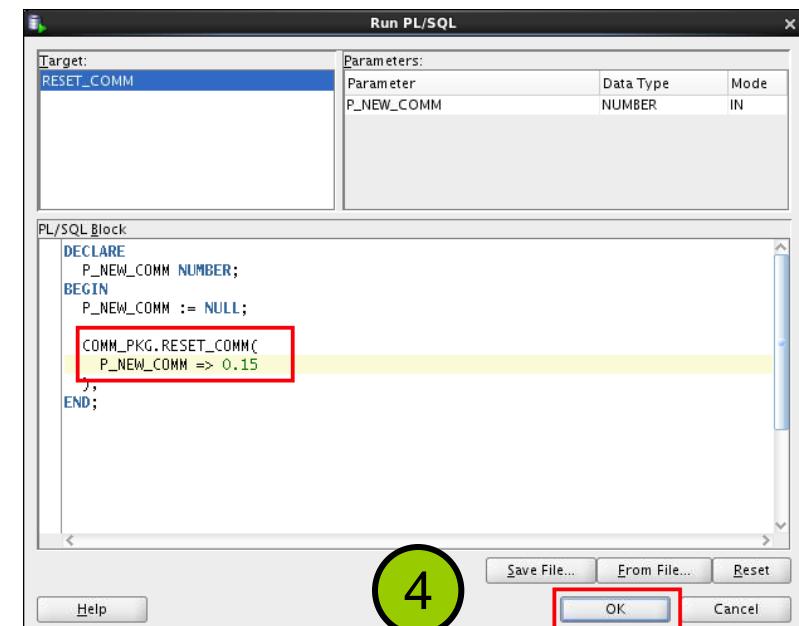
```
-- Invoke a package procedure from SQL*Plus:  
EXECUTE comm_pkg.reset_comm(0.15)
```

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

Invocar los subprogramas de paquete: Usando SQL Developer



1. Seleccione paquete y botón derecho
2. Seleccione **Ejecutar**
3. Introduzca los parámetros del procedimiento
4. Haga clic en OK para ejecutar



Creación y uso de paquetes sin cuerpo

- Las variables y las constantes declaradas dentro de los subprogramas autónomos sólo existen durante el período de ejecución del subprograma.
- Para proporcionar datos que existan durante la duración de la sesión de usuario, cree una cabecera de paquete que contenga variables públicas (globales) y declaraciones de constantes.
- En este caso, cree una especificación de paquete sin un cuerpo de paquete, conocido como paquete sin cuerpo.

Creación y uso de paquetes sin cuerpo

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

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

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

Visualización de paquetes. Data Dictionary

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

El código fuente de los paquetes PL/SQL también se almacena en las vistas del diccionario de datos **USER_SOURCE** y **ALL_SOURCE**.

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

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

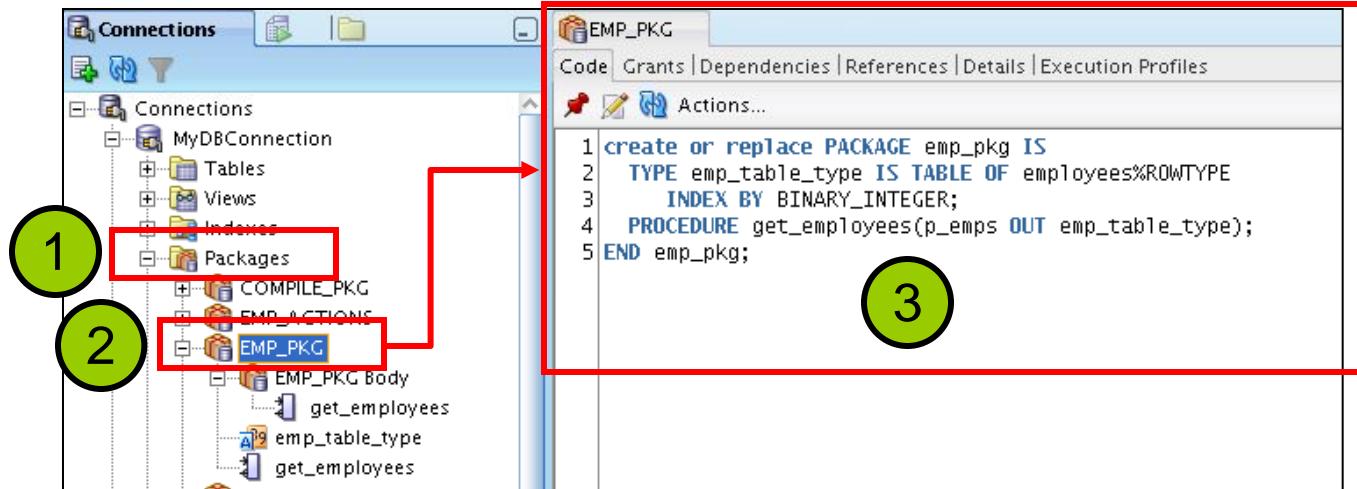
Al consultar el paquete, utilice una condición en la que la columna **TYPE** es:

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

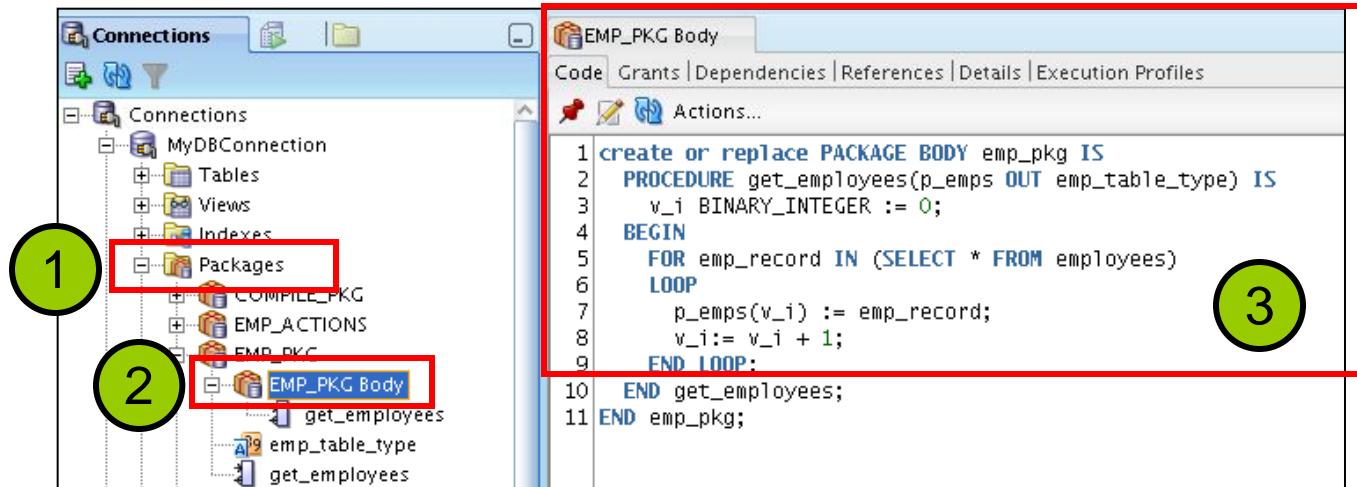
TEXT
1 PACKAGE BODY comm_pkg IS
2 FUNCTION validate(comm NUMBER) RETURN BOOLEAN IS
3 max_comm employees.commission_pct%type;
4 BEGIN
5 SELECT MAX(commission_pct) INTO max_comm
6 FROM employees;
7 RETURN (comm BETWEEN 0.0 AND max_comm);

Visualización de paquetes. SQL Developer

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

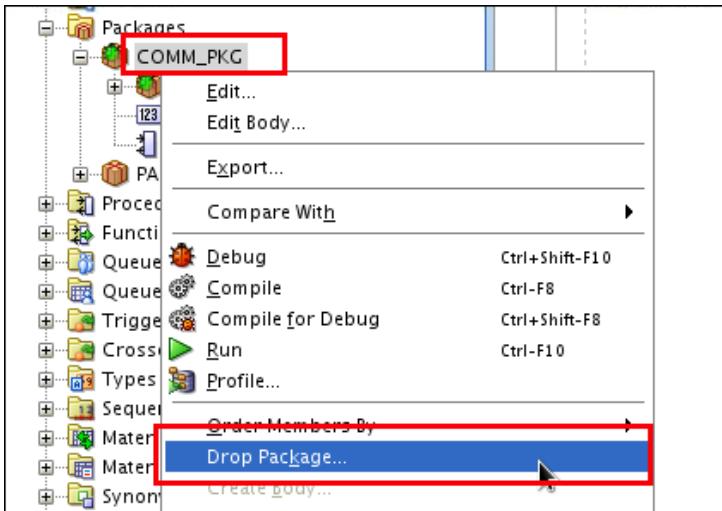


Para ver el cuerpo del paquete, haga clic en el cuerpo del paquete.

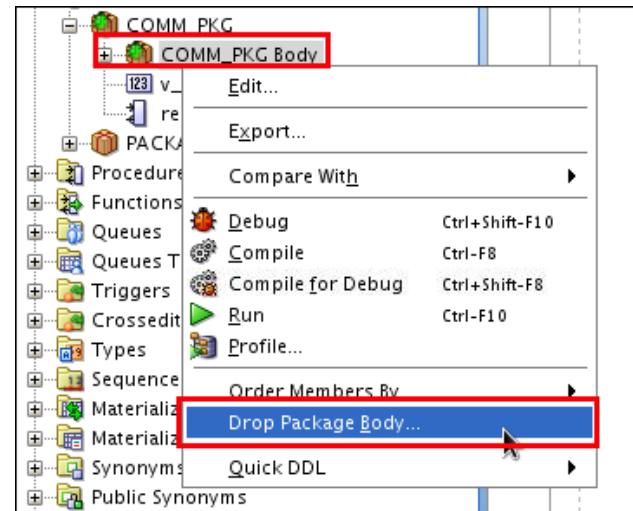


Eliminar paquetes mediante SQL Developer o la instrucción SQL DROP

Borrar la **cabecera** del paquete y el **cuerpo**.



Borrar sólo el **cuerpo** del paquete.



```
-- Eliminar la cabecera y cuerpo del paquete  
DROP PACKAGE package_name;
```

```
-- Eliminar sólo el cuerpo del paquete  
DROP PACKAGE BODY package_name;
```

Pautas para escribir paquetes

- Desarrollar paquetes para **uso general**.
- Defina la cabecera del paquete antes del cuerpo.
- La cabecera del paquete debe contener sólo aquellas construcciones que desea que sean **públicas**.
- Coloque los elementos en la parte de declaración del cuerpo del paquete cuando debe mantenerlos durante una sesión o entre transacciones.
- La gestión interna de paquetes permite **reducir la necesidad de recompilar subprogramas** de referencia cuando cambia una especificación de paquete.
- La cabecera del paquete debe contener sólo los elementos necesarios.

Quiz

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

- a. True
- b. False

Resumen

En esta lección, debes haber aprendido a:

- Describir los paquetes y listar sus componentes
- Crear un paquete para agrupar variables relacionadas, cursores, constantes, excepciones, procedimientos y funciones
- Designar una construcción de paquete como público o privado
Invocar una construcción de paquete
- Describir el uso de un paquete sin cuerpo

Prácticas 4

En esta lección, realiza las siguientes prácticas:

- Creación de paquetes
- Invocación de unidades de programa de paquete

5

Trabajando con paquetes



ORACLE®

Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Sobrecarga de Procedimientos y funciones en el paquete
- Utilizar declaraciones `forward`
- Crear un bloque de inicialización en un cuerpo de paquete
- Administrar estados de datos de paquete persistentes durante la vida de una sesión
- Utilizar matrices asociativas (tablas `index-by`) y registros en paquetes

Agenda

- Sobrecarga de subprogramas de paquete
- Gestión de estados de datos de paquetes persistentes y el uso de matrices asociativas y registros en paquetes

Sobrecarga de subprogramas en PL/SQL

- La característica de sobrecarga en PL/SQL le permite desarrollar dos o más subprogramas empaquetados con el mismo nombre.
- La sobrecarga es útil cuando:
 - Subprogramas similares, que acepte conjuntos de parámetros similares, pero el tipo o número de parámetros utilizados varía.
 - Proporcionar formas alternativas para encontrar diferentes datos con diferentes criterios de búsqueda (por fecha, nombre, etc)

Sobrecarga de subprogramas en PL/SQL

- **La regla clave:**
 - Se puede usar el mismo **nombre** para subprogramas diferentes siempre y cuando **sus parámetros formales** difieran en **número, orden o familia de tipo de datos**.
- Considere el uso de sobrecarga **cuando se encuentre:**
 - Las reglas de procesamiento para dos o más subprogramas son similares, pero el tipo o número de parámetros utilizados varía
 - Proporcionar formas alternativas para encontrar diferentes datos con diferentes criterios de búsqueda.
 - Ampliación de la funcionalidad cuando no desea reemplazar el código existente
- Nota:
 - Los subprogramas autónomos **no** pueden estar sobrecargados.

Sobrecarga de subprogramas en PL/SQL

RESTRICCIONES

- Cuando sobrecarga subprogramas con las características siguientes, se obtiene un error de tiempo de ejecución.
- No se puede sobrecargar:
 - Dos subprogramas si sus **parámetros formales difieren sólo en el tipo de datos y éstos pertenecen a la misma familia** (NUMBER y DECIMAL pertenecen a la misma familia).
 - Dos subprogramas si sus **parámetros formales difieren sólo en el subtipo y los diferentes subtipos se basan en tipos de la misma familia** (VARCHAR y STRING son subtipos PL/SQL de VARCHAR2).
 - Dos **funciones que sólo difieren en el tipo de retorno**, incluso si los tipos están en diferentes familias

Ejemplo de sobrecarga en procedimientos: Creación de la especificación del paquete

```
CREATE OR REPLACE PACKAGE dept_pkg IS
    PROCEDURE add_department
        (p_deptno departments.department_id%TYPE,
         p_name departments.department_name%TYPE := 'unknown',
         p_loc departments.location_id%TYPE := 1700);

    PROCEDURE add_department
        (p_name departments.department_name%TYPE := 'unknown',
         p_loc departments.location_id%TYPE := 1700);
END dept_pkg;
/
```

Paquete **dept_pkg** con un procedimiento sobrecargado llamado **add_department** con 3 y 2 parámetros respectivamente

Ejemplo de sobrecarga en procedimientos: Creación del cuerpo del paquete

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

Sobrecarga y el paquete STANDARD

- Un paquete denominado STANDARD define el entorno PL/SQL y declara globalmente los **tipos, excepciones y subprogramas** que están disponibles automáticamente para los programas PL/SQL.
- La mayoría de las funciones built-in están sobrecargadas. Un ejemplo es la función TO_CHAR:

```
FUNCTION TO_CHAR (p1 DATE) RETURN VARCHAR2;
FUNCTION TO_CHAR (p2 NUMBER) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 DATE, P2 VARCHAR2) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 NUMBER, P2 VARCHAR2) RETURN VARCHAR2;
. . .
```

- Si crea un subprograma con el mismo nombre que una función built-in, entonces su declaración local anula el built-in.
 - Para poder acceder al built-in, debe calificarlo con su nombre de paquete. **STANDARD.TO_CHAR**

Referencia ilegal a Procedimiento

- Los lenguajes estructurados en bloques como PL/SQL deben declarar identificadores antes de referenciarlos.
- Ejemplo de un problema de referencia:

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

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

Referencia ilegal a Procedimiento

- En general, PL/SQL es como otros lenguajes estructurados en bloques y **no permite referencias directas**.
 - Es decir, debe declarar un identificador antes de usarlo.
 - Por ejemplo, debe declararse un subprograma antes de poder llamarlo.
- Normalmente se soluciona el problema de referencia ilegal invirtiendo el orden de los dos procedimientos.
- Si esto no es posible, podríamos solucionarlo mediante declaraciones **forward** proporcionadas en PL/SQL.
 - Una declaración **forward** le permite declarar el **prototipo de un subprograma**, dentro de la definición del subprograma que lo vaya a utilizar.

Utilización de declaraciones Forward para resolver las referencias ilegales a procedimientos

En el cuerpo del paquete, una declaración forward es una especificación de subprograma privada terminada por un punto y coma.

```
CREATE OR REPLACE PACKAGE BODY forward_pkg IS
    →PROCEDURE calc_rating (...);-- forward declaration

    -- Subprograms defined in alphabetical order

    PROCEDURE award_bonus(...) IS
    BEGIN
        calc_rating (...);           -- reference resolved!
        . . .
    END;

    PROCEDURE calc_rating (...) IS -- implementation
    BEGIN
        . . .
    END;
END forward_pkg;
```

Declaración forward

- Las declaraciones `forward` ayudan a:
 - Definir subprogramas en **orden lógico o alfabético**
 - Defina subprogramas **mutuamente recursivos**.
 - Los programas mutuamente recursivos son programas que se llaman directamente o indirectamente.
 - **Agrupar y organizar lógicamente** subprogramas en un cuerpo de paquete
- Al crear una declaración `forward` :
 - Los parámetros formales deben aparecer tanto en la declaración `forward` como en el cuerpo del subprograma
 - El cuerpo del subprograma puede aparecer en cualquier lugar después de la declaración `forward`, pero ambos deben aparecer en la misma unidad de programa

Inicialización de paquetes

- La primera vez que se hace referencia a un componente de un paquete, **todo el paquete se carga en la memoria para la sesión** de usuario.
- De forma predeterminada, el valor inicial de las variables es `NULL` (si no se inicializa explícitamente).
- Para inicializar variables de paquete, puede:
 - Realizar la asignación en el momento de la declaración
 - **Añadir bloque de código al final de un cuerpo** de paquete para tareas de inicialización más complejas
- El bloque de código al final de un cuerpo de paquete, se ejecuta una vez, **cuando el paquete se invoca por primera vez** dentro de la sesión de usuario.

Inicialización de paquetes

La variable pública **v_tax** que se inicializa al valor en la tabla **tax_rates** la primera vez que se hace referencia al paquete

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

El bloque de inicialización finaliza con la palabra clave **END** del **cuerpo** del paquete.

Agenda

- Sobrecarga de subprogramas de paquete
- Gestión de estados de datos de paquetes persistentes y el uso de matrices asociativas y registros en paquetes

Estado persistente de los paquetes

- El estado del paquete es el **conjunto de valores almacenados** en todas las variables del paquete en un momento dado dentro de la sesión.
- El estado del paquete existe para la **duración de la sesión de usuario**.
- Las variables de paquete se inicializan la primera vez que se carga un paquete en la memoria para una sesión de usuario.
- **Las variables de paquete son**, de forma predeterminada, **únicas para cada sesión** y mantienen sus valores hasta que se termina la sesión de usuario.
 - Las variables se almacenan en la memoria de área global de usuario (**UGA**)

Estado persistente de los paquetes

- Dentro de los subprogramas podemos utilizar directivas del compilador (PRAGMA) que cambiar su tratamiento normal.
- PRAGMAS se procesan en tiempo de compilación, no en tiempo de ejecución.
 - No afectan el significado de un programa; Simplemente transmiten información al compilador.
- Si agrega `PRAGMA SERIALLY_RESUABLE` a la especificación del paquete, la base de datos almacena las variables del paquete en el área global del sistema (SGA) compartida entre las sesiones de usuario.
 - No se guarda en cada memoria de usuario (1 sola copia)

Estado persistente de los paquetes

- De esta manera, el área de trabajo del paquete puede ser reutilizada.
 - Cuando termina la llamada al servidor, la memoria se devuelve al pool.
- Cada vez que se reutiliza el paquete, **sus variables públicas se inicializan a sus valores predeterminados o NULL**.
- Si un paquete tiene una cabecera y un cuerpo, debe marcar ambos como **PRAGMA SERIALLY_RESUABLE**.

Los paquetes reutilizables no se pueden acceder desde **triggers** de bases de datos u otros subprogramas PL/SQL que se llaman desde sentencias SQL

Estado persistente de variables de paquete: Ejemplo

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

Estado persistente de un cursor de paquete: Ejemplo

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

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

Estado persistente de un cursor de paquete: Ejemplo

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

La declaración del cursor es **privada** para el paquete.
Por lo tanto, el estado del cursor puede verse influido invocando el procedimiento del paquete y las funciones enumeradas en la diapositiva.

Ejecutar el paquete CURS_PKG

The screenshot shows two windows from Oracle SQL Developer. The top window is a 'Worksheet' containing a PL/SQL block:

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

The bottom window is a 'Script Output' window showing the results of the execution:

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

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

Red arrows point from the text 'curs_pkg.open' in the first code line to the output 'anonymous block completed' in the first two lines of the script output, and from the text 'next(3)' in the fifth code line to the output 'Id: 103' in the second set of lines.

- El estado de una variable de paquete o cursor persiste a través de transacciones dentro de una sesión.
- Sin embargo, el estado no persiste en sesiones diferentes para el mismo usuario
- La llamada a **curs_pkg.open** abre el cursor, que permanece abierto hasta que se termina la sesión, o el cursor se cierra explícitamente
- Ejecutar secuencia de comandos (o presione F5) **de nuevo**, se muestran las siguientes tres filas

Uso de Arrays asociativas en paquetes

- Los arrays asociativas también son conocidos como index by tables
- Se puede utilizar procedimientos almacenados que lean las filas de la tabla y las devuelva utilizando variables OUT.
- Los puntos clave son los siguientes:
 - El array asociativo debe de declararse dentro de la cabecera del paquete para que sea público.
 - La variable FORMAL de ese tipo debe de ser de tipo OUT.
 - Deberemos llamar al procedimiento con una variable REAL del mismo tipo que el array asociativo creado

Uso de matrices asociativas en paquetes

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

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

```
DECLARE
    v_employees emp_pkg.emp_table_type;
BEGIN
    emp_pkg.get_employees(v_employees);
    DBMS_OUTPUT.PUT_LINE(
        'Emp 5: ||v_employees(4).last_name');
END;
```

Quiz

Sobrecarga de subprogramas en PL/SQL: (elegir todas las posibles)

- a. Permite crear dos o más subprogramas con el mismo nombre
- b. Requiere que los parámetros formales del subprograma difieran en número, orden o familia de tipo de datos
- c. Permite crear formas flexibles de invocar subprogramas con datos diferentes
- d. Proporciona una forma de ampliar la funcionalidad sin pérdida del código existente; Es decir, añadir nuevos parámetros a los subprogramas existentes

Resumen

En esta lección, debes haber aprendido a:

- Sobrecarga de Procedimientos y funciones en el paquete
- Utilizar declaraciones `forward`
- Crear un bloque de inicialización en un cuerpo de paquete
- Administrar estados de datos de paquete persistentes durante la vida de una sesión
- Utilizar matrices asociativas (tablas `index-by`) y registros en paquetes

Práctica 5

En esta lección, realiza las siguientes prácticas:

- Uso de subprogramas sobrecargados
- Creación de un bloque de inicialización de paquetes
- Utilización de una declaración `forward`

6

Uso de paquetes suministrados por Oracle



ORACLE®

Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Describir cómo funciona el paquete DBMS_OUTPUT
- Utilice UTL_FILE para dirigir la salida a los archivos del sistema operativo
- Describir las principales características de UTL_MAIL

Agenda

- Identificar los beneficios de usar los paquetes suministrados por Oracle y enumerar algunos de esos paquetes
- Utilizando los siguientes paquetes suministrados por Oracle:
 - DBMS_OUTPUT
 - UTL_FILE
 - UTL_MAIL

Uso de paquetes suministrados por Oracle

- Los paquetes suministrados por Oracle:
 - Se proporcionan con el servidor de Oracle
 - Ampliar la funcionalidad de la base de datos
 - Habilitar el acceso a ciertas funciones de SQL que normalmente están restringidas para PL/SQL
- La mayoría de los paquetes estándar se crean ejecutando el script **catproc.sql**
- El paquete `DBMS_OUTPUT` es el que estarás más familiarizado durante este curso, fue diseñado originalmente para depurar programas PL/SQL.

Ejemplos de algunos paquetes suministrados por Oracle

A continuación se muestra una lista abreviada de algunos paquetes suministrados por Oracle:

Paquete	Descripción
DBMS_OUTPUT	Proporciona depuración y almacenamiento en búfer
UTL_FILE	Permite la lectura y escritura de archivos de texto del S.O
UTL_MAIL	Permite componer y enviar mensajes de correo electrónico.
DBMS_ALERT	Admite la notificación asincrónica de eventos de base de datos.
DBMS_LOCK	Se utiliza para solicitar, combinar y liberar bloqueos a través de los servicios de Oracle Lock Management.
DBMS_SESSION	Permite el uso programático de la instrucción SQL ALTER SESSION y otros comandos de nivel de sesión
DBMS_APPLICATION_INFO	Puede utilizarse con Oracle Trace y SQL trace
HTP	Escribe datos con HTML en los búferes de la base de datos
DBMS_SCHEDULER	Permite programar y automatizar la ejecución de bloques PL/SQL, procedimientos almacenados y procedimientos externos y ejecutables

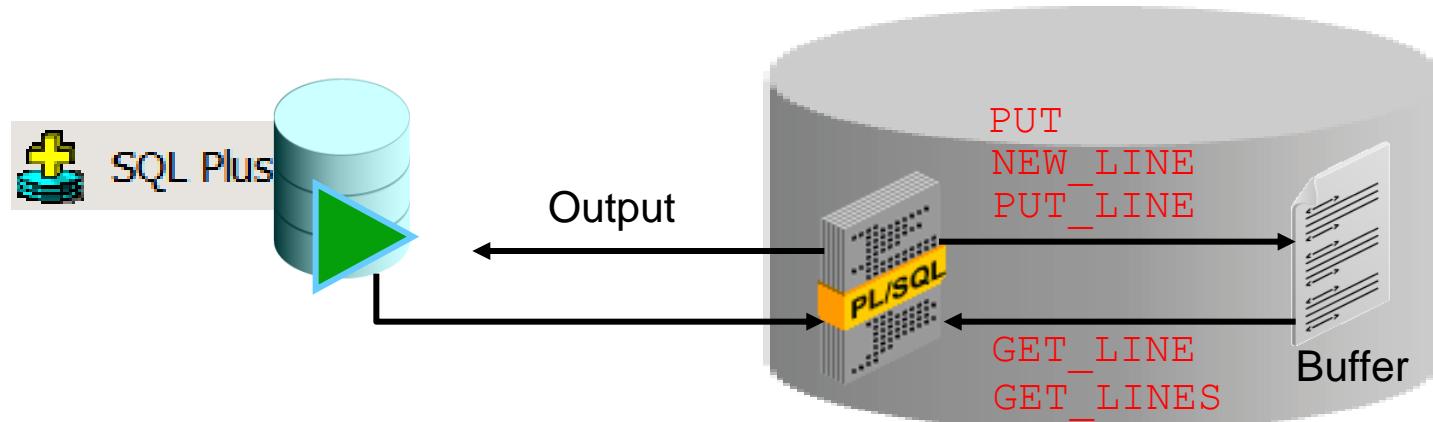
Agenda

- Identificar los beneficios de usar los paquetes suministrados por Oracle y enumerar algunos de esos paquetes
- Utilizando los siguientes paquetes suministrados por Oracle:
 - DBMS_OUTPUT
 - UTL_FILE
 - UTL_MAIL

Cómo funciona el paquete DBMS_OUTPUT

- El paquete DBMS_OUTPUT envía mensajes de texto de cualquier bloque PL/SQL a un búfer en la base de datos.
- Los procedimientos proporcionados por el paquete incluyen lo siguiente:
 - `PUT` anexa texto a la línea actual del búfer de salida de línea.
 - `NEW_LINE` coloca un marcador de fin de línea en el búfer de salida.
 - `PUT_LINE` combina la acción de `PUT` y `NEW_LINE`
 - `GET_LINE` recupera la línea actual del búfer en una variable
 - `GET_LINES` recupera una matriz de líneas en una variable
 - `ENABLE/DISABLE` activa y desactiva las llamadas a los procedimientos DBMS_OUTPUT.
- El tamaño del búfer se puede establecer mediante la opción **SIZE n**
 - `SET SERVEROUTPUT ON SIZE n`
 - El mínimo es de **2.000** y el máximo es ilimitado. El valor predeterminado es **20.000**.

Cómo funciona el paquete DBMS_OUTPUT

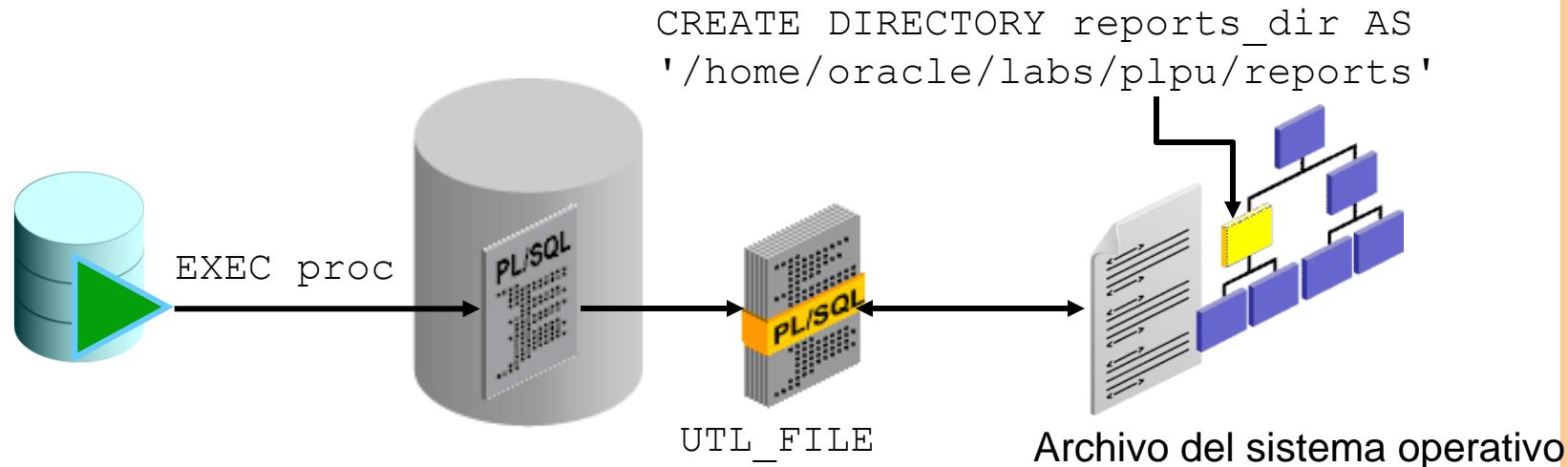


```
SET SERVEROUT ON [SIZE n]  
EXECUTE proc
```

Utilizar el paquete UTL_FILE para interactuar con archivos del sistema operativo

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

- La base de datos proporciona acceso de lectura y escritura a **directorios** específicos del sistema operativo mediante la orden **CREATE DIRECTORY**
 - Asocia un alias con un directorio del sistema operativo
 - El privilegio **CREATE DIRECTORY** sólo se concede a SYS y SYSTEM por defecto.

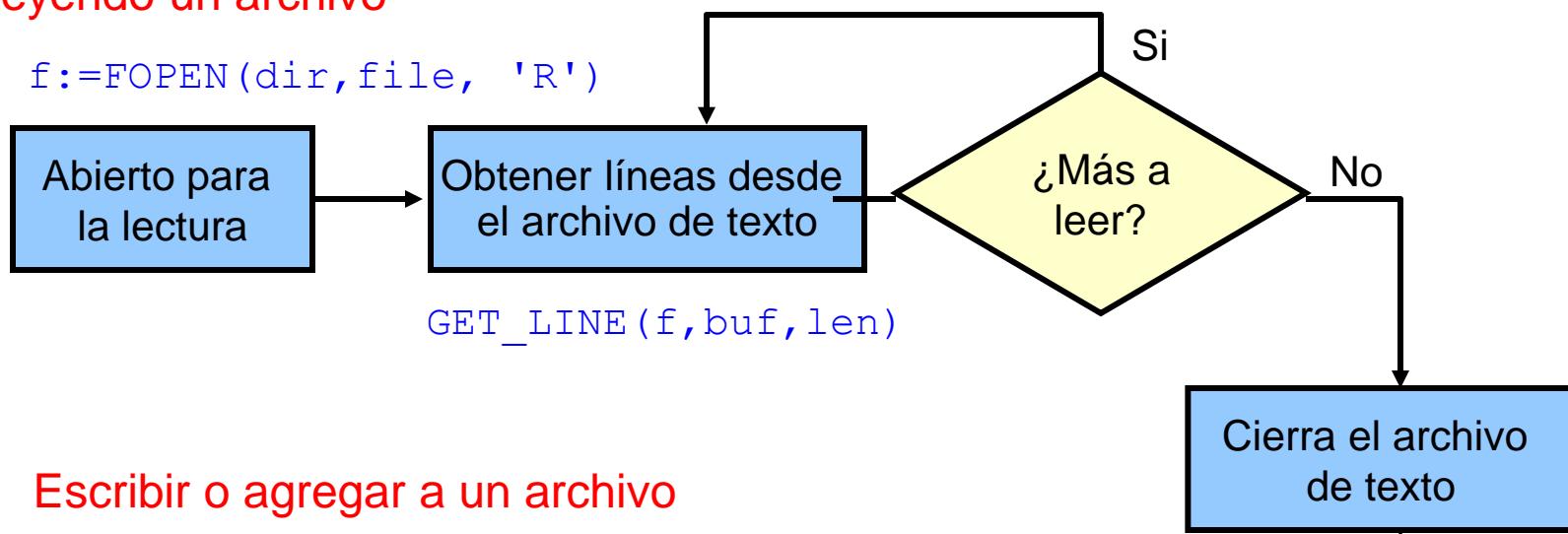


Algunos de los procedimientos y funciones de UTL_FILE

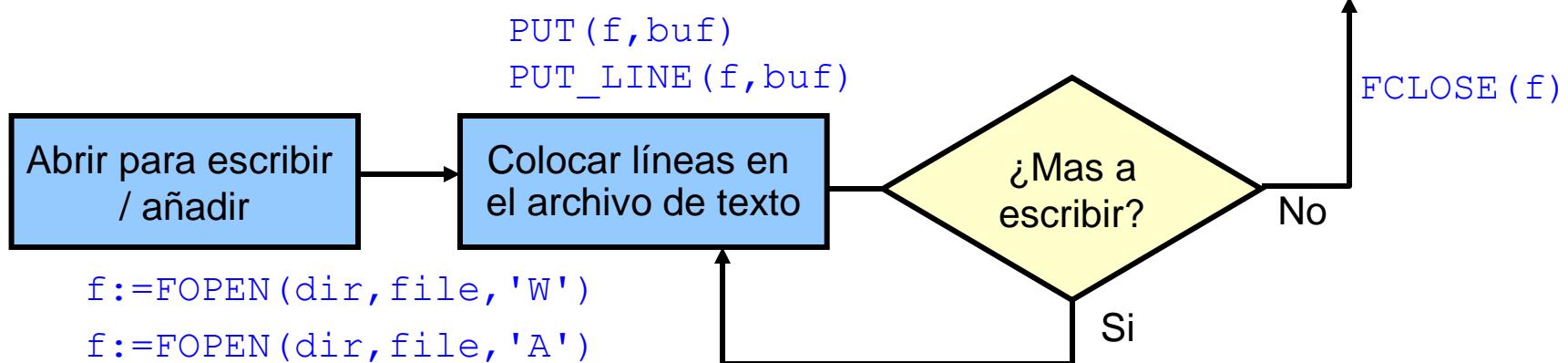
Subprograma	Descripción
<code>ISOPEN</code> function	Determina si un archivo está abierto
<code>FOPEN</code> function	Abre un archivo para entrada o salida
<code>FCLOSE</code> function	Cierra todos los identificadores de archivos abiertos
<code>FCOPY</code> procedure	Copia una porción contigua de un archivo en un archivo recién creado
<code>FGETATTR</code> procedure	Lee y devuelve los atributos de un archivo de disco
<code>GET_LINE</code> procedure	Lee el texto de un archivo abierto
<code>FREMOVE</code> procedure	Elimina un archivo de disco, si tiene suficientes privilegios
<code>FRENAME</code> procedure	Cambia el nombre de un archivo existente a un nuevo nombre
<code>PUT</code> procedure	Escribe una cadena en un archivo
<code>PUT_LINE</code> procedure	Escribe una línea en un archivo y, por tanto, agrega un terminador de línea específico del sistema operativo

Procesamiento de archivos mediante el paquete UTL_FILE: Visión general

Leyendo un archivo



Escribir o agregar a un archivo



Uso de las excepciones declaradas disponibles en el paquete UTL_FILE

Nombre de excepción	Descripción
INVALID_PATH	La ubicación del archivo no es válida
INVALID_MODE	El parámetro <code>open_mode</code> en <code>FOPEN</code> no es válido
INVALID_HANDLE	Handle de archivo no es válido
INVALID_OPERATION	No se pudo abrir ni operar el archivo como se solicitó
READ_ERROR	Se produjo un error del sistema operativo durante la operación de lectura
WRITE_ERROR	Se produjo un error del sistema operativo durante la operación de escritura
INTERNAL_ERROR	Error no especificado de PL/SQL

Funciones FOPEN y IS_OPEN : Ejemplo

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

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

Directorio
Nombre_Archivo
Modo_Aertura (r,w,a)

Descriptor Fichero

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

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

Funciones FOPEN y IS_OPEN : Ejemplo

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

Usando UTL_FILE: Ejemplo

En el ejemplo de la diapositiva, el procedimiento **sal_status** crea un informe de empleados para cada departamento, junto con sus salarios

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

```

Los datos se escriben en un archivo de texto utilizando el paquete UTL_FILE.

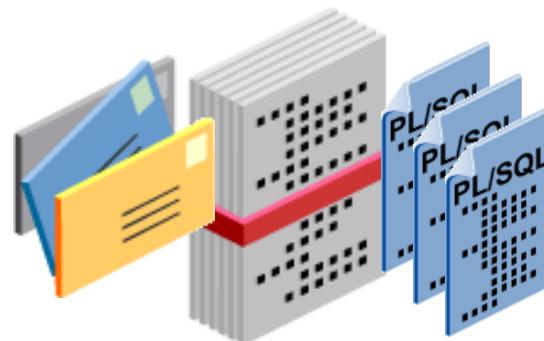
Usando UTL_FILE: Ejemplo

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

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

¿Qué es el paquete UTL_MAIL?

- Una utilidad para administrar el correo electrónico
- Requiere la configuración del parámetro de inicialización de la base de datos `SMTP_OUT_SERVER`
- Proporciona los siguientes procedimientos:
 - `SEND` para mensajes sin adjuntos
 - `SEND_ATTACH_RAW` para mensajes con archivos adjuntos binarios
 - `SEND_ATTACH_VARCHAR2` para mensajes con archivos adjuntos de texto



¿Qué es el paquete UTL_MAIL?

- El paquete UTL_MAIL no se instala de forma predeterminada.
- Al instalar UTL_MAIL , debe tomar medidas para evitar que el puerto definido por SMTP_OUT_SERVER sea inundado por transmisiones de datos.
- Para instalar UTL_MAIL , inicie sesión como usuario de DBA en SQL*Plus y ejecute las siguientes secuencias de comandos:
`@$ORACLE_HOME/rdbms/admin/utlmail.sql`
`@$ORACLE_HOME/rdbms/admin/prvtmail.plb`

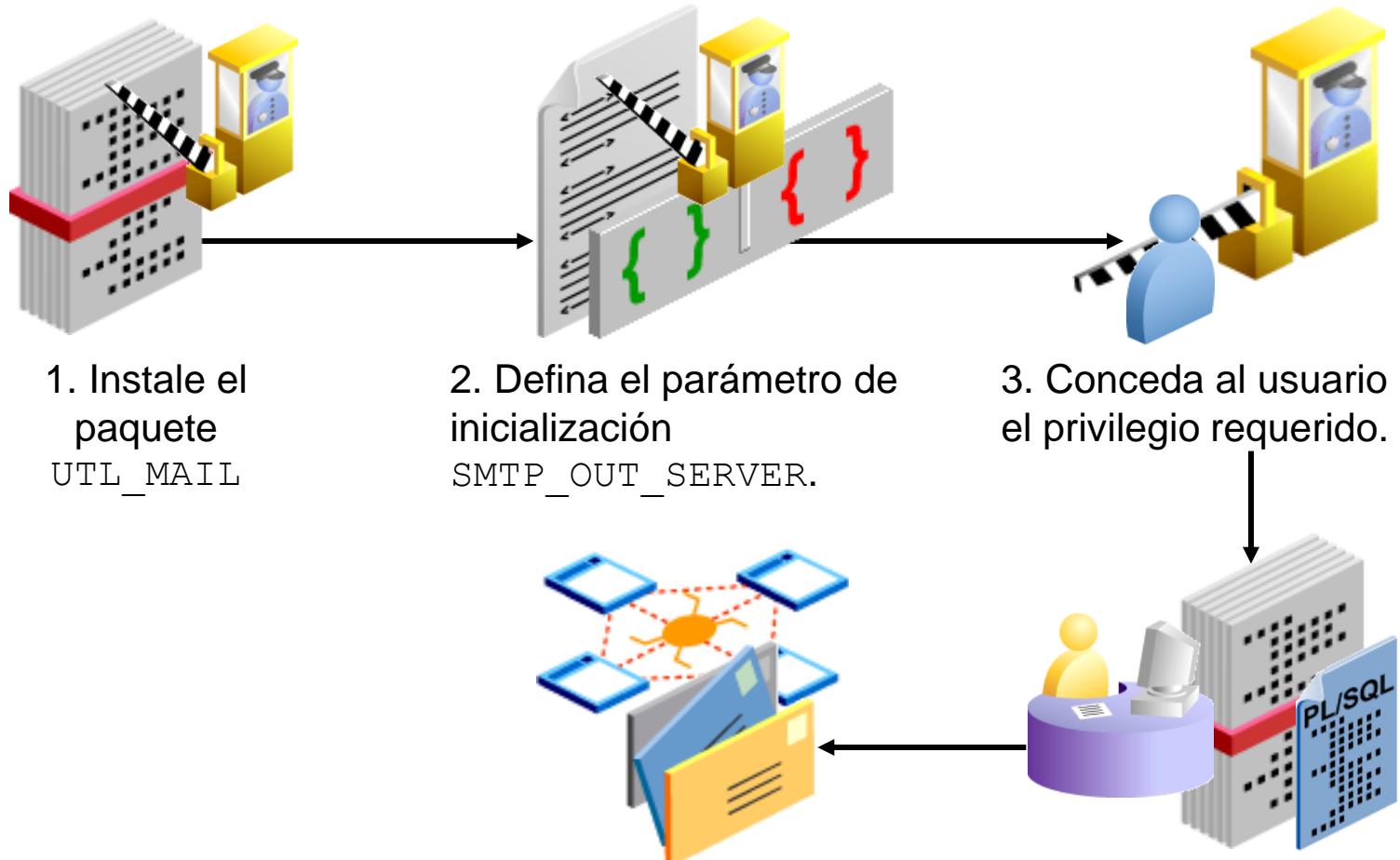
Debe definir el parámetro SMTP_OUT_SERVER en el archivo de inicialización de la base de datos de archivos init.ora:

`SMTP_OUT_SERVER=mysmtpserver.mydomain.com`

¿Qué es el paquete UTL_MAIL?

- El parámetro `SMTP_OUT_SERVER` especifica el host SMTP y el puerto a los que `UTL_MAIL` envía el correo electrónico saliente.
- Se pueden especificar varios servidores, separados por comas.
 - Si el primer servidor de la lista no está disponible, `UTL_MAIL` intenta el segundo servidor, y así sucesivamente.
- Si no se define `SMTP_OUT_SERVER`, se invoca una configuración predeterminada derivada del parámetro `DB_DOMAIN`.
- Por ejemplo:
`Db_domain = mydomain.com`

Configuración y uso de la UTL_MAIL: Visión general



Resumen de subprogramas UTL_MAIL

Subprogramas	Descripcion
SEND procedure	Envía un mensaje de correo electrónico, localiza la información SMTP y envía el mensaje al servidor SMTP para reenviarlo a los destinatarios
SEND_ATTACH_RAW Procedure	Representa el procedimiento SEND sobrecargado para los archivos adjuntos RAW – Binarios
SEND_ATTACH_VARCHAR2 Procedure	Representa el procedimiento SEND sobrecargado para los archivos adjuntos VARCHAR2 – texto

Instalación y usando UTL_MAIL

- En la siguiente diapositiva muestra cómo configurar el parámetro `SMTP_OUT_SERVER` con el nombre del host SMTP de su red y cómo instalar el paquete `UTL_MAIL` que no está instalado de forma predeterminada.
- Cambiar el parámetro `SMTP_OUT_SERVER` requiere reiniciar la instancia de la base de datos. Estas tareas son realizadas por un usuario con capacidades `SYSDBA`.
- También se muestra como enviar un mensaje de texto mediante el procedimiento `UTL_MAIL SEND` con al menos un asunto y el mensaje.

Instalación y usando UTL_MAIL

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

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

- Establezca SMTP_OUT_SERVER

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

- Como desarrollador, invoque un procedimiento UTL_MAIL:

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

Sintaxis del procedimiento SEND

Envía un mensaje de correo electrónico en el formato apropiado,

```
UTL_MAIL.SEND (
    sender      IN      VARCHAR2 CHARACTER SET ANY_CS,
                    ## Dirección de correo electrónico del remitente
    recipients   IN      VARCHAR2 CHARACTER SET ANY_CS,
                    ## Destinatarios
    cc          IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
                    ## Con copia a
    bcc         IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
                    ## Con copia oculta
    subject      IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
                    ## Asunto
    message      IN      VARCHAR2 CHARACTER SET ANY_CS,
                    ## Mensaje
    mime_type    IN      VARCHAR2 DEFAULT 'text/plain',
                    ## Tipo de Mensaje
    priority     IN      PLS_INTEGER DEFAULT NULL);
                    ## Prioridad
```

El procedimiento SEND_ATTACH_RAW

Representa el procedimiento SEND sobrecargado para los archivos adjuntos
RAW - Binarios

```
UTL_MAIL.SEND_ATTACH_RAW (
    sender          IN      VARCHAR2 CHARACTER SET ANY_CS,
    recipients      IN      VARCHAR2 CHARACTER SET ANY_CS,
    cc              IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    bcc             IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    subject         IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    message         IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    mime_type       IN      VARCHAR2 DEFAULT CHARACTER SET ANY_CS
                            DEFAULT 'text/plain; charset=us-ascii',
    priority        IN      PLS_INTEGER DEFAULT 3,
    attachment      IN      RAW,
        ## Archivo adjunto RAW. Maximo 32.767 caracteres binarios
    att_inline      IN      BOOLEAN DEFAULT TRUE,
        ## Especifica si el archivo adjunto está visible en linea con el
        ## cuerpo del mensaje
    att_mime_type   IN      VARCHAR2 CHARACTER SET ANY_CS
        ## Formato del archivo DEFAULT 'text/plain; charset=us-ascii');
    att_filename    IN      VARCHAR2 CHARACTER SET ANY_CS );
        ## asigna cualquier nombre (alias) de archivo al archivo
```

Envío de email con un archivo adjunto binario: Ejemplo

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

La función **get_image** en el ejemplo utiliza un BFILE para leer los datos de la imagen
Sig. Transparencia

Envío de email con un archivo adjunto binario: Ejemplo

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

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

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

El procedimiento SEND_ATTACH_VARCHAR2

Representa el procedimiento SEND sobrecargado para los archivos adjuntos
VARCHAR2 – texto

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

Envío de Email con un archivo adjunto de texto: Ejemplo

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

La función **get_file** en el ejemplo utiliza un BFILE para leer un archivo de texto,
Sig. Transparencia

Envío de Email con un archivo adjunto de texto: Ejemplo

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

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

Quiz

El paquete `UTL_FILE` suministrado por Oracle se utiliza para acceder a archivos de texto en el sistema operativo del servidor de bases de datos.

La base de datos proporciona funcionalidad a través de objetos de directorio para permitir el acceso a directorios específicos del sistema operativo.

- a. True
- b. False

Resumen

En esta lección, debes haber aprendido a:

- Describir cómo funciona el paquete DBMS_OUTPUT
- Utilice UTL_FILE para dirigir la salida a los archivos del sistema operativo
- Describir las principales características de UTL_MAIL

Práctica 6

Esta práctica cubre cómo usar UTL_FILE para generar un informe de texto.

7

Uso de SQL dinámico



ORACLE®

Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Describir el flujo de ejecución de sentencias SQL
- Generar y ejecutar instrucciones SQL de forma dinámica mediante Native Dynamic SQL (NDS)
- Identifique situaciones en las que debe utilizar el paquete DBMS_SQL en lugar de NDS para generar y ejecutar instrucciones SQL de forma dinámica.

Agenda

- Usando Native Dynamic SQL (NDS)
- Uso del paquete DBMS_SQL

Flujo de ejecución de sentencias SQL

- Todas las sentencias de SQL pasan por algunas o todas las siguientes etapas:
 - Parse Análisis sintáctico
 - Bind Obtención de valores para las variables de intercambio
 - Execute El servidor tiene toda la información y la sentencia se ejecuta
 - Fetch Es aplicable a las consultas, las filas se seleccionan y se gestionan
- Algunas etapas pueden no ser relevantes para todas las declaraciones:
 - La fase de `fetch` es aplicable a las consultas.
 - Para sentencias SQL SELECT, DML, MERGE, COMMIT, SAVEPOINT y ROLLBACK, las fases de `parse` y `bind` se realizan en tiempo de compilación.
 - Para sentencias SQL dinámicas, todas las fases se realizan en tiempo de ejecución

Trabando con SQL dinámico

- Las sentencias SQL incorporadas disponibles en PL/SQL se limitan a SELECT, INSERT, UPDATE, DELETE, MERGE, COMMIT y ROLLBACK.
- Todas las anteriores, se analizan en tiempo de compilación, es decir, tienen una estructura fija.
- Utilice SQL dinámico para crear una sentencia SQL cuya estructura puede cambiar y ejecutarse en tiempo de ejecución.
- Las sentencias SQL dinámicas se pueden ejecutar:
 - Modo nativo con `EXECUTE IMMEDIATE`
 - Mediante el paquete `DBMS_SQL`

Trabando con SQL dinámico

SQL dinámico:

- Se construye y se almacena como una cadena de caracteres, una variable de cadena o una expresión de cadena dentro de la aplicación
- Es una instrucción SQL con datos de columna variables o diferentes condiciones con o sin variables bind.
- Permite que sentencias DDL o DCL puedan ser ejecutadas desde PL/SQL
- Su sintaxis se comprueba en tiempo de ejecución en lugar de en tiempo de compilación
- Se ejecuta con sentencias `Native Dynamic SQL` o el paquete `DBMS_SQL`

Uso de SQL dinámico

- Utilice SQL dinámico cuando el texto completo de la instrucción SQL dinámica es **desconocido** hasta el tiempo de ejecución.
- Utilice SQL dinámico cuando uno de los elementos siguientes **no se conoce** en tiempo de precompilación:
 - Texto de la instrucción SQL, como comandos, cláusulas, etc.
 - El número y los tipos de datos de las variables del host
 - Referencias a objetos de base de datos como tablas, columnas, índices, secuencias, nombres de usuario y vistas
- Utilice SQL dinámico para que sus programas PL/SQL sean más generales y flexibles.

Native Dynamic SQL (NDS)

- Native Dynamic SQL proporciona la capacidad de ejecutar dinámicamente sentencias SQL cuya estructura se construye en tiempo de ejecución
- Proporciona soporte **nativo** para SQL dinámico directamente en el lenguaje PL/SQL.
- Las siguientes sentencias se han agregado o extendido en PL/SQL para admitir Native Dynamic SQL:
 - EXECUTE IMMEDIATE
 - OPEN-FOR
 - FETCH
 - CLOSE

Native Dynamic SQL (NDS)

EXECUTE IMMEDIATE

- Prepara una sentencia, la ejecuta, devuelve variables y luego desasigna recursos

OPEN-FOR

- Prepara y ejecuta una sentencia utilizando una variable de cursor

FETCH

- Recupera los resultados de una sentencia abierta utilizando la variable de cursor

CLOSE

- Cierra el cursor utilizado por la variable de cursor y desasigna recursos

- Puede utilizar variables de enlace en los parámetros dinámicos en las instrucciones EXECUTE IMMEDIATE y OPEN

Uso de la instrucción EXECUTE IMMEDIATE

Utilice la sentencia `EXECUTE IMMEDIATE` de NDS para bloques anónimos PL/SQL:

```
EXECUTE IMMEDIATE dynamic_string  
[INTO] {define_variable  
[, define_variable] ... | record} ]  
[USING] [IN|OUT|IN OUT] bind_argument  
[, [IN|OUT|IN OUT] bind_argument] ... ];
```

- **Dynamic_string**
 - Es una expresión de cadena que representa una instrucción SQL dinámica
- **INTO**
 - Se utiliza para consultas de una sola fila y especifica las variables o registros en los que se recuperan los valores de columna.
- **USING**
 - Se utiliza para indicar los argumentos, tipo y modo
- **Bind_argument**
 - Es una expresión cuyo valor se pasa a la sentencia SQL dinámica o al bloque PL/SQL.

Métodos disponibles para utilizar NDS

Method #	Tipo de sentencia SQL	NDS Sentencias SQL utilizadas
Method 1	<i>NO-Query sin variables bind</i>	EXECUTE IMMEDIATE sin clausula USING ni INTO
Method 2	<i>No-query con un número conocido de variables de host de entrada</i>	EXECUTE IMMEDIATE con la clausula USING
Method 3	<i>Query con un número conocido de elementos y variables de host de entrada</i>	EXECUTE IMMEDIATE con las clausulas USING y INTO
Method 4	<i>Query con un número NO conocido de elementos y variables de host de entrada</i>	Utilice el paquete DBMS_SQL

SQL dinámico con una instrucción DDL: ejemplos

```
-- Create a table using dynamic SQL
CREATE OR REPLACE PROCEDURE create_table(
    p_table_name VARCHAR2, p_col_specs VARCHAR2) IS
BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE ' || p_table_name ||
                       ' (' || p_col_specs || ')';
END;
/
```

Procedimiento **create_table** acepta el nombre de la tabla y las definiciones de columna (especificaciones) como parámetros.

```
-- Call the procedure
BEGIN
    create_table('EMPLOYEE_NAMES',
                 'id NUMBER(4) PRIMARY KEY, name VARCHAR2(40)');
END;
/
```

SQL dinámico con una instrucción DDL: ejemplos

```
CREATE OR REPLACE PROCEDURE add_col(p_table_name
VARCHAR2, p_col_spec VARCHAR2) IS
    v_stmt VARCHAR2(100) := 'ALTER TABLE ' || p_table_name
                           || ' ADD ' || p_col_spec;
BEGIN
    EXECUTE IMMEDIATE v_stmt;
END;
/
```

Procedimiento **add_col** se utiliza para añadir una columna a una tabla.

```
-- Call the procedure
BEGIN
    EXECUTE add_col('employee_names',
                     'salary number(8,2)')
END;
/
```

SQL dinámico con instrucciones DML

```
-- Delete rows from any table:  
CREATE FUNCTION del_rows(p_table_name VARCHAR2)  
RETURN NUMBER IS  
BEGIN  
    EXECUTE IMMEDIATE 'DELETE FROM ' || p_table_name;  
    RETURN SQL%ROWCOUNT;  
END;  
/  
BEGIN DBMS_OUTPUT.PUT_LINE(  
    del_rows('EMPLOYEE_NAMES') || ' rows deleted.');//  
END;  
/
```

**Método 1, NO-Query
sin variables de host**

```
-- Insert a row into a table with two columns:  
CREATE PROCEDURE add_row(p_table_name VARCHAR2,  
    p_id NUMBER, p_name VARCHAR2) IS  
BEGIN  
    EXECUTE IMMEDIATE 'INSERT INTO ' || p_table_name ||  
        ' VALUES (:1, :2)' USING p_id, p_name;  
END;
```

Valores de entrada a una
instrucción SQL dinámica
con la cláusula **USING**

SQL dinámico con una consulta de una fila:

```
CREATE FUNCTION get_emp( p_emp_id NUMBER )
RETURN employees%ROWTYPE IS
    v_stmt VARCHAR2(200);
    v_emprec employees%ROWTYPE;
BEGIN
    v_stmt := 'SELECT * FROM employees ' ||
              'WHERE employee_id = :p_emp_id';
    EXECUTE IMMEDIATE v_stmt INTO v_emprec USING p_emp_id;
    RETURN v_emprec;
END;
/
DECLARE
    v_emprec employees%ROWTYPE := get_emp(100);
BEGIN
    DBMS_OUTPUT.PUT_LINE('Emp: ' || v_emprec.last_name);
END;
/
```

1

2

```
FUNCTION GET_EMP compiled
anonymous block completed
Emp: King
```

Método 3 con una sola fila consultada:
Consulta con un número conocido de elementos
y variables de host de entrada.

Ejecutar dinámicamente un bloque anónimo de PL/SQL

```
CREATE FUNCTION annual_sal( p_emp_id NUMBER)
RETURN NUMBER IS
  v_plsql varchar2(200) :=
    'DECLARE ||
      rec_emp employees%ROWTYPE; ||
      BEGIN ||
        rec_emp := get_emp(:empid); ||
        :res := rec_emp.salary * 12; ||
      END;';
  v_result NUMBER;
BEGIN
  EXECUTE IMMEDIATE v_plsql
    USING IN p_emp_id, OUT v_result;
  RETURN v_result;
END;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(annual_sal(100))
```

The diagram illustrates the execution flow of the PL/SQL code. It highlights several key components with red boxes:

- A red box surrounds the parameter `p_emp_id` in the function header.
- A red box surrounds the variable `:res` in the assignment statement.
- A red box surrounds the parameter `p_emp_id` in the `EXECUTE IMMEDIATE` statement.
- A red box surrounds the function call `annual_sal(100)` in the final `EXECUTE` statement.

Four green circles with numbers 1, 2, 1, and 2 are connected by arrows to these highlighted areas, indicating the flow of data or control:

- Circle 1 (top left) points to the `p_emp_id` parameter in the function header.
- Circle 2 (middle left) points to the `:res` variable in the assignment statement.
- Circle 1 (bottom center) points to the `p_emp_id` parameter in the `EXECUTE IMMEDIATE` statement.
- Circle 2 (bottom right) points to the `annual_sal(100)` function call in the final `EXECUTE` statement.

Tipos PL/SQL Bind

- En Oracle 11g, cuando PL/SQL invocaba SQL, no se podía enlazar un valor si su tipo de datos era BOOLEAN o una colección o un Record Type declarado en una especificación de paquete
- PL/SQL no pudo invocar dinámicamente un subprograma PL/SQL que tenía un parámetro formal cuyo tipo de datos eran los anteriores.
- A partir de Oracle Database 12c, estas restricciones se han eliminado



Subprograma con un parámetro BOOLEAN

```
CREATE OR REPLACE PROCEDURE p (x BOOLEAN) AUTHID
    DEFINER AS
BEGIN
    IF x THEN
        DBMS_OUTPUT.PUT_LINE('x is true');
    END IF;
END;
/
DECLARE
    dyn_stmt VARCHAR2(200);
    b BOOLEAN := TRUE;
BEGIN
    dyn_stmt := 'BEGIN p(:x); END;';
    EXECUTE IMMEDIATE dyn_stmt USING b;
END;
/
```

PROCEDURE P compiled
anonymous block completed
x is true

Utilizar Native Dynamic SQL para compilar código PL / SQL

Utilice cualquiera a de las siguientes sentencias ALTER para compilar código PL/SQL

```
ALTER PROCEDURE name COMPILE  
ALTER FUNCTION name COMPILE  
ALTER PACKAGE name COMPILE [ SPECIFICATION ]  
ALTER PACKAGE name COMPILE [ BODY ]
```

```
CREATE PROCEDURE compile_plsql(p_name VARCHAR2,  
p_plsql_type VARCHAR2, p_options VARCHAR2 := NULL) IS  
v_stmt varchar2(200) := 'ALTER ' || p_plsql_type ||  
' ' || p_name || ' COMPILE';  
BEGIN  
IF p_options IS NOT NULL THEN  
v_stmt := v_stmt || ' ' || p_options;  
END IF;  
EXECUTE IMMEDIATE v_stmt;  
END;  
/
```

El procedimiento **compile_plsql** se puede utilizar para compilar código PL/SQL

```
EXEC compile_plsql ('get_emp', 'function')
```

Agenda

- Usando Native Dynamic SQL (NDS)
- Uso del paquete DBMS_SQL

Usando el paquete DBMS_SQL

- El paquete `DBMS_SQL` se utiliza para escribir SQL dinámico en procedimientos almacenados y para analizar las instrucciones DDL.
- Debe utilizar el paquete `DBMS_SQL` para ejecutar una instrucción **SQL dinámica que tiene un número desconocido de variables** de entrada o salida (Método 4)
 - En la mayoría de los casos, NDS es más fácil de usar y funciona mejor que `DBMS_SQL`.
- Por ejemplo, debe utilizar el paquete `DBMS_SQL` en las siguientes situaciones:
 - No conoce la lista `SELECT` en tiempo de compilación.
 - No sabe cuántas columnas volverá una sentencia `SELECT` o cuáles serán sus tipos de datos.

Uso de los subprogramas de paquete DBMS_SQL

- El paquete DBMS_SQL proporciona los siguientes subprogramas para ejecutar SQL dinámico:
 - OPEN_CURSOR
 - Para abrir un nuevo cursor y devolver un número de ID de cursor
 - PARSE
 - Para analizar la instrucción SQL. Puede analizar cualquier instrucción DML o DDL
 - BIND_VARIABLE
 - Para enlazar un valor dado a una variable BIND
 - EXECUTE
 - Para ejecutar la instrucción SQL y devolver el número de filas procesadas
 - FETCH_ROWS
 - Para recuperar la siguiente fila de una consulta (uso en un bucle para varias filas)
 - CLOSE_CURSOR
 - Para cerrar el cursor especificado

Uso de los subprogramas de paquete DBMS_SQL

NOTAS

- El uso del paquete DBMS_SQL para ejecutar instrucciones DDL puede resultar en un interbloqueo, hay que tener cuidado en su uso.
- El parámetro **LANGUAGE_FLAG** del procedimiento PARSE determina cómo Oracle maneja la sentencia SQL. Se pueden indicar diferentes valores
 - **V6 (o 0)**, que especifica el comportamiento de la versión 6.
 - **NATIVE (o 1)** para este parámetro especifica el uso del comportamiento normal asociado con la base de datos a la que está conectado el programa
 - **V7 (o 2)**, se especifica el comportamiento de la versión 7 de la base de datos de Oracle.

Uso de los subprogramas de paquete DBMS_SQL

- Para procesar dinámicamente una instrucción DML, realice los pasos siguientes:
 1. Utilice **OPEN_CURSOR** para establecer un área en la memoria para procesar una instrucción SQL.
 2. Utilice **PARSE** para establecer la validez de la instrucción SQL.
 3. Utilice la función **EXECUTE** para ejecutar la instrucción SQL.
 1. Esta función devuelve el número de filas procesadas.
 4. Utilice **CLOSE_CURSOR** para cerrar el cursor.

Utilizar DBMS_SQL con una instrucción DML: Eliminación de filas

```
CREATE OR REPLACE FUNCTION delete_all_rows
(p_table_name    VARCHAR2) RETURN NUMBER IS
  v_cur_id        INTEGER;
  v_rows_del      NUMBER;
BEGIN
  v_cur_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(v_cur_id,
    'DELETE FROM '|| p_table_name, DBMS_SQL.NATIVE);
  v_rows_del := DBMS_SQL.EXECUTE (v_cur_id);
  DBMS_SQL.CLOSE_CURSOR(v_cur_id);
  RETURN v_rows_del;
END;
/
```

```
CREATE TABLE temp_emp AS SELECT * FROM employees;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Rows Deleted: ' ||
delete_all_rows('temp_emp'));
END;
/
```

Utilizar DBMS_SQL con una instrucción DML parametrizada

```
CREATE PROCEDURE insert_row (p_table_name VARCHAR2,  
    p_id VARCHAR2, p_name VARCHAR2, p_region NUMBER) IS  
    v_cur_id      INTEGER;  
    v_stmt         VARCHAR2(200);  
    v_rows_added  NUMBER;  
BEGIN  
    v_stmt := 'INSERT INTO '|| p_table_name ||  
              ' VALUES (:cid, :cname, :rid)';  
    v_cur_id := DBMS_SQL.OPEN_CURSOR;  
    DBMS_SQLPARSE(v_cur_id, v_stmt, DBMS_SQL.NATIVE);  
    DBMS_SQLBIND_VARIABLE v_cur_id, ':cid', p_id);  
    DBMS_SQLBIND_VARIABLE v_cur_id, ':cname', p_name);  
    DBMS_SQLBIND_VARIABLE v_cur_id, ':rid', p_region);  
    v_rows_added := DBMS_SQLEXECUTE(v_cur_id);  
    DBMS_SQLCLOSE_CURSOR(v_cur_id);  
    DBMS_OUTPUT.PUT_LINE(v_rows_added||' row added');  
END;  
/
```

El ejemplo demuestra el paso adicional necesario para asociar valores para variables bind

Uso de los subprogramas de paquete DBMS_SQL SELECT

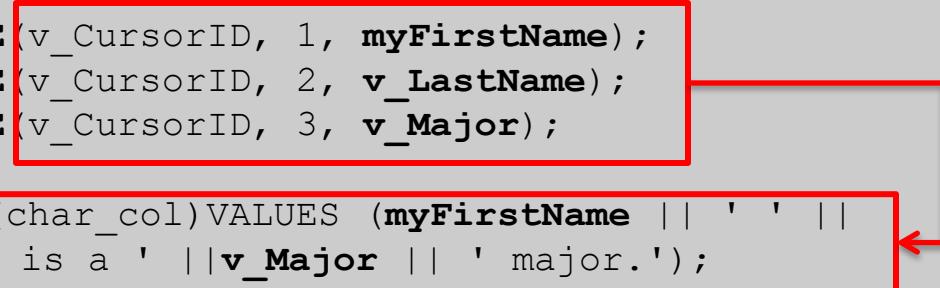
- Para procesar dinámicamente una instrucción SELECT, realice los siguientes pasos después de abrir y antes de cerrar el cursor:
 1. Ejecute `DBMS_SQL.DEFINE_COLUMN` para cada columna seleccionada.
 2. Ejecute `DBMS_SQL.BIND_VARIABLE` para cada variable bind de la consulta.
 3. Para cada fila, realice los pasos siguientes:
 - Ejecute `DBMS_SQL.FETCH_ROWS` para recuperar una fila y devolver el número de filas obtenidas. Detener procesamiento adicional cuando se devuelve un valor cero.
 - Ejecute `DBMS_SQL.COLUMN_VALUE` para recuperar cada valor de columna seleccionado en cada variable PL/SQL para su procesamiento.
- Aunque este proceso de codificación no es complejo, es más lento escribir y es propenso a error en comparación con el uso del enfoque SQL Nativo Dinámico.

Utilizar DBMS_SQL con una instrucción: SELECT

```
CREATE OR REPLACE PROCEDURE DynamicQuery (
 2      p_Major1 IN lecturer.major%TYPE DEFAULT NULL,
 3      p_Major2 IN lecturer.major%TYPE DEFAULT NULL) AS
 4
 5      v_CursorID    INTEGER;
 6      v_SelectStmt  VARCHAR2(500);
 7      myFirstName   lecturer.first_name%TYPE;
 8      v_LastName    lecturer.last_name%TYPE;
 9      v_Major       lecturer.major%TYPE;
10      v_Dummy       INTEGER;
11
12  BEGIN
13      v_CursorID := DBMS_SQL.OPEN_CURSOR;
14
15      v_SelectStmt := 'SELECT first_name, last_name, major
16                           FROM lecturer
17                           WHERE major IN (:m1, :m2)
18                           ORDER BY major, last_name';
19      DBMS_SQLPARSE(v_CursorID, v_SelectStmt, DBMS_SQL.V7);
20
21      DBMS_SQLBIND_VARIABLE(v_CursorID, ':m1', p_Major1);
22      DBMS_SQLBIND_VARIABLE(v_CursorID, ':m2', p_Major2);
23
:
```

Utilizar DBMS_SQL con una instrucción: SELECT

```
:  
24      DBMS_SQL.DEFINE_COLUMN(v_CursorID, 1, myFirstName, 20);  
25      DBMS_SQL.DEFINE_COLUMN(v_CursorID, 2, v_LastName, 20);  
26      DBMS_SQL.DEFINE_COLUMN(v_CursorID, 3, v_Major, 30);  
27  
28      v_Dummy := DBMS_SQL.EXECUTE(v_CursorID);  
29  
30      LOOP  
:  
36          DBMS_SQL.COLUMN_VALUE(v_CursorID, 1, myFirstName);  
37          DBMS_SQL.COLUMN_VALUE(v_CursorID, 2, v_LastName);  
38          DBMS_SQL.COLUMN_VALUE(v_CursorID, 3, v_Major);  
39  
40          INSERT INTO MyTable (char_col) VALUES (myFirstName || ' ' ||  
                                         v_LastName || ' is a ' || v_Major || ' major.');
```



Quiz

El texto completo de la instrucción SQL dinámica puede ser desconocido hasta el tiempo de ejecución; Por lo tanto, su sintaxis se comprueba en *tiempo de ejecución* en lugar de en *tiempo de compilación*.

- a. True
- b. False

Resumen

En esta lección, debes haber aprendido a:

- Describir el flujo de ejecución de sentencias SQL
- Generar y ejecutar instrucciones SQL de forma dinámica mediante Native Dynamic SQL (NDS)
- Identifique situaciones en las que debe utilizar el paquete DBMS_SQL en lugar de NDS para generar y ejecutar instrucciones SQL de forma dinámica.

Práctica 7

En esta lección, realiza las siguientes prácticas:

- Creación de un paquete que utiliza Native Dynamic SQL para crear o eliminar una tabla y para llenar, modificar y eliminar filas de una tabla
- Creación de un paquete que compila el código PL/SQL en su esquema

8

Consideraciones de diseño para el código PL/SQL



ORACLE®

Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Crear constantes y excepciones estándar
- Escribir y llamar a subprogramas locales
- Controlar los privilegios de tiempo de ejecución de un subprograma
- Realizar transacciones autónomas
- Asignar funciones a paquetes PL/SQL y subprogramas almacenados independientes
- Pasar los parámetros por referencia usando NOCOPY
- Utilice el `hint PARALLEL ENABLE` para optimizar
- Utilice la caché de resultado de la función PL/SQL de `cross-session`
- Utilice la cláusula `DETERMINISTIC` con funciones
- Utilice la cláusula `RETURNING` y el enlace masivo con DML

Agenda

- Estandarizar constantes y excepciones, usar subprogramas locales, controlar los privilegios de tiempo de ejecución de un subprograma y realizar transacciones autónomas
- Asignación de roles a los paquetes PL/SQL y subprogramas almacenados independientes
- Utilizando los hints NOCOPY y PARALLEL ENABLE, la caché de resultado de la función PL/SQL entre sesiones y la cláusula DETERMINISTIC
- Uso de la función de derechos de invocador del caché de resultados
- Utilizar la cláusula RETURNING y la vinculación a granel con DML

Estandarización de Constantes y Excepciones

- Cuando varios desarrolladores están escribiendo sus propios códigos en una aplicación, puede haber inconsistencias en el manejo de diferentes situaciones.
- A menos que se cumplan ciertas normas, la situación puede llegar a ser confusa o irreversible.
- Para superar estos, usted puede:
 - Implementar **estándares** de empresa que usan un enfoque consistente para manejar errores en toda la aplicación.
 - Crear manejadores de **excepciones** genéricos predefinidos que generan coherencia en la aplicación
 - Escribir y llamar a **programas que producen mensajes** de error consistentes

Estandarización de Constantes y Excepciones

Constantes y excepciones se implementan típicamente utilizando un paquete sin cuerpo (es decir, una especificación de paquete).

- La estandarización ayuda a:
 - Desarrollar programas que sean consistentes
 - Promover un mayor grado de reutilización de código
 - Mantenimiento del código de facilidad
 - Implementar los estándares de la empresa en todas las aplicaciones
- Comience con la estandarización de:
 - Nombres de excepción
 - Definiciones constantes

Estandarizar Excepciones Personalizadas

Cree un paquete de tratamiento de errores estandarizado que incluya todas las excepciones definidas por el programador y definidas para ser utilizadas en la aplicación.

```
CREATE OR REPLACE PACKAGE error_pkg IS
    e_fk_err          EXCEPTION;
    e_seq_nbr_err     EXCEPTION;
    PRAGMA EXCEPTION_INIT (e_fk_err, -2292);
    PRAGMA EXCEPTION_INIT (e_seq_nbr_err, -2277);
    ...
END error_pkg;
/
```

- En el ejemplo de la diapositiva, el paquete **error_pkg** es un paquete de excepciones estandarizado
 - Declara un conjunto de identificadores de excepción definidos por el programador mediante la directiva PRAGMA EXCEPTION_INIT para asociar nombres

Estandarizar Excepciones Personalizadas

Hacer la implementación anterior permite referirse a cualquiera de las excepciones de una manera estándar en sus aplicaciones, como en el ejemplo siguiente:

```
BEGIN  
    DELETE FROM departments  
    WHERE department_id = deptno;  
    ...  
EXCEPTION  
    WHEN error_pkg.e_fk_err THEN  
    ...  
    WHEN OTHERS THEN  
    ...  
END;  
/
```

Estandarización de las Excepciones de Usuario

- El tratamiento de excepciones de usuarios se puede implementar como:
 - Un subprograma independiente
 - Un subprograma agregado al paquete que define las excepciones.
- Considere la posibilidad de crear un paquete con:
 - Cada excepción de usuario que se va a utilizar en la aplicación
 - Estas excepciones deben de tener números de error -20000 a -20999.
 - Un programa para llamar `RAISE_APPLICATION_ERROR` basado en excepciones de paquete
 - Un programa para mostrar un error basado en los valores de `SQLCODE` y `SQLERRM`
 - Objetos adicionales, como tablas de registro de errores y programas para acceder a las tablas

Estandarización de las Excepciones de Usuario

- Una práctica común es utilizar parámetros que identifiquen el nombre del procedimiento y la ubicación en la que se ha producido el error.
 - Esto le permite realizar un seguimiento más rápido de los errores de ejecución.
- Una alternativa es utilizar el procedimiento incorporado `RAISE_APPLICATION_ERROR` para mantener una traza de excepciones que puede utilizarse para realizar un seguimiento de la secuencia de llamadas que conduce al error.

```
RAISE_APPLICATION_ERROR (-20001, 'Mi error', TRUE);
```

`RAISE_APPLICATION_ERROR` utilizando capacidades de rastreo de pila, con el tercer argumento establecido en `TRUE`

Estandarización de **constantes**

- Por definición, el valor de una variable cambia, mientras que el valor de una constante no se puede cambiar.
- Si tiene programas que usan variables locales cuyos valores no deberían cambiar y que no cambian, convierta las variables en constantes.
- Esto puede ayudar con el mantenimiento y la depuración de su código.
- Considere la posibilidad de crear un único paquete compartido con todas sus constantes en él.
 - Este procedimiento o paquete se puede cargar en el inicio del sistema para un mejor rendimiento.

Estandarización de constantes

El ejemplo en la diapositiva muestra el paquete `constant_pkg` que contiene algunas constantes.

```
CREATE OR REPLACE PACKAGE constant_pkg IS
    c_order_received CONSTANT VARCHAR(2) := 'OR';
    c_order_shipped   CONSTANT VARCHAR(2) := 'OS';
    c_min_sal         CONSTANT NUMBER(3)  := 900;
END constant_pkg;
```

Refiérase a cualquiera de las constantes del paquete en su aplicación, según sea necesario:

```
BEGIN
    UPDATE employees
        SET salary = salary + 200
        WHERE salary <= constant_pkg.c_min_sal;
END;
```

Subprogramas locales

- Los subprogramas locales son subprogramas que son creados dentro de un código PL/SQL y no va a ser utilizado fuera de éste.
- Reducen el tamaño de un módulo eliminando el código redundante.
 - Si un módulo necesita la misma rutina varias veces, pero sólo este módulo necesita la rutina, a continuación, defina como un subprograma local.
- Puede ser definido dentro de la sección declarativa de cualquier programa PL/SQL, procedimiento, función o bloque anónimo.

Subprogramas locales

- Los subprogramas locales tienen las siguientes **características**:
 - Son accesibles sólo al bloque en el que se definen.
 - Se compilan como parte de sus bloques adjuntos.
- Los **beneficios**:
 - Reducción de código repetitivo
 - Mejora de la legibilidad del código y facilidad de mantenimiento
 - Menos administración porque hay un programa para mantener en lugar de dos

Subprogramas locales

El ejemplo ilustra un cálculo del impuesto sobre la renta del salario de un empleado.

```
CREATE PROCEDURE employee_sal(p_id NUMBER) IS
    v_emp employees%ROWTYPE;
    FUNCTION tax(p_salary VARCHAR2) RETURN NUMBER IS
        BEGIN
            RETURN p_salary * 0.825;
        END tax;
    BEGIN
        SELECT * INTO v_emp
        FROM EMPLOYEES WHERE employee_id = p_id;
        DBMS_OUTPUT.PUT_LINE('Tax: ' || tax(v_emp.salary));
    END;
/
EXECUTE employee_sal(100)
```

```
PROCEDURE EMPLOYEE_SAL compiled
anonymous block completed
Tax: 19800
```

Derechos del Definidor versus Derechos del Invocador

- De forma predeterminada, todos los programas son ejecutados con los privilegios del usuario que **creó** el subprograma.
- Esta forma de programación se utilizan para controlar el acceso y los privilegios al ejecutar un procedimiento creado por el usuario o por otro usuario diferente.

Derechos del definidor:

- AUTHID DEFINER
- Los programas se ejecutan con los privilegios del usuario **creador**.
- El usuario no requiere privilegios en los objetos subyacentes a los que accede el procedimiento.
- El usuario solo necesita privilegio para ejecutar un procedimiento.

Derechos del invocador:

- AUTHID CURRENT_USER
- Los programas se ejecutan con los privilegios del usuario **llamante**.
- El usuario requiere privilegios en los objetos subyacentes a los que accede el procedimiento.

Especificación de los derechos del invocador: Establecer AUTHID en CURRENT_USER

- Puede configurar los derechos del invocador para diferentes construcciones de subprograma PL/SQL de la siguiente manera:

```
CREATE FUNCTION name RETURN type AUTHID CURRENT_USER IS...
CREATE PROCEDURE name                      AUTHID CURRENT_USER IS...
CREATE PACKAGE name                         AUTHID CURRENT_USER IS...
CREATE TYPE name                           AUTHID CURRENT_USER IS OBJECT...
```

- El valor predeterminado es AUTHID DEFINER, que especifica que el subprograma se ejecuta con los privilegios de su **propietario**
 - La mayoría de los paquetes PL/SQL suministrados por Oracle son de este tipo

Especificación de los derechos del invocador: CURRENT_USER y DEFINER

```
CREATE OR REPLACE PROCEDURE add_dept(
    p_id NUMBER, p_name VARCHAR2) AUTHID CURRENT_USER IS
BEGIN
    INSERT INTO departments
        VALUES (p_id, p_name, NULL, NULL);
END;
```

```
CREATE OR REPLACE PROCEDURE add_dept(
    p_id NUMBER, p_name VARCHAR2) AUTHID DEFINER IS
BEGIN
    INSERT INTO departments
        VALUES (p_id, p_name, NULL, NULL);
END;
```

Especificación de los derechos del invocador: Establecer AUTHID en CURRENT_USER

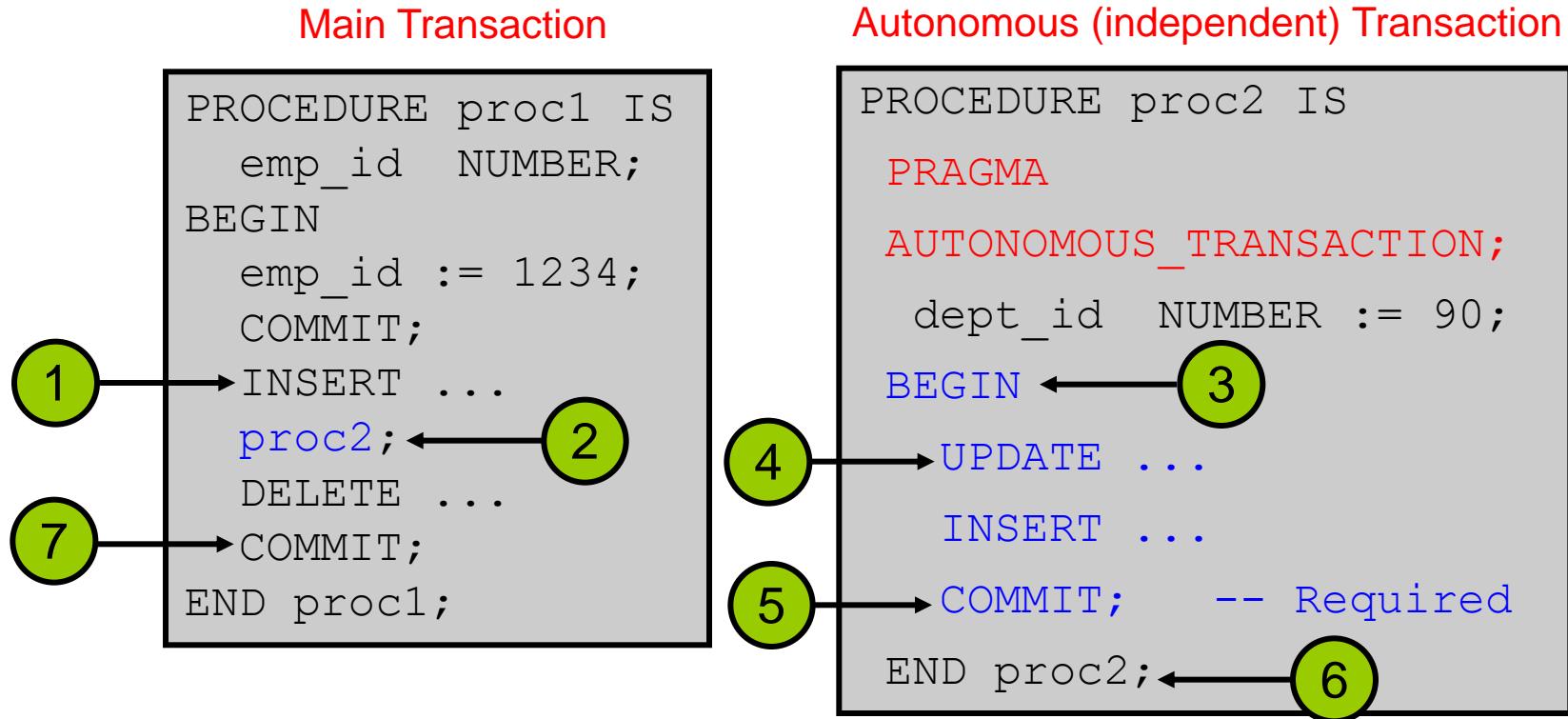
```
CREATE OR REPLACE PROCEDURE add_dept(
    p_id NUMBER, p_name VARCHAR2) AUTHID CURRENT_USER IS
BEGIN
    INSERT INTO departments
    VALUES (p_id, p_name, NULL, NULL);
END;
```

- Los nombres utilizados en `SELECT`, instrucciones de lenguaje de manipulación de datos (`DML`), `SQL dinámico` y `DBMS_SQL` se resuelven en el esquema del **invocador**.
- Todas las demás instrucciones, como llamadas a `paquetes`, `funciones` y `procedimientos`, se resuelven en el esquema del **definidor**.

Autonomous Transactions

- Transacción:
 - Es una serie de sentencias que realizan una unidad lógica de trabajo que se completa o falla como una unidad integrada.
- Hay veces en las cuales, se debe de realizar unas operaciones dentro de una transacción sin que comprometa la transacción principal.
 - Por ejemplo, grabar los intentos de INSERT incluso si al final no se realiza la inserción.
- Una transacción autónoma (**AT**) es una transacción independiente iniciada por otra transacción principal (**MT**)
- Se especifican con **PRAGMA AUTONOMOUS_TRANSACTION**
 - No puede utilizar PRAGMA para marcar todos los subprogramas de un paquete como autónomos. Sólo las rutinas individuales pueden ser marcadas como autónomas.

Autonomous Transactions



1. Comienza la transacción principal.
2. Se llama a un procedimiento **proc2** para iniciar la transacción autónoma.
3. Se suspende la operación principal y salta a la nueva transacción.
4. Comienza la transaccional autónoma.

5. La transacción autónoma se valida.
6. Se reanuda la operación principal.
7. La transacción principal termina.

Características de las transacciones autónomas

- Las transacciones autónomas presentan las siguientes características:
 - Son independientes de la transacción principal
 - Suspender la transacción principal hasta que se completen las transacciones autónomas
 - No hay límites distintos a los límites de recursos sobre cuántas transacciones autónomas pueden ser recursivamente llamadas.
 - No se deshace si la transacción principal se deshace.
 - Habilita los cambios para que sean visibles para otras transacciones y deben de ser **EXPLICITAMENTE** validadas/revertidas (Commit/Rollback)
 - No puede marcar un bloque PL/SQL anidado o anónimo como autónomo.

Uso de transacciones autónomas: Ejemplo

- Para definir transacciones autónomas, utilice `PRAGMA AUTONOMOUS_TRANSACTION`.
- Puede codificar `PRAGMA` en cualquier parte de la sección declarativa de una rutina. Sin embargo, para la legibilidad, se coloca mejor al principio de la sección de la declaración.
- Beneficios:
 - una transacción autónoma es totalmente independiente.
 - No comparte bloqueos, recursos ni dependencias de `commit` con la transacción principal
 - Las transacciones autónomas le ayudan a construir componentes de software modulares y reutilizables.

Uso de transacciones autónomas: Ejemplo

```
CREATE TABLE usage (card_id NUMBER, loc NUMBER)
/
CREATE TABLE txn (acc_id NUMBER, amount NUMBER)
/
CREATE OR REPLACE PROCEDURE log_usage (p_card_id NUMBER, p_loc NUMBER)
IS
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO usage
    VALUES (p_card_id, p_loc);
    COMMIT;
END log_usage;
/
CREATE OR REPLACE PROCEDURE bank_trans(p_cardnbr NUMBER,p_loc NUMBER) IS
BEGIN
    INSERT INTO txn VALUES (9001, 1000);
    log_usage (p_cardnbr, p_loc);
END bank_trans;
/
EXECUTE bank_trans(50, 2000)
```

En el ejemplo de la diapositiva, realiza un seguimiento del uso de la tarjeta bancaria, independientemente de si la transacción se ha realizado correctamente

Agenda

- Estandarizar constantes y excepciones, usar subprogramas locales, controlar los privilegios de tiempo de ejecución de un subprograma y realizar transacciones autónomas
- Asignación de roles a los paquetes PL/SQL y subprogramas almacenados independientes
- Utilizando los hints NOCOPY y PARALLEL ENABLE, la caché de resultado de la función PL/SQL entre sesiones y la cláusula DETERMINISTIC
- Uso de la función de derechos de invocador del caché de resultados
- Utilizar la cláusula RETURNING y la vinculación a granel con DML

Asignación de Roles a paquetes PL/SQL y subprogramas almacenados

- Antes de Oracle Database 12c, cuando queríamos crear un elemento de PL/SQL que todos los usuarios pudieran utilizar, incluso si sus privilegios eran más bajos que los suyos, tenía que ser una **unidad DR** (derechos de un definidor)
 - La **unidad DR** siempre funcionaba con todos **sus privilegios**, independientemente del usuario que lo invocara.
- A partir de Oracle Database 12c, puede:
 - Otorgar roles a paquetes individuales de PL/SQL y a subprogramas independientes.
 - En lugar de una **unidad DR**, puede crear una **unidad IR** (Derechos de invocador) y luego concederle roles.



Asignación de Roles a paquetes PL/SQL y subprogramas almacenados

- La **unidad IR** (invoker's rights) funciona con los privilegios del invocador sumado a los roles aplicados, pero sin privilegios adicionales que tenga.
- Se otorgan roles a una **unidad IR** para que los usuarios con privilegios inferiores a los suyos puedan ejecutar la unidad con sólo los privilegios necesarios para hacerlo.
- Mediante el comando **GRANT** de SQL, puede conceder roles a paquetes PL/SQL y a subprogramas almacenados independientes.
 - Los roles otorgados a una unidad PL/SQL no afectan a la compilación.

Agenda

- Estandarizar constantes y excepciones, usar subprogramas locales, controlar los privilegios de tiempo de ejecución de un subprograma y realizar transacciones autónomas
- Asignación de roles a los paquetes PL/SQL y subprogramas almacenados independientes
- Utilizando los Hints NOCOPY y PARALLEL ENABLE, la caché de resultado de la función PL/SQL entre sesiones y la cláusula DETERMINISTIC
- Uso de la función de derechos de invocador del caché de resultados
- Utilizar la cláusula RETURNING y la vinculación a granel con DML

Usando el Hint NOCOPY

- Permite al compilador PL/SQL pasar los parámetros OUT e IN OUT por referencia y no por valor
 - Por defecto IN → Por Referencia
 - OUT e IN OUT → Por VALOR
- La utilización del Hint NOCOPY en grandes estructuras de datos (como colecciones, registros etc) con parámetros OUT e IN OUT mejora el rendimiento y reduce la sobrecarga en la memoria.
- La siguiente diapositiva muestra un ejemplo de declarar un parámetro IN OUT con el Hint NOCOPY.
- Como NOCOPY es un HINT y no una directiva, el compilador **puede decidir** pasar parámetros NOCOPY a un subprograma por valor o por referencia.

Usando el Hint NOCOPY

```
DECLARE
  TYPE rec_emp_type IS TABLE OF employees%ROWTYPE;
  rec_emp rec_emp_type;

  PROCEDURE populate(p_tab IN OUT NOCOPY emptabtype) IS
    BEGIN
      . . .
    END;
BEGIN
  populate(rec_emp);
END;
/
```

Efectos del Hint NOCOPY

- El uso de NOCOPY afecta el manejo de excepciones de las siguientes maneras:
 - Como NOCOPY es un HINT y no una directiva, el compilador puede pasar parámetros NOCOPY a un subprograma por valor o por referencia.
 - Si un subprograma las asignaciones a los parámetros formales afectan inmediatamente a los parámetros reales.
 - Por lo tanto, si el subprograma sale con una excepción no controlada, no puede confiar en los valores de los parámetros reales NOCOPY.
- El protocolo RPC se utiliza para ejecutar procedimientos remotos. Éstos permiten pasar parámetros sólo por valor.
 - Si movemos un procedimiento local con parámetros NOCOPY a un sitio remoto, esos parámetros ya no se pasan por referencia.

¿Cuándo el compilador PL/SQL ignora el Hint NOCOPY?

- En los siguientes casos, el compilador PL/SQL ignora el Hint NOCOPY y utiliza el método de paso de parámetro by-value cuando:
 - El parámetro actual:
 - Es un elemento de matrices asociativas (index-by tables)
 - Está restringido (por escala o NOT NULL)
 - No se aplica a las cadenas de caracteres con restricciones de tamaño
 - Y el parámetro formal son registros, donde uno o ambos registros se declararon usando %ROWTYPE o %TYPE, y las restricciones en los campos correspondientes en los registros difieren
 - Requiere una conversión de tipo de datos implícita
 - El subprograma participa en una llamada de procedimiento externa o remota

Uso del Hint PARALLEL_ENABLE

- La palabra clave **PARALLEL_ENABLE** se puede utilizar en la sintaxis para declarar una **función**.
- Indica que la función se puede utilizar en una *consulta paralela* o en una *sentencia DML paralela*.
 - Oracle divide el trabajo de ejecución de una sentencia SQL en varios procesos
- Las funciones llamadas desde una instrucción SQL que se ejecuta en paralelo pueden tener una copia separada en cada uno de estos procesos.

Uso del Hint PARALLEL_ENABLE

- La palabra clave **PARALLEL_ENABLE** se coloca después del tipo de valor devuelto en la declaración de la función, como se muestra en el ejemplo de la diapositiva.

```
CREATE OR REPLACE FUNCTION f2 (p_p1 NUMBER)
  RETURN NUMBER PARALLEL_ENABLE IS
BEGIN
  RETURN p_p1 * 2;
END f2;
```

FUNCTION F2 compiled

Uso de la caché de resultados en funciones PL/SQL entre sesiones

- A partir de Oracle Database 11g, puede utilizar el mecanismo de **almacenamiento en caché** de resultados de funciones de PL/SQL
- Este mecanismo permite almacenar los resultados de las funciones PL/SQL en un área global compartida (SGA), que está compartida por todas las sesiones.
- Permite liberar al desarrollador de diseñar y crear funciones caches y sus políticas de administración de caché.
- El rendimiento y la escalabilidad se mejoran.

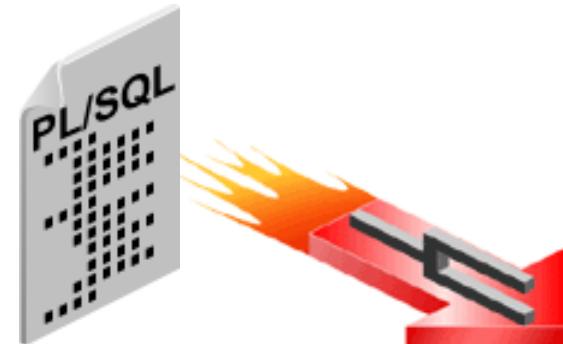
Uso de la caché de resultados en funciones PL/SQL entre sesiones

- Cada vez que se llama a una función PL/SQL en caché de resultados con diferentes valores de parámetro, esos parámetros y sus resultados se almacenan en la memoria caché.
- Posteriormente, cuando se llama a la misma función con los mismos valores de parámetro, el resultado se recupera de la memoria caché, en lugar de ser recalculado.
- Utilice la función de caché de resultados con funciones que se llaman con **frecuencia** y dependen de **información que nunca cambia o cambia con poca frecuencia**.

Habilitar el caché de resultados para una función

Para habilitar el almacenamiento en caché de resultados para una función PL/SQL haga:

- Incluya la cláusula `RESULT_CACHE` en los siguientes elementos:
 - La **declaración** de función
 - La **definición** de función
- Incluya una cláusula `RELIES_ON` opcional para especificar las tablas o vistas de las que dependen los resultados de la función.
- Si la ejecución de la función resulta en una **excepción no controlada**, el resultado de la excepción no se almacena en la memoria caché.



Declaración y definición de una función de caché de resultados: Ejemplo

- Utilice la cláusula RESULT_CACHE para habilitar la caché de resultados para una función.
- Ejemplo:

```
CREATE OR REPLACE FUNCTION get_hire_date (emp_id NUMBER)
  RETURN VARCHAR
RESULT_CACHE
AUTHID CURRENT_USER
IS
  date_hired DATE;
BEGIN
  SELECT hire_date INTO date_hired
  from employees
  WHERE employee_id = emp_id;
  RETURN TO_CHAR(date_hired);
END ;
/
SELECT get_hire_date(206) from DUAL;
```

La función devuelve la fecha de contratación del empleado pasado como parámetro

Uso de la cláusula DETERMINISTIC con funciones

- Puede utilizar la cláusula de función **DETERMINISTIC** para indicar que **la función devuelve el mismo valor de resultado siempre que se llama con los mismos valores** para sus argumentos.
- Cuando Oracle encuentra una función determinística en uno de estos contextos, **intenta utilizar resultados previamente calculados** cuando es posible en lugar de ejecutar la función de nuevo.
- No especifique **DETERMINISTIC** para una función cuyo resultado depende del estado de las variables de sesión o de los objetos de esquema

Uso de la cláusula DETERMINISTIC con funciones: Ejemplo

- Ejemplo:

```
CREATE OR REPLACE FUNCTION get_date_determ
RETURN DATE DETERMINISTIC IS
BEGIN
    RETURN df_demo.td;
END;
/

CREATE OR REPLACE FUNCTION slow_function (p_in IN NUMBER)
RETURN NUMBER
DETERMINISTIC
AS
BEGIN
    DBMS_LOCK.sleep(1);
    RETURN p_in;
END;
/
```

Agenda

- Estandarizar constantes y excepciones, usar subprogramas locales, controlar los privilegios de tiempo de ejecución de un subprograma y realizar transacciones autónomas
- Asignación de funciones a los paquetes PL/SQL y subprogramas almacenados independientes
- Utilizando las sugerencias NOCOPY y PARALLEL ENABLE, la caché de resultado de la función PL/SQL entre sesiones y la cláusula DETERMINISTIC
- Utilizar la cláusula RETURNING y la vinculación a granel con DML

Utilizar la cláusula RETURNING

- A menudo, las aplicaciones necesitan información sobre la fila afectada por una operación SQL.
- Las sentencias `INSERT`, `UPDATE` y `DELETE` pueden incluir una cláusula `RETURNING`, que devuelve los **valores de columna de la fila afectada**
 - Esto elimina la necesidad de realizar una `SELECT` después de un `INSERT` o `UPDATE`, o antes de un `DELETE` para conocer dicho resultado
 - Como resultado, se requieren menos tráfico de red, menos tiempo de CPU de servidor, menos cursoras y menos memoria de servidor.
- Estos valores pueden ser devueltos a:
 - Variables PL/SQL
 - Variables de host

Utilizar la cláusula RETURNING

- El ejemplo de la diapositiva muestra cómo **actualizar el salario** de un empleado y, al mismo tiempo, **recuperar el apellido del empleado y el nuevo salario** en una variable PL/SQL local..

```
CREATE OR REPLACE PROCEDURE update_salary(p_emp_id
    NUMBER) IS
    v_name      employees.last_name%TYPE;
    v_new_sal   employees.salary%TYPE;
BEGIN
    UPDATE employees
        SET salary = salary * 1.1
    WHERE employee_id = p_emp_id
    RETURNING last_name, salary INTO v_name, v_new_sal;
    DBMS_OUTPUT.PUT_LINE(v_name || ' new salary is ' ||
        v_new_sal);
END update_salary;
/
```

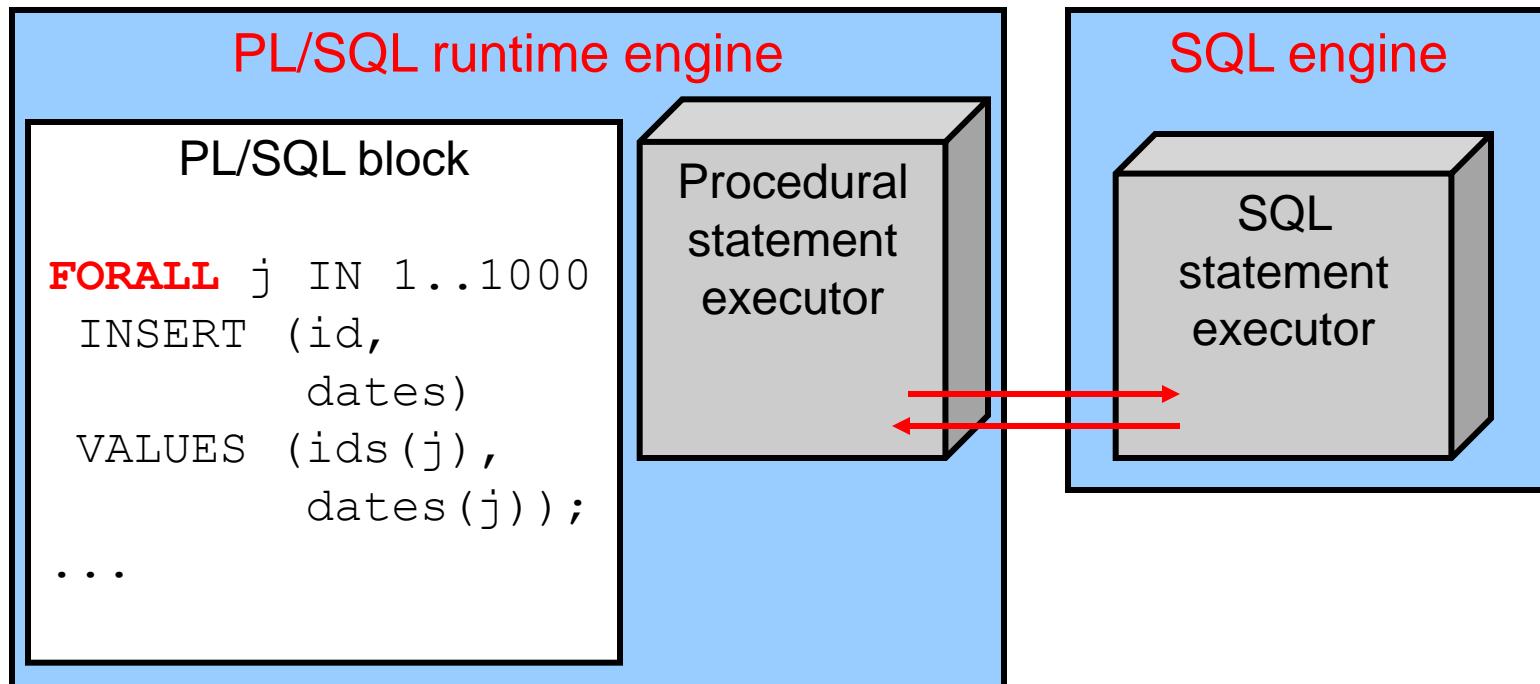
```
BEGIN
    update_salary(108);
END;
```

Usando Bulk Binding

- El servidor Oracle utiliza dos motores para ejecutar bloques y subprogramas PL/SQL:
 - El **motor de PL/SQL**, que ejecuta las instrucciones procedimentales (PLSQL) y reenvía las instrucciones SQL al motor de SQL
 - El **motor SQL**, que analiza y ejecuta la instrucción SQL y, en algunos casos, devuelve datos al motor PL/SQL
- Durante la ejecución, cada instrucción SQL provoca un **cambio de contexto** entre los dos motores, lo que resulta en una penalización de rendimiento.
 - Esto es típico de las aplicaciones que tienen una *instrucción SQL en un bucle* que utiliza valores de **colecciones**.
- El rendimiento se puede mejorar sustancialmente minimizando el número de comutadores de contexto mediante el uso de **bulk binding**

Usando Bulk Binding

- **Bulk Binding** hace que una colección entera se emita en una sola llamada al motor SQL y solo haya un cambio de contexto.
 - Cuantas más filas afectadas por una sentencia SQL, mayor será la ganancia de rendimiento con el enlace masivo.



Bulk Binding: Sintaxis y palabras clave

- La palabra clave FORALL hace que el motor PL/SQL trate las colecciones de **entrada** de forma masiva antes de enviarlas al motor SQL.

```
FORALL index IN lower_bound .. upper_bound  
[SAVE EXCEPTIONS]  
sql_statement;                                → Solo 1 Sentencia DML
```

- La palabra clave BULK COLLECT hace que el motor SQL enlace las colecciones de **salida** de forma masiva antes de devolverlas al motor PL/SQL.

```
... BULK COLLECT INTO  
collection_name [, collection_name] ...
```

Bulk Binding FORALL: Ejemplo

```
CREATE PROCEDURE raise_salary(p_percent NUMBER) IS
    TYPE numlist_type IS TABLE OF NUMBER
        INDEX BY BINARY_INTEGER;
    v_id  numlist_type; -- collection
BEGIN
    v_id(1):= 100; v_id(2):= 102; v_id(3):= 104; v_id(4) := 110;
    -- bulk-bind the PL/SQL table
    FORALL i IN v_id.FIRST .. v_id.LAST
        UPDATE employees
            SET salary = (1 + p_percent/100) * salary
            WHERE employee_id = v_id(i);
END;
/
```

```
ALTER TRIGGER update_job_history DISABLE;
```

```
EXECUTE raise_salary(10)
```

```
anonymous block completed
```

- El bloque PL/SQL aumenta el salario de los empleados con ID 100, 102, 104 o 110

Atributo de cursor adicional operaciones DML: %BULK_ROWCOUNT

```
CREATE TABLE num_table (n NUMBER);
DECLARE
    TYPE num_list_type IS TABLE OF NUMBER      INDEX BY
        BINARY_INTEGER;
    v_nums num_list_type;
BEGIN
    v_nums(1) := 1;
    v_nums(2) := 3;
    v_nums(3) := 5;
    v_nums(4) := 7;
    v_nums(5) := 11;
    FORALL i IN v_nums.FIRST .. v_nums.LAST
        INSERT INTO v num table (n) VALUES (v_nums(i));
    FOR i IN v_nums.FIRST .. v_nums.LAST
        LOOP
            dbms_output.put_line('Inserted ' ||
                SQL%BULK_ROWCOUNT(i) || ' row(s)' ||
                ' on iteration ' || i);
        END LOOP;
    END;
```


Usando BULK COLLECT INTO con Queries

- A partir de Oracle Database 10g, al utilizar una instrucción SELECT en PL/SQL, puede utilizar la sintaxis de recopilación masiva
 - La instrucción SELECT admite la sintaxis BULK COLLECT INTO.

```
CREATE PROCEDURE get_departments(p_loc NUMBER) IS
  TYPE dept_tab_type IS
    TABLE OF departments%ROWTYPE;
  v_depts dept_tab_type;
BEGIN
  SELECT * BULK COLLECT INTO v_depts
  FROM departments
  WHERE location_id = p_loc;
  FOR i IN 1 .. v_depts.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_depts(i).department_id
      || ' ' || v_depts(i).department_name);
  END LOOP;
END;
```

Lee todas las filas de la tabla departments de una región especificada

Usando BULK COLLECT INTO con Cursos

- La instrucción FETCH se ha mejorado para admitir la sintaxis BULK COLLECT INTO.

```
CREATE OR REPLACE PROCEDURE get_departments(p_loc NUMBER) IS
CURSOR cur_dept IS
    SELECT * FROM departments
    WHERE location_id = p_loc;
TYPE dept_tab_type IS TABLE OF cur_dept%ROWTYPE;
v_depts dept_tab_type;
BEGIN
    OPEN cur_dept;
    FETCH cur_dept BULK COLLECT INTO v_depts;
    CLOSE cur_dept;
    FOR i IN 1 .. v_depts.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE(v_depts(i).department_id
            || ' ' || v_depts(i).department_name);
    END LOOP;
END;
```

También puede agregar una cláusula **LIMIT** para controlar el número de filas obtenidas en cada operación

Usando BULK COLLECT INTO con la Cláusula RETURNING

```
CREATE OR REPLACE PROCEDURE raise_salary(p_rate NUMBER)
IS
    TYPE emplist_type IS TABLE OF NUMBER;
    TYPE numlist_type IS TABLE OF employees.salary%TYPE
        INDEX BY BINARY_INTEGER;
    v_emp_ids emplist_type :=
        emplist_type(100,101,102,104);
    v_new_sals numlist_type;
BEGIN
    FORALL i IN v_emp_ids.FIRST .. v_emp_ids.LAST
        UPDATE employees
            SET commission_pct = p_rate * salary
            WHERE employee_id = v_emp_ids(i)
            RETURNING salary BULK COLLECT INTO v_new_sals;
    FOR i IN 1 .. v_new_sals.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE(v_new_sals(i));
    END LOOP;
END;
```

Atributo de cursor adicional operaciones DML: SQL%**BULK_ROWCOUNT**

- Otro atributo de cursor agregado para dar soporte a operaciones masivas es **SQL%BULK_ROWCOUNT**
- El atributo **SQL%BULK_ROWCOUNT** es una estructura compuesta diseñada para su uso con la instrucción **FORALL**
- Este atributo actúa como una tabla de índice.
 - Su *i*-ésimo elemento almacena el número de filas procesadas por la *i*-ésima ejecución de una instrucción UPDATE o DELETE
- Podemos ver un ejemplo en la siguientes transparencia

Ejemplo : SQL%BULK_ROWCOUNT

```
CREATE TABLE num_table (n NUMBER);

DECLARE
    TYPE num_list_type IS TABLE OF NUMBER
        INDEX BY BINARY_INTEGER;
    v_nums num_list_type;
BEGIN
    v_nums(1) := 1;  v_nums(2) := 3;  v_nums(3) := 5;
    v_nums(4) := 7;  v_nums(5) := 11;
    FORALL i IN v_nums.FIRST .. v_nums.LAST
        INSERT INTO num_table (n) VALUES (v_nums(i));
    FOR i IN v_nums.FIRST .. v_nums.LAST
        LOOP
            dbms_output.put_line('Inserted ' ||
                SQL%BULK_ROWCOUNT(i) || ' row(s)' ||
                ' on iteration ' || i);
        END LOOP;
    END;
/
DROP TABLE num_table;
```

FORALL y ROLLBACK

- En una ejecución de la cláusula FORALL que provoque una excepción no controlada, se produce un rollback de las operaciones producidas.
- Sin embargo, sin una excepción es capturada y tratada, los cambios realizados, se pueden conservar, y producir un rollback to savepoint del registro que ha dado la excepción.
- Para poder conservar los cambios hasta ese momento, deberemos utilizar la orden COMMIT en el tratamiento de la excepción.
- EXCEPTION SAVE (tratado mas adelante)

Utilización de SAVE EXCEPTIONS

- `SAVE EXCEPTION` se utiliza para poder tratar las diferentes excepciones que se producen en el comando `FORALL`.
 - Sino introducimos esté código, si una fila falla durante el bucle `FORALL` la ejecución del bucle se termina.

```
FORALL index IN lower_bound..upper_bound
  SAVE EXCEPTIONS
    {insert_stmt | update_stmt | delete_stmt}
```

- Con `SAVE_EXCEPTIONS` permite que el bucle continúe procesando las demás filas y lanzar al final un *error ORA-24381* para su tratamiento
- Todas las excepciones producidas se guardan en el cursor `%BULK_EXCEPTIONS` en forma de registros. (*Solo disponible en la zona exceptions*)

Utilización de SAVE EXCEPTIONS

- El cursor SQL%BULK_EXCEPTIONS dispone de 2 campos ERROR_INDEX y ERROR_CODE

Field	Definition
<u>ERROR_INDEX</u>	Holds the iteration of the FORALL statement where the exception was raised
<u>ERROR_CODE</u>	Holds the corresponding Oracle error code

`SQL%BULK_EXCEPTIONS(i).ERROR_INDEX`

Iteración del bucle en la que se presentó el error (fila)

`SQL%BULK_EXCEPTIONS(i).ERROR_CODE`

Código de Oracle del Error

`SQL%BULK_EXCEPTIONS.COUNT`

Número de excepciones producidas, registros que contiene el cursor

`SQL%BULK_EXCEPTIONS.COUNT = 0`

→ Ninguna excepción

Usando Bulk Binds en Sparse Collections

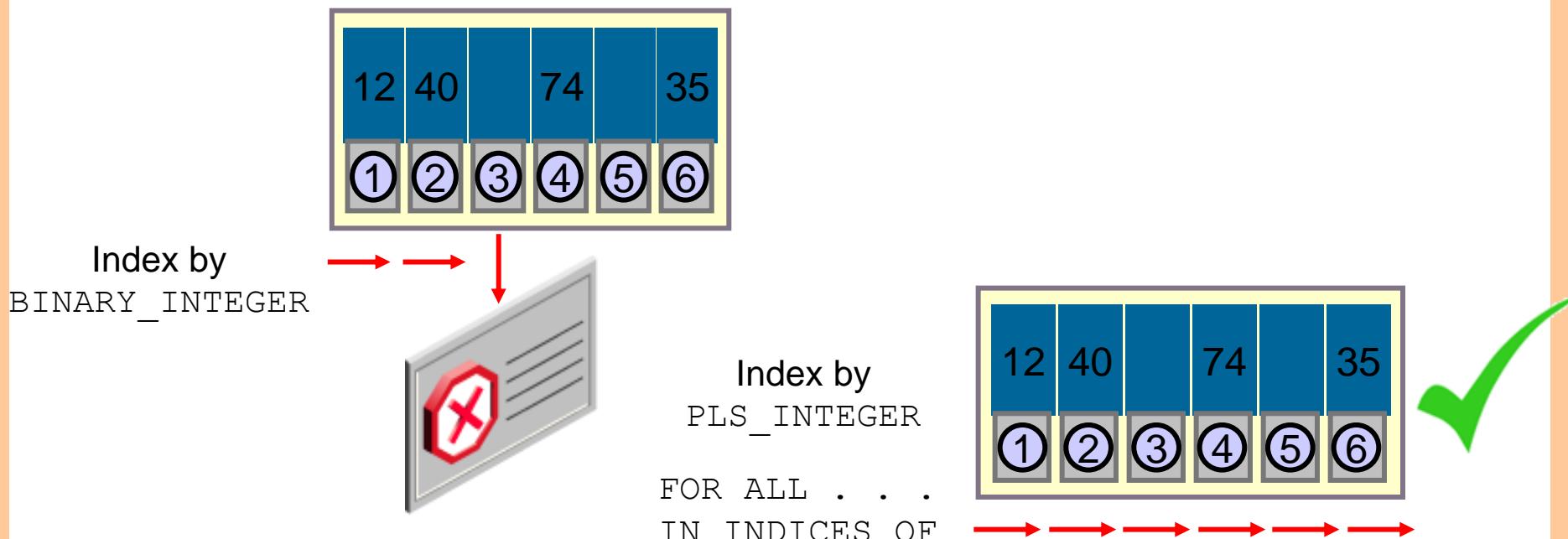
Colecciones Dispersas

- Una colección es **dispersa** si su distribución no es homogénea.
 - Por ejemplo, una colección dispersa tiene un elemento asignado al valor de índice 1 y otro al valor de índice 10, pero nada entre ellos.
 - Lo contrario de una colección dispersa es denso.
- En versiones anteriores a 10g, el motor PL/SQL intenta iterar sobre todos los elementos (existentes y no existentes).
 - Esto degradó sustancialmente el rendimiento de la operación DML si el porcentaje relativo de los elementos es alto.
- Podemos utilizar la palabra clave `INDICES`, para que el bucle se realiza de manera mas óptima con la instrucción `FORALL`.

Usando Bulk Binds en Sparse Collections

Colecciones Dispersas

- También ocurría en versiones anteriores, que si no se especificaba la palabra clave `SAVE EXCEPTIONS`, para capturar los errores, la instrucción se terminaba cuando se encontraba el primer elemento eliminado.



Usando Bulk Binds en Sparse Collections

Colecciones Dispersas

```
-- The INDICES OF syntax allows the bound arrays  
-- themselves to be sparse.  
  
FORALL index_name IN INDICES OF sparse_array_name  
    BETWEEN LOWER_BOUND AND UPPER_BOUND -- optional  
    SAVE EXCEPTIONS -- optional, but recommended  
        INSERT INTO table_name VALUES  
        sparse_array(index_name);  
    . . .
```

```
-- The VALUES OF syntax lets you indicate a subset  
-- of the binding arrays.  
  
FORALL index_name IN VALUES OF index_array_name  
    SAVE EXCEPTIONS -- optional, but recommended  
        INSERT INTO table_name VALUES  
        binding_array_name(index_name);  
    . . .
```

Using Bulk Binds in Sparse Collections

The typical application for this feature is an order entry and order processing system where:

- Users enter orders through the Web
- Orders are placed in a staging table before validation
- Data is later validated based on complex business rules (usually implemented programmatically using PL/SQL)
- Invalid orders are separated from valid ones
- Valid orders are inserted into a permanent table for processing

Usando Bulk Binds con Index Array

```
CREATE OR REPLACE PROCEDURE ins_emp2 AS
  TYPE emptab_type IS TABLE OF employees%ROWTYPE;
  v_emp emptab_type;
  TYPE values_of_tab_type IS TABLE OF PLS_INTEGER
    INDEX BY PLS_INTEGER;
  v_num      values_of_tab_type;
  . . .
BEGIN
  . . .
  FORALL k IN VALUES OF v_num
    INSERT INTO new_employees VALUES v_emp(k);
END;
```

Quiz

El hint NOCOPY permite al compilador PL/SQL pasar los parámetros OUT e IN OUT por referencia en lugar de por valor. Esto mejora el rendimiento reduciendo la sobrecarga al pasar los parámetros.

- a. True
- b. False

Resumen

En esta lección, debes haber aprendido a:

- Crear constantes y excepciones estándar
- Escribir y llamar a subprogramas locales
- Controlar los privilegios de tiempo de ejecución de un subprograma
- Realizar transacciones autónomas
- Asignar funciones a paquetes PL/SQL y subprogramas almacenados independientes
- Pasar los parámetros por referencia usando NOCOPY
- Utilice el `hint PARALLEL ENABLE` para optimizar
- Utilice la caché de resultado de la función PL/SQL de `cross-session`
- Utilice la cláusula `DETERMINISTIC` con funciones
- Utilice la cláusula `RETURNING` y el enlace masivo con DML

Práctica 8

En esta lección, realiza las siguientes prácticas:

- Creación de un paquete que utiliza operaciones bulk fetch
- Creación de un subprograma local para realizar una transacción autónoma para auditar una operación empresarial

9

Creación de Triggers



ORACLE®

Objetivos

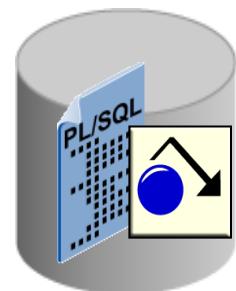
Después de completar esta lección, usted debería ser capaz de:

- Describir los triggers de la base de datos y sus usos
- Describir los diferentes tipos de triggers
- Crear triggers de base de datos
- Describir las reglas de activación de disparo de la base de datos
- Eliminar triggers de la base de datos
- Mostrar información de los triggers

¿Qué son los Triggers?

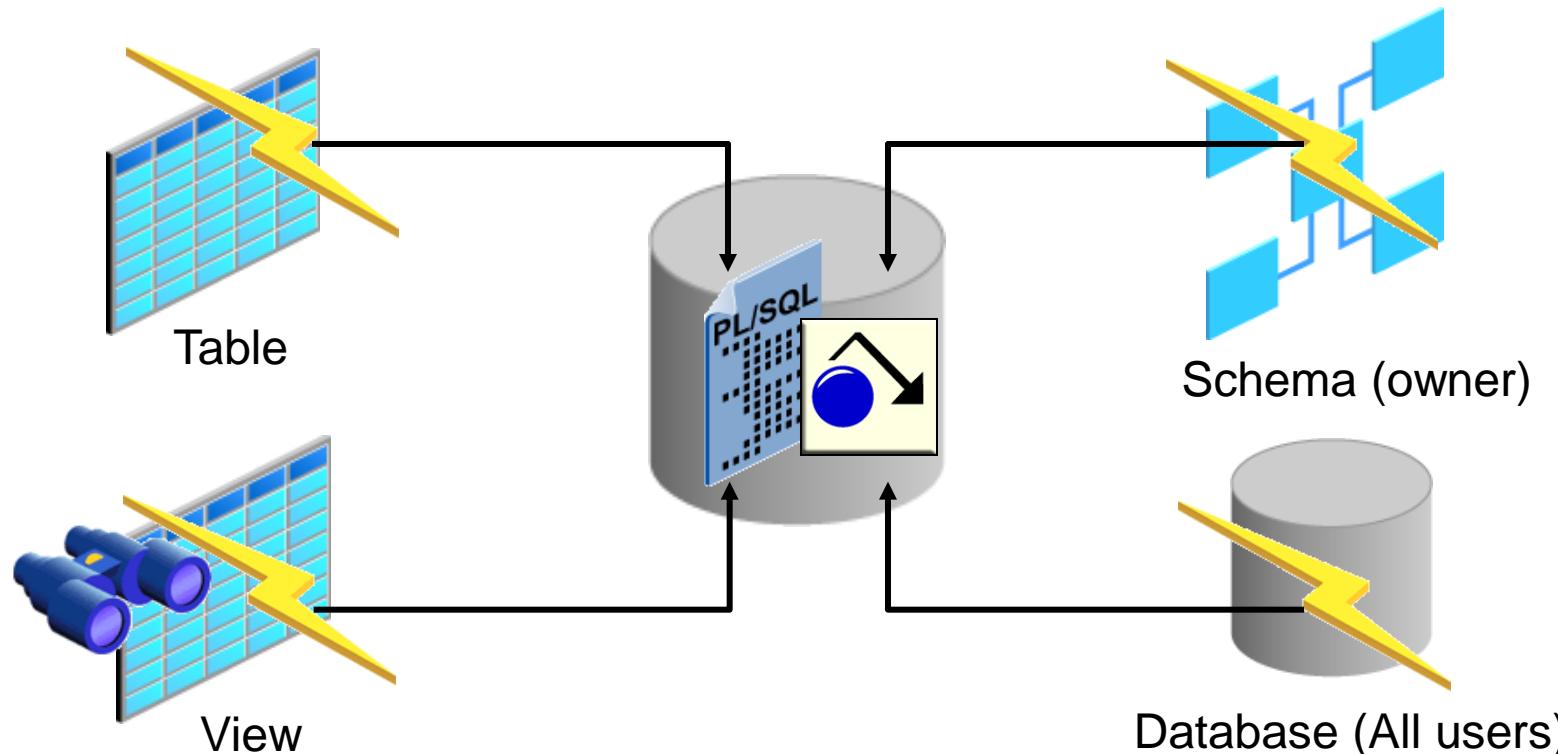
- Un trigger es un bloque PL/SQL que se almacena en la base de datos y se dispara (es ejecutado) en respuesta a un evento especificado.
- La base de datos Oracle ejecuta automáticamente un trigger cuando se producen las condiciones especificadas.
- Los triggers son similares a los procedimientos almacenados

Trigger	Subprograma
Es implícitamente ejecutado	Es explícitamente ejecutado
No permiten parámetros	Si permiten parámetros
No permiten órdenes DVL	Si permiten órdenes DVL



Definición de Triggers

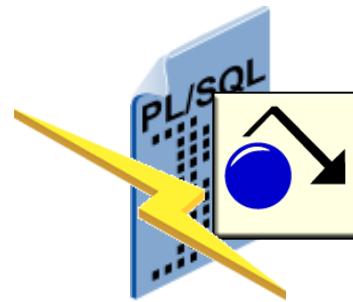
Un trigger se puede definir en la tabla, vista, esquema (propietario del esquema) o base de datos (todos los usuarios).



Tipos de Eventos en los Triggers

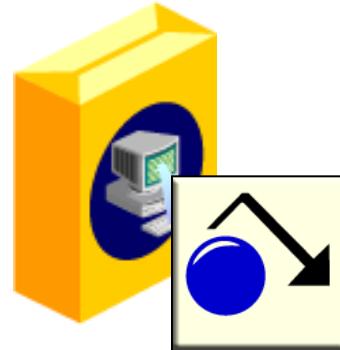
Puede escribir triggers que se disparan cuando se produce una de las siguientes operaciones en la base de datos:

- Una instrucción de manipulación de base de datos (**DML**) (DELETE, INSERT o UPDATE) en una tabla o vista
- Una sentencia de definición de base de datos (**DDL**) (CREATE, ALTER o DROP) en cualquier objeto del sistema
- Una operación de base de datos como SERVERERROR, LOGON, LOGOFF, STARTUP o SHUTDOWN

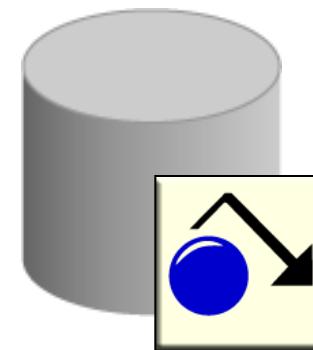


Triggers de Aplicación y de base de datos

- Trigger de base de datos (cubierto en este curso):
 - Se activa siempre que se produce un **DML**, un **DLL** o un **evento del sistema** en un esquema o una base de datos
- Trigger de Application:
 - Se activa cuando ocurre un evento dentro de una aplicación particular
 - Por ejemplo un evento en FORMS



Application Trigger



Database Trigger

Escenarios de aplicaciones para implementar Triggers

Podemos desarrollar triggers de base de datos para mejorar las características o funcionalidades a las proporcionadas por el servidor de Oracle.

- **Seguridad**
 - El servidor de Oracle permite el acceso de tablas a usuarios o roles.
 - Los triggers permiten el acceso a la tabla **según los valores de los datos**.
- **Auditoria**
 - El servidor Oracle realiza un seguimiento de las operaciones de datos en tablas.
 - Los triggers permiten auditor los valores para operaciones DML
- **Integridad de los datos**
 - El servidor de Oracle impone restricciones de integridad.
 - Los triggers implementan reglas complejas de integridad
- **Integridad referencial**
 - El servidor Oracle aplica las reglas de integridad referencial estándar.
 - Los triggers implementan funcionalidad no estándar
- **Registro de eventos**
 - El servidor de Oracle registra eventos de forma explícita.
 - Los triggers registran eventos de forma transparente.

Tipos de triggers disponibles

- Triggers DML simples
 - BEFORE
 - AFTER
 - INSTEAD OF
- Triggers Compuestos
- Triggers No-DML
 - Triggers de eventos DDL
 - Triggers de eventos de base de datos

Tipos de Eventos que producen Triggers DML

- Los tipos de Eventos que producen Triggers DML pueden ser operaciones:
 - INSERT
 - UPDATE [OF column]
 - DELETE
 - Cuando el evento de disparo es una instrucción [UPDATE](#), puede incluir una **lista de columnas** para identificar qué columnas deben cambiarse para disparar el disparador
- Todos trigger debe de definir un cuerpo con la acción se realiza.
 - Este cuerpo es un bloque de PL/SQL o una LLAMADA a un procedimiento.
 - El tamaño de un disparador no puede ser mayor de 32 KB.

Trigger Event Types and Body

- El trigger creado, puede contener una, dos o las tres operaciones DML.
... INSERT o UPDATE o DELETE
... INSERT o UPDATE de job_id. . .
- Si un trigger contiene mas de un evento, en su cuerpo deberemos tratarlo con las funciones:
INSERTING, UPDATING, DELETING
- Podemos crear tantos triggers como queramos y con eventos repetidos.

Creación de Triggers DML mediante la instrucción CREATE TRIGGER

```
CREATE [OR REPLACE] TRIGGER trigger_name  
timing -- when to fire the trigger  
event1 [OR event2 OR event3]  
ON object_name  
[REFERENCING OLD AS old | NEW AS new]  
FOR EACH ROW -- default is statement level trigger  
WHEN (condition) ]]  
DECLARE]  
BEGIN  
... trigger_body -- executable statements  
[EXCEPTION . . .]  
END [trigger_name];
```

timing = BEFORE | AFTER | INSTEAD OF

event = INSERT | DELETE | UPDATE | UPDATE OF column_list

Creación de Triggers DML mediante la instrucción CREATE TRIGGER

```
CREATE [OR REPLACE] TRIGGER trigger_name  
timing -- when to fire the trigger  
event1 [OR event2 OR event3]  
ON object name  
[REFERENCING OLD AS old | NEW AS new]  
FOR EACH ROW -- default is statement level trigger  
WHEN (condition) ]]  
DECLARE]  
BEGIN  
... trigger_body -- executable statements  
[EXCEPTION . . .]  
END [trigger_name];
```

REFERENCING = para elegir nombres de correlación para hacer referencia a los valores antiguos y nuevos de la fila actual (los valores predeterminados son **OLD** y **NEW**)

FOR EACH ROW Para que el trigger se ejecute una vez por cada fila

Especificación del evento de disparo (sincronización)

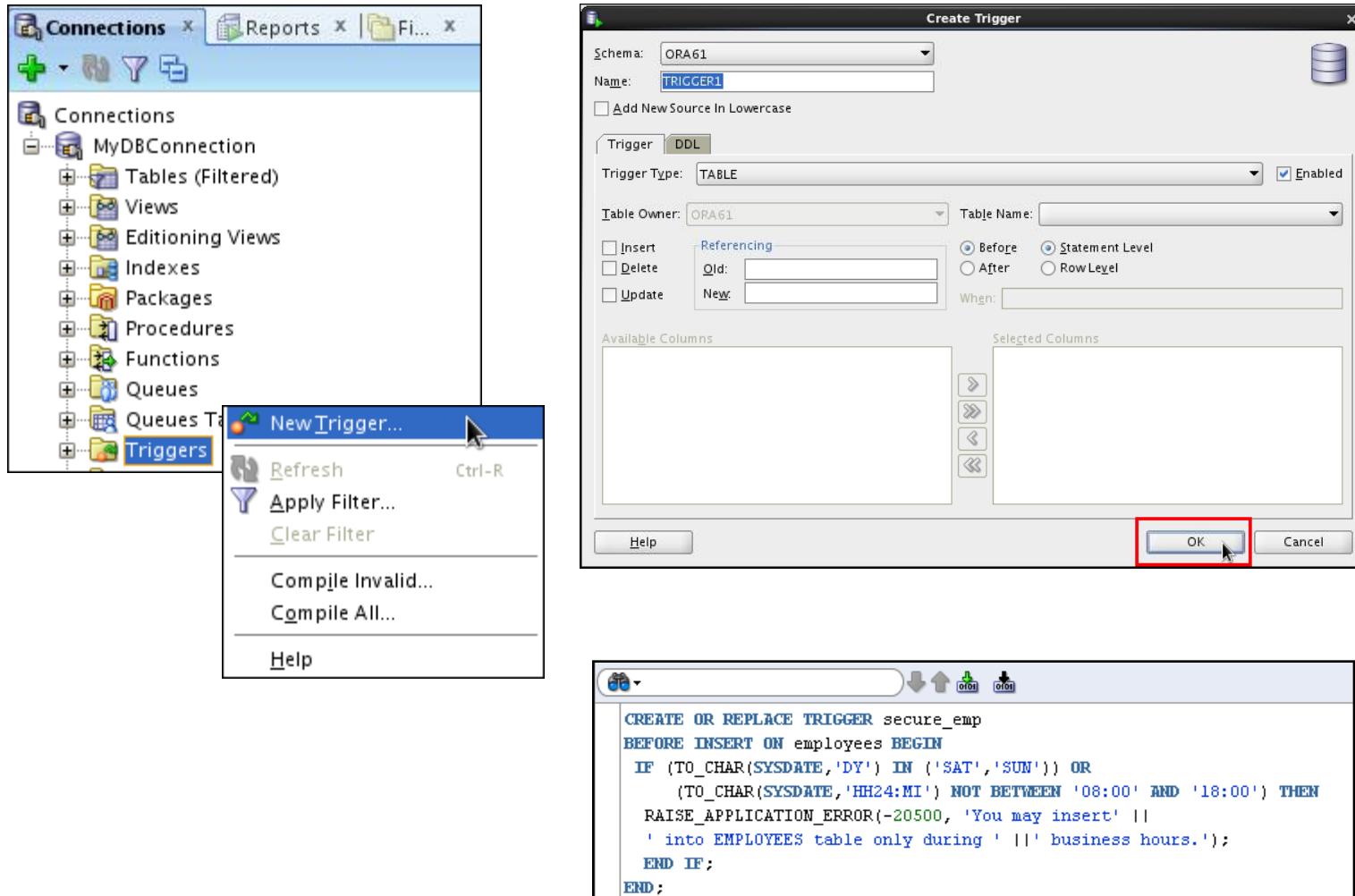
Puede especificar la temporización del trigger en cuanto a si se debe ejecutar la acción antes o después de la sentencia desencadenante:

- BEFORE:
 - Ejecutar el cuerpo del trigger antes del evento DML desencadenante en una tabla.
- AFTER:
 - Ejecutar el cuerpo del trigger después del evento DML desencadenante en una tabla.
- INSTEAD OF:
 - Ejecuta el cuerpo del trigger en lugar de la sentencia desencadenante.
 - Se utiliza para vistas que no son modificables de otra manera.

Triggers a nivel de instrucción versus Triggers de nivel de fila

Triggers a nivel de instrucción	Triggers de nivel de fila
Es el valor predeterminado cuando se crea un trigger	Use la cláusula <code>FOR EACH ROW</code> para crear un trigger.
Se dispara una vez para el evento DML	Se dispara una vez por cada fila afectada por el evento desencadenante
Se dispara una vez incluso si no hay filas afectadas	No se dispara si el evento de disparo no afecta a ninguna fila

Creación de triggers DML mediante SQL Developer



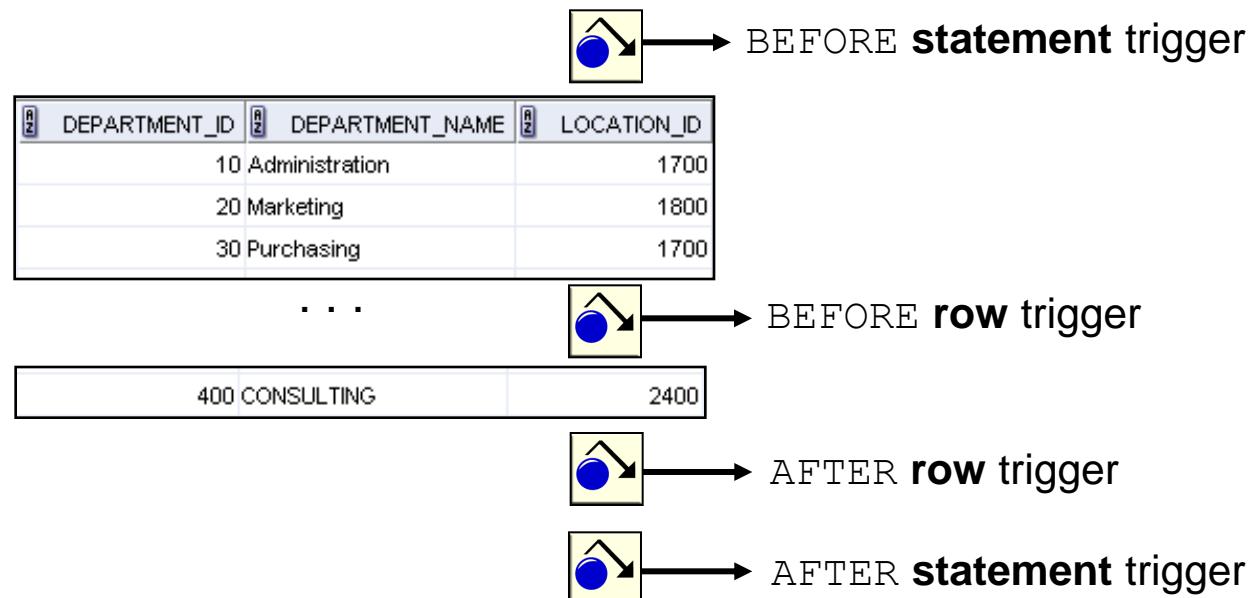
Secuencia trigger-evento: Manipulación de una fila

- Cree un trigger de sentencia o un trigger de fila basado en el requisito de que el trigger :
 - Debe iniciarse una vez para cada fila afectada por la sentencia desencadenante o
 - Solo una vez para la instrucción desencadenante, independientemente del número de filas afectadas.
- Cuando la sentencia DML desencadenante afecta a una sola fila, tanto el trigger de sentencia como el trigger de fila disparan **exactamente una vez**.

Secuencia trigger-evento: Manipulación de una fila

Ejemplo

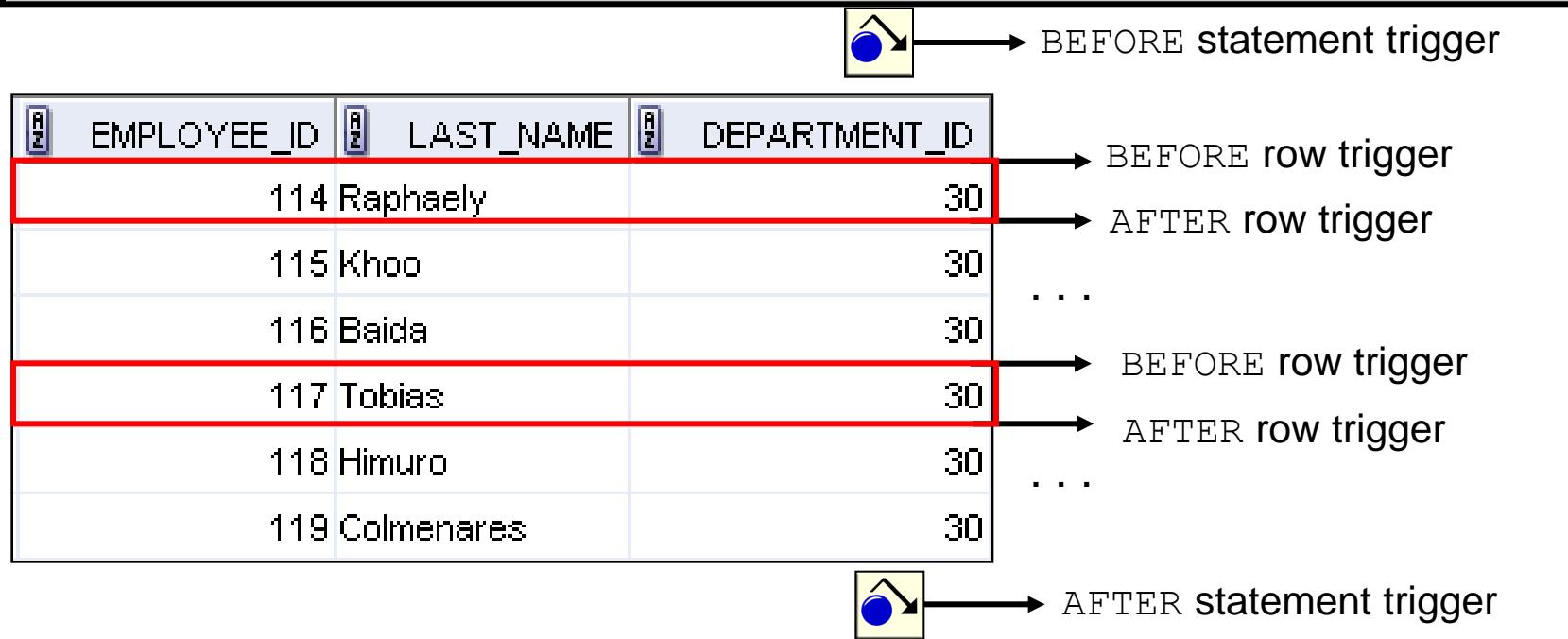
```
INSERT INTO departments
(department_id, department_name, location_id)
VALUES (400, 'CONSULTING', 2400);
```



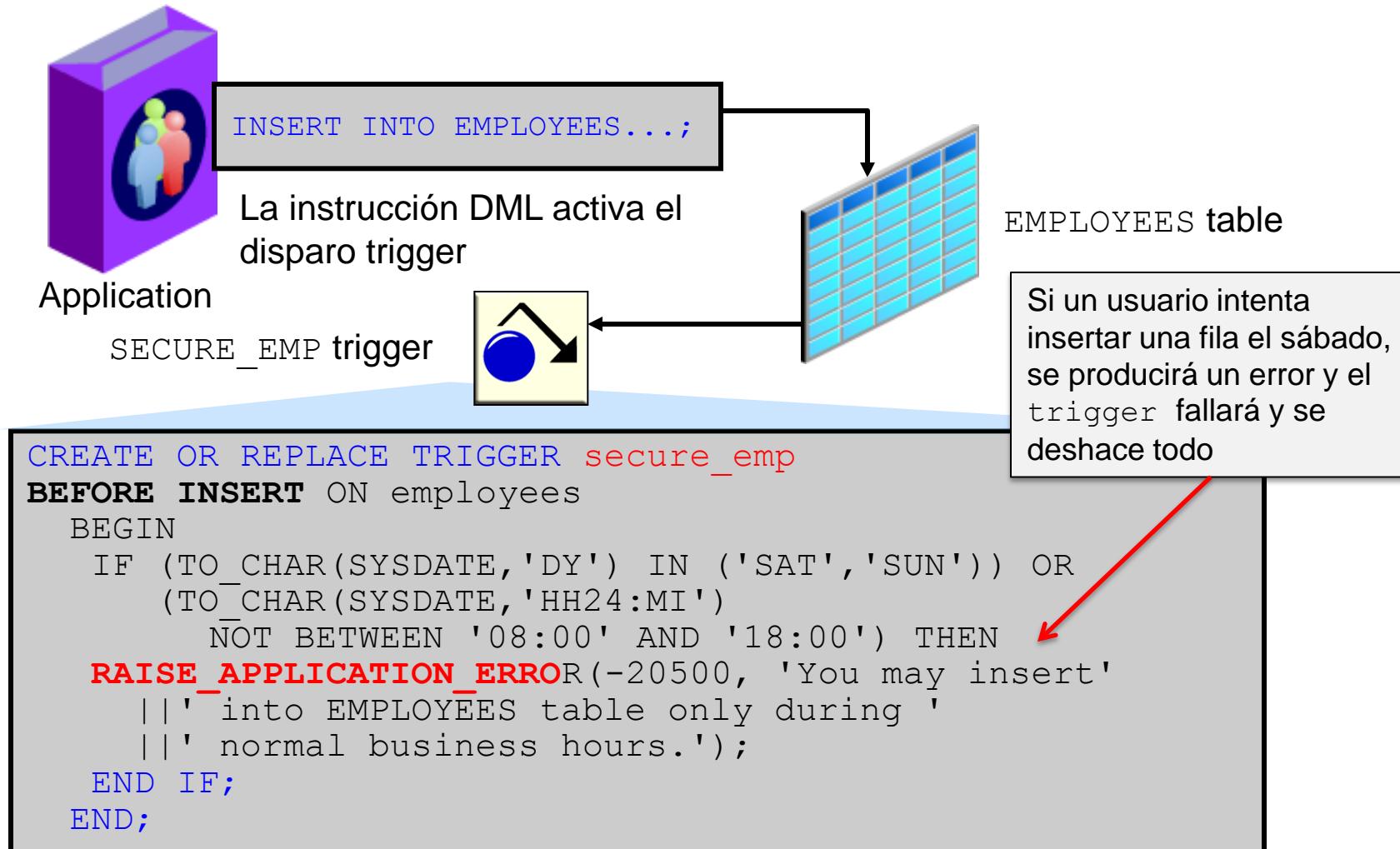
Secuencia trigger-evento: Manipulación Multifila

Cuando la sentencia DML afecta a muchas filas, el **trigger de instrucción** se activa exactamente **una vez** y el **trigger de fila** se activa **una vez por cada fila** afectada por la sentencia.

```
UPDATE employees  
SET salary = salary * 1.1  
WHERE department_id = 30;
```

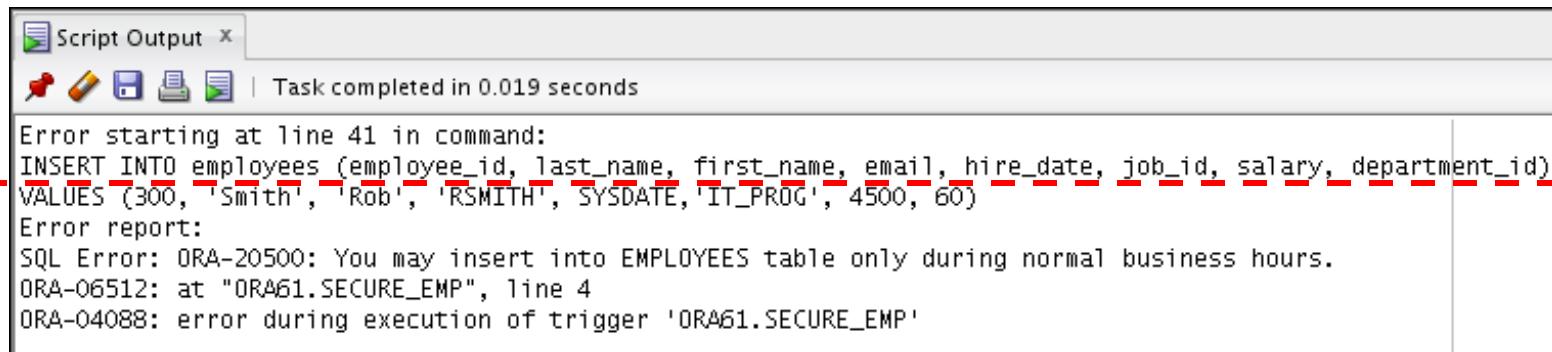


Ejemplo de creación de un trigger de instrucción DML: SECURE_EMP



Testeando el trigger SECURE_EMP

```
INSERT INTO employees (employee_id, last_name,
    first_name, email, hire_date, job_id, salary,
    department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,
    'IT_PROG', 4500, 60);
```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. The window title is 'Script Output'. It displays the following text:

Task completed in 0.019 seconds

Error starting at line 41 in command:

```
INSERT INTO employees (employee_id, last_name, first_name, email, hire_date, job_id, salary, department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE, 'IT_PROG', 4500, 60)
```

Error report:

```
SQL Error: ORA-20500: You may insert into EMPLOYEES table only during normal business hours.
ORA-06512: at "ORA61.SECURE_EMP", line 4
ORA-04088: error during execution of trigger 'ORA61.SECURE_EMP'
```

A red dashed box highlights the error message and the trigger name 'ORA61.SECURE_EMP'.

Uso de predicados condicionales

- Si definimos un trigger que pueda ser disparado en varios tipos de operaciones DML (por ejemplo, **ON INSERT** o **DELETE** o **UPDATE**), se puede usar en el cuerpo del trigger los predicados condicionales **INSERTING**, **DELETING** y **UPDATING**
- Mediante estos predicados condicionales conocemos la operación DML ejecutada.

```
IF INSERTING THEN  
    ...  
ELSIF UPDATING THEN  
    .....  
ELSE  
    .....  
END IF
```

Uso de predicados condicionales

```
CREATE OR REPLACE TRIGGER secure_emp BEFORE
INSERT OR UPDATE OR DELETE ON employees
BEGIN
    IF (TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN')) OR
        (TO_CHAR(SYSDATE, 'HH24')
            NOT BETWEEN '08' AND '18') THEN
        IF DELETING THEN RAISE_APPLICATION_ERROR(
            -20502, 'You may delete from EMPLOYEES table' ||
            'only during normal business hours.');
        ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(
            -20500, 'You may insert into EMPLOYEES table' ||
            'only during normal business hours.');
        ELSIF UPDATING ('SALARY') THEN
            RAISE_APPLICATION_ERROR(-20503, 'You may ' ||
                'update SALARY only normal during business hours.');
        ELSE RAISE_APPLICATION_ERROR(-20504, 'You may' ||
            ' update EMPLOYEES table only during' ||
            ' normal business hours.');
        END IF;
    END IF;
END;
```

Creación de un trigger de fila

```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
    IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
        AND :NEW.salary > 15000 THEN
        RAISE_APPLICATION_ERROR (-20202,
            'Employee cannot earn more than $15,000.');
    END IF;
END;
```

```
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell';
```

Puede crear un trigger de fila **BEFORE** para evitar que la operación tenga éxito si se infringe determinada condición.

```
Error starting at line 1 in command:
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell'
Error report:
SQL Error: ORA-20202: Employee cannot earn more than $15,000.
ORA-06512: at "ORA62.RESTRICT_SALARY", line 4
ORA-04088: error during execution of trigger 'ORA62.RESTRICT_SALARY'
```

Usando los cualificadores OLD y NEW

- Cuando se lanza un trigger de nivel de fila, el motor de ejecución de PL/SQL crea y rellena dos estructuras de datos:
 - **OLD**:
 - Almacena los valores originales del registro procesado por el trigger
 - **NEW**:
 - Contiene los nuevos valores
- **NEW** y **OLD** tienen la misma estructura que un registro declarado utilizando **%ROWTYPE** en la tabla a la que se usa el trigger.

Operación	Valor :OLD	Valor :NEW
INSERT	NULL	Valor insertado
UPDATE	Valor antes de la actualización	Valor tras actualización
DELETE	Valor antes de borrar	NULL

Usando OLD y NEW: Ejemplo

```
CREATE TABLE audit_emp (
    user_name      VARCHAR2(30),
    time_stamp     date,
    id             NUMBER(6),
    old_last_name VARCHAR2(25),
    new_last_name VARCHAR2(25),
    old_title      VARCHAR2(10),
    new_title      VARCHAR2(10),
    old_salary     NUMBER(8,2),
    new_salary     NUMBER(8,2) )
/
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO audit_emp(user_name, time_stamp, id,
        old_last_name, new_last_name, old_title,
        new_title, old_salary, new_salary)
    VALUES (USER, SYSDATE, :OLD.employee_id,
        :OLD.last_name, :NEW.last_name, :OLD.job_id,
        :NEW.job_id, :OLD.salary, :NEW.salary);
END;
```

El trigger registra los valores de varias columnas antes y después de los cambios de datos utilizando los calificadores **OLD y NEW** con el nombre de columna respectivo.

Usando OLD y NEW: Ejemplo (Continuacion)

```
INSERT INTO employees (employee_id, last_name, job_id,
salary, email, hire_date)
VALUES (999, 'Temp emp', 'SA_REP', 6000, 'TEMPEMP',
TRUNC(SYSDATE))
/
UPDATE employees
SET salary = 7000, last_name = 'Smith'
WHERE employee_id = 999
/
SELECT *
FROM audit_emp;
```

Query Result

All Rows Fetched: 2 in 0.005 seconds

	USER_NAME	TIME_STAMP	ID	OLD_LAST_NAME	NEW_LAST_NAME	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
1	ORA61	20-NOV-12	(null) (null)	Temp emp	(null)	SA_REP	SA_REP	(null)	6000
2	ORA61	20-NOV-12	999 Temp emp	Smith	Smith	SA_REP	SA_REP	6000	7000

Utilizar la cláusula WHEN para condicionar un trigger FOR EACH ROW

- Opcionalmente, puede incluir una restricción dentro de un trigger FOR EACH ROW especificando una expresión booleana en una cláusula WHEN
- Si incluye una cláusula WHEN en el trigger, la condición se evalúa para cada fila que afecte al trigger.
- Si la expresión se evalúa como
 - TRUE
 - El cuerpo del trigger se ejecuta con los valores del registro tratado
 - FALSO o NULL
 - El cuerpo del trigger no se ejecuta para esa fila
- Una cláusula WHEN no se puede incluir en la definición de un trigger de sentencia.

Utilizar la cláusula WHEN para condicionar un trigger FOR EACH ROW

```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
BEGIN
  IF INSERTING THEN
    :NEW.commission_pct := 0;
  ELSIF :OLD.commission_pct IS NULL THEN
    :NEW.commission_pct := 0;
  ELSE
    :NEW.commission_pct := :OLD.commission_pct+0.05;
  END IF;
END;
/
```

El calificador **NEW** no puede ser prefijado con dos puntos en la cláusula **WHEN** porque la cláusula **WHEN** está fuera de los bloques PL/SQL.

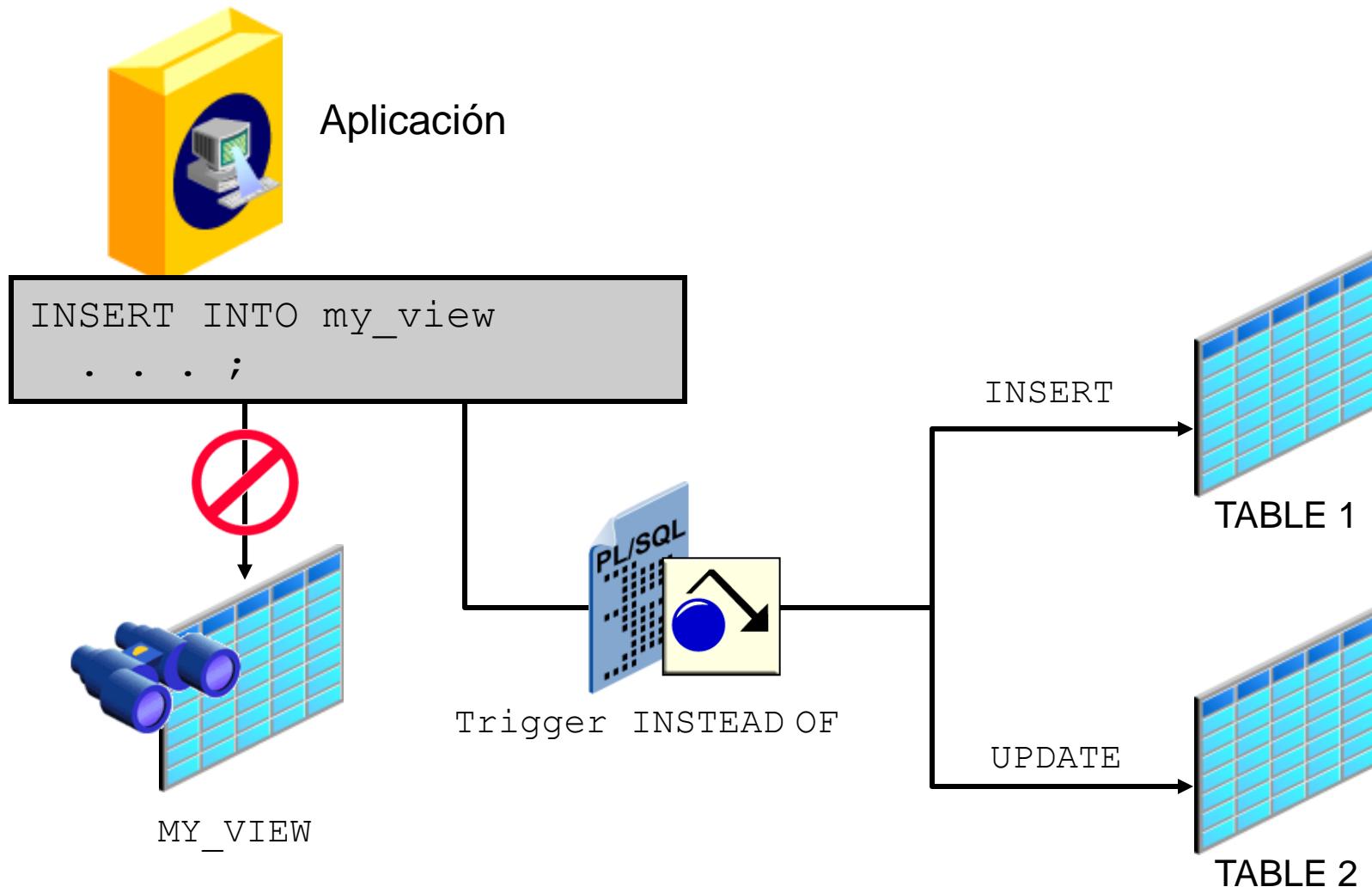
Resumen del modelo de ejecución de Trigger

1. Ejecutar todos los *triggers* BEFORE STATEMENT .
2. Loop para cada fila afectada por la instrucción SQL:
 - a. Ejecute todos los triggers BEFORE ROW para esa fila.
 - b. Ejecute la instrucción DML y realizar la comprobación de restricción de integridad para esa fila.
 - c. Ejecute todos los triggers AFTER ROW para esa fila.
3. Ejecutar todos los *triggers* AFTER STATEMENT.

Triggers INSTEAD OF

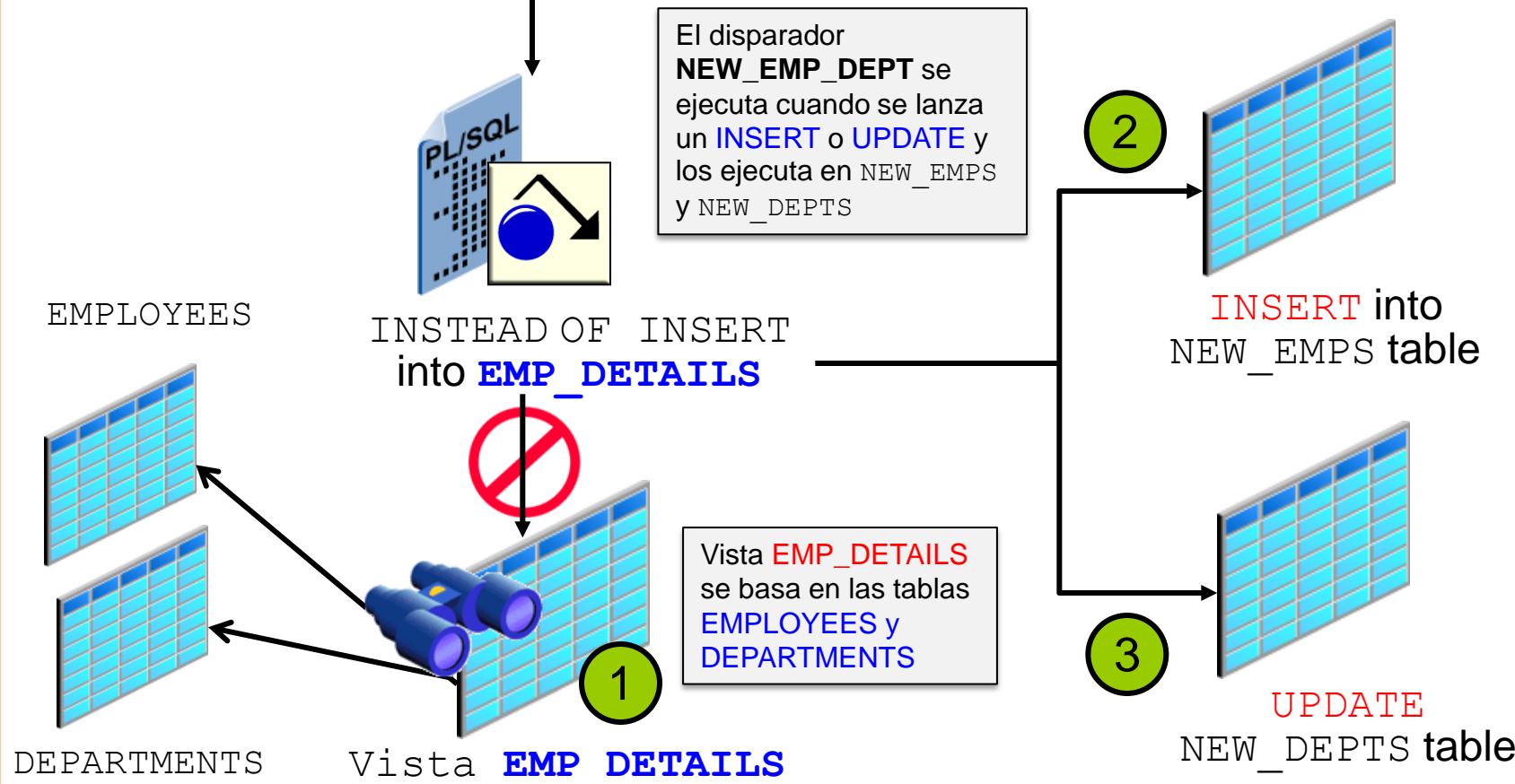
- Utilice los triggers **INSTEAD OF** para ejecutar una sentencia DML producida en una vista no actualizable.
 - Se llaman triggers **INSTEAD OF** porque, a diferencia de otros triggers , el servidor Oracle dispara el triggers en lugar de ejecutar la sentencia DML.
- Estos disparadores se utilizan para realizar operaciones **INSERT**, **UPDATE** y **DELETE** directamente en las tablas subyacentes.
 - El trigger **INSTEAD OF** funciona de forma invisible en segundo plano para realizar las acciones correctas.
- Una vista no puede ser modificada por sentencias DML normales si la consulta de vista contiene:
Operadores de conjunto, funciones de grupo, cláusulas como GROUP BY, CONNECT BY, START, el operador DISTINCT o uniones

Triggers INSTEAD OF



Usando un Trigger INSTEAD OF : Ejemplo

```
INSERT INTO emp_details  
VALUES (9001, 'ABBOTT', 3000, 10, 'Administration');
```



Creación de un Trigger INSTEAD OF para realizar DML en vistas complejas

```
CREATE TABLE new_emps AS  
SELECT employee_id, last_name, salary, department_id  
FROM employees;
```

```
CREATE TABLE new_depts AS  
SELECT d.department_id, d.department_name,  
       sum(e.salary) dept_sal  
  FROM employees e, departments d  
 WHERE e.department_id = d.department_id;
```

```
CREATE VIEW emp_details AS  
SELECT e.employee_id, e.last_name, e.salary,  
       e.department_id, d.department_name  
  FROM employees e, departments d  
 WHERE e.department_id = d.department_id  
 GROUP BY d.department_id, d.department_name;
```

```

CREATE OR REPLACE TRIGGER new_emp_dept
INSTEAD OF INSERT OR UPDATE OR DELETE ON emp_details
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO new_emps VALUES (:NEW.employee_id,
            :NEW.last_name,:NEW.salary, :NEW.department_id);
        UPDATE new_depts
            SET dept_sal = dept_sal + :NEW.salary
            WHERE department_id = :NEW.department_id;
    ELSIF DELETING THEN
        DELETE FROM new_emps WHERE employee_id = :OLD.employee_id;
        UPDATE new_depts
            SET dept_sal = dept_sal - :OLD.salary
            WHERE department_id = :OLD.department_id;
    ELSIF UPDATING ('salary') THEN
        UPDATE new_emps
            SET salary = :NEW.salary
            WHERE employee_id = :OLD.employee_id;
    :

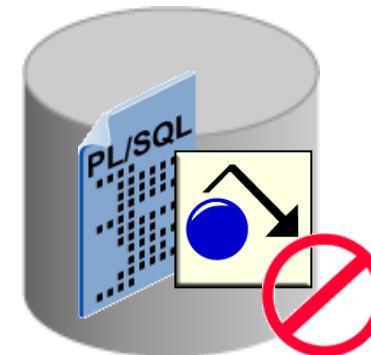
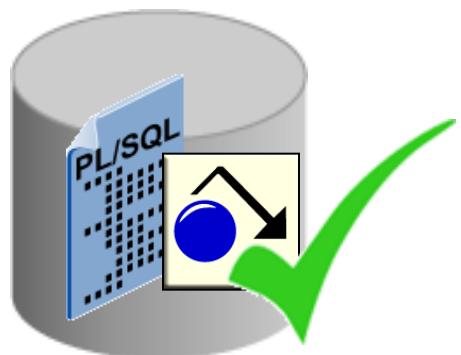
```

```
:  
    UPDATE new_depts  
        SET dept_sal = dept_sal + (:NEW.salary - :OLD.salary)  
        WHERE department_id = :OLD.department_id;  
ELSIF UPDATING ('department_id') THEN  
    UPDATE new_emps  
        SET department_id = :NEW.department_id  
        WHERE employee_id = :OLD.employee_id;  
    UPDATE new_depts  
        SET dept_sal = dept_sal - :OLD.salary  
        WHERE department_id = :OLD.department_id;  
    UPDATE new_depts  
        SET dept_sal = dept_sal + :NEW.salary  
        WHERE department_id = :NEW.department_id;  
END IF;  
END;  
/
```

El estado de un Trigger

Un trigger puede estar en cualquiera de estos **dos** modos distintos:

- **Enabled:**
 - El trigger **está** habilitado y se ejecuta si se produce la sentencia que lo provoca
- **Disabled:**
 - El trigger **no está** habilitado y no se ejecuta si se produce la sentencia que lo provoca



Creación de un trigger desactivado

- Antes de Oracle Database 11g, si creamos un trigger con un error de compilacion en el cuerpo PL/SQL, cuando se ejecute la sentencia DML que lo provoque, ésta fallará.
- Aparece el siguiente mensaje de error:

```
ORA-04098: trigger 'TRG' is invalid and failed re-validation
```

- Desde Oracle Database 11g, puede crear un trigger **deshabilitado** y habilitarlo sólo cuando sepa que se compilará correctamente.
- Por defecto cuando se crear un trigger su estado es ENABLED

Creación de un trigger desactivado

- Se puede desactivar temporalmente un trigger en las siguientes situaciones, cuando:
 - Un objeto referenciado no está disponible.
 - Debe realizar una carga de datos grande y desea que se realice rápidamente sin activar los triggers.
 - Se realiza **Reload** de datos

```
CREATE OR REPLACE TRIGGER mytrg
  BEFORE INSERT ON mytable FOR EACH ROW
  DISABLE
BEGIN
  :New.ID := my_seq.Nextval;
  . . .
END;
/
```

Administrar triggers mediante las instrucciones ALTER y DROP

```
-- Desactivar o Activar un trigger de base de datos:
```

```
ALTER TRIGGER trigger_name DISABLE | ENABLE;
```

```
-- Desactivar o activar todos los triggers de una tabla:
```

```
ALTER TABLE table_name DISABLE | ENABLE ALL TRIGGERS;
```

```
-- Recomilar un trigger de una tabla:
```

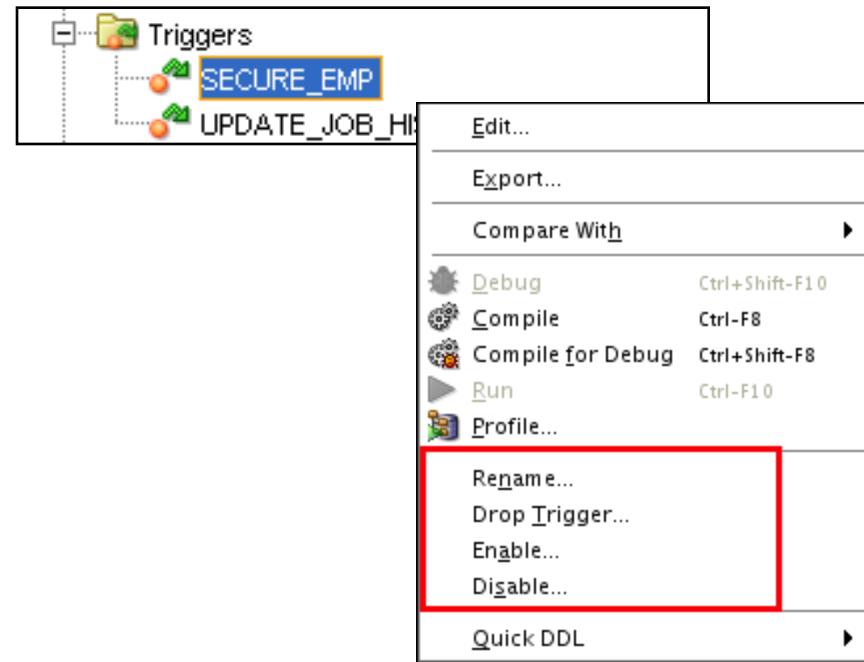
```
ALTER TRIGGER trigger_name COMPILE;
```

```
-- Eliminar un trigger de una base de datos:
```

```
DROP TRIGGER trigger_name;
```

Gestión de triggers mediante SQL Developer

- Puede utilizar el nodo triggers en el árbol de navegación.
- Haga clic con el botón derecho del ratón en un triggers y, a continuación, seleccione una de las siguientes opciones:
 - Edit
 - Compile
 - Compile for Debug
 - Rename
 - Drop Trigger
 - Enable
 - Disable



Testeando Triggers

- El código de prueba puede ser un proceso que consume mucho tiempo.
- Asegúrese de que el trigger funcione correctamente probando un número de casos por separado:
 - Primero pruebe los escenarios de éxito más comunes.
 - Pruebe las condiciones de fallo más comunes para ver que se administran correctamente.
 - Utilice el paquete DBMS_OUTPUT para depurar los disparadores.
 - Pruebe cada caso indicado de la cláusula WHEN.
 - Pruebe el efecto del disparador en otros disparadores.

Visualización de la información de Trigger

Puede ver la siguiente información del trigger en:

Data Dictionary View	Description
USER_OBJECTS	Muestra información del objeto
USER/ALL/DBA_TRIGGERS	Muestra la información del trigger
USER_ERRORS	Muestra errores de sintaxis de PL/SQL para un trigger

Usando USER_TRIGGERS

```
DESCRIBE user_triggers
```

Name	Null	Type
TRIGGER_NAME		VARCHAR2(128)
TRIGGER_TYPE		VARCHAR2(16)
TRIGGERING_EVENT		VARCHAR2(246)
TABLE_OWNER		VARCHAR2(128)
BASE_OBJECT_TYPE		VARCHAR2(18)
TABLE_NAME		VARCHAR2(128)
COLUMN_NAME		VARCHAR2(4000)
REFERENCING_NAMES		VARCHAR2(422)
WHEN_CLAUSE		VARCHAR2(4000)
STATUS		VARCHAR2(8)
DESCRIPTION		VARCHAR2(4000)
ACTION_TYPE		VARCHAR2(11)
TRIGGER_BODY		LONG()
CROSSEDITION		VARCHAR2(7)
BEFORE_STATEMENT		VARCHAR2(3)
BEFORE_ROW		VARCHAR2(3)
AFTER_ROW		VARCHAR2(3)
AFTER_STATEMENT		VARCHAR2(3)
INSTEAD_OF_ROW		VARCHAR2(3)
FIRE_ONCE		VARCHAR2(3)
APPLY_SERVER_ONLY		VARCHAR2(3)

También puede examinar las vistas [ALL_TRIGGERES](#) y [DBA_TRIGGERES](#), cada una de las cuales contiene la columna adicional OWNER, para el propietario del objeto.

```
SELECT trigger_type, trigger_body  
FROM user_triggers  
WHERE trigger_name = 'SECURE_EMP';
```

Quiz

Un evento desencadenante puede ser uno o más de los siguientes:
(elegir todas las posibles)

- a. Una instrucción INSERT, UPDATE o DELETE en una tabla específica (o vista, en algunos casos)
- b. Una sentencia CREATE, ALTER o DROP en cualquier objeto de esquema
- c. Un arranque de base de datos o un cierre de instancia
- d. Un mensaje de error específico o cualquier mensaje de error
- e. Un inicio de sesión de usuario o cierre de sesión

Resumen

En esta lección, debes haber aprendido a:

- Describir los triggers de la base de datos y sus usos
- Describir los diferentes tipos de triggers
- Crear triggers de base de datos
- Describir las reglas de activación de disparo de la base de datos
- Eliminar triggers de la base de datos
- Mostrar información de los triggers

Práctica 9

En esta lección, realiza las siguientes prácticas:

- Creación de triggers de fila
- Creación de un triggers de sentencia
- Llamando a procedimientos desde un triggers

10

Creación de triggers compuestos, DDL y Event Database



ORACLE®

Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Describir los triggers compuestos
- Describir las tablas de mutantes
- Crear triggers de sentencias DDL
- Crear triggers de eventos del sistema
- Mostrar información sobre triggers

¿Qué es un Trigger compuesto?

- A partir de Oracle Database 11g, puede utilizar un trigger compuesto.
- Un trigger compuesto. es un solo trigger en una tabla que le permite especificar acciones para cada uno de los cuatro puntos de temporización del trigger :
 - Before
 - Before each row
 - After each row
 - After

Trabajar con trigger compuestos

- Dentro del cuerpo del trigger compuesto se puede programar cualquiera de los puntos de eventos que puede tener un trigger.
- Cuando un punto de evento finaliza, éste se destruye automáticamente, incluso cuando la instrucción de disparo provoca un error.
- Son utilizados para evitar errores de **tabla mutante**
- Un trigger compuesto tiene una sección de declaración y una sección para cada uno de sus puntos de temporización/evento.

Usando Trigger compuestos

Puede utilizar desencadenantes compuestos para:

- Programe un enfoque en el que deseé que las acciones que implemente para los diferentes puntos de sincronización compartan datos comunes
- Acumule filas destinadas a una segunda tabla para que pueda insertarlas periódicamente
- Evite el error de la [tabla mutante](#) (ORA-04091) al permitir que las filas destinadas a una segunda tabla se acumulen y, a continuación, insertarlas en masa

Secciones de punto de sincronización de un trigger compuesto

Un disparador compuesto definido en una tabla tiene una o más de las siguientes **secciones** de punto de tiempo.

Las **secciones** de punto de sincronización deben aparecer en el orden mostrado en la tabla.

Punto de tiempo	Sección de Trigger Compuesto
Antes de ejecutar la sentencia	BEFORE
Después de ejecutar la sentencia	AFTER
Antes de cada fila que la sentencia	BEFORE EACH ROW
Antes de cada fila que la sentencia	AFTER EACH ROW

Si una sección de punto de tiempo está ausente, no ocurre nada en su punto de temporización.

Estructura de triggers compuestos para tablas

```
CREATE OR REPLACE TRIGGER schema.trigger  
FOR dml_event_clause ON schema.table  
COMPOUND TRIGGER
```

Una **sección inicial** donde se declaran variables y subprogramas

-- Initial section
-- Declarations
-- Subprograms

Un trigger compuesto tiene **dos secciones principales**:

1

-- Optional section
BEFORE STATEMENT IS ...;

-- Optional section
AFTER STATEMENT IS ...;

-- Optional section
BEFORE EACH ROW IS ...;

-- Optional section
AFTER EACH ROW IS ...;

Sección opcional que define el código para cada posible punto de activación.

2

Estructura de triggers compuestos para Vistas

- Con vistas, la única sección permitida es una cláusula INSTEAD OF EACH ROW.

```
CREATE OR REPLACE TRIGGER
schema.trigger
FOR dml_event_clause ON schema.view
COMPOUND TRIGGER

-- Initial section
-- Declarations
-- Subprograms

-- Optional section (exclusive)
INSTEAD OF EACH ROW IS
...;
```

Restricciones de triggers compuestos

- Un trigger compuesto **debe ser un trigger DML** y definido en una tabla o una vista.
- El cuerpo de un trigger compuesto **debe ser un bloque**, escrito en PL/SQL.
- Un cuerpo de trigger compuesto **no puede tener un bloque de inicialización**;
- Una excepción que se produce en una sección debe tratarse en esa sección. No puede transferir el control a otra sección.
- **:OLD** and **:NEW** no pueden aparecer en las declaraciones, **BEFORE STATEMENT** ni **AFTER STATEMENT**.
- Sólo la sección **BEFORE EACH ROW** puede cambiar el valor de **:NEW**.

Restricciones de triggers en las tablas mutantes

- Una tabla mutante es:
 - Una tabla que está siendo modificada por una instrucción **UPDATE**, **DELETE**, o **INSERT** o
 - Una tabla que podría ser actualizada por los efectos de una restricción **DELETE CASCADE**
 - Se produce un error de tabla de modificación (ORA-4091) cuando un trigger de nivel de fila intenta cambiar o examinar una tabla que ya está sufriendo cambios a través de una sentencia DML.
 - La sesión que emitió la instrucción del trigger no puede consultar o modificar una tabla mutante.

Restricciones de triggers en las tablas mutantes

- Esta restricción (error de tabla mutante) impide que un trigger pueda ver y trabajar con un conjunto inconsistente de datos.
- Esta restricción se aplica a todos los disparadores que utilizan la cláusula FOR EACH ROW .
- Las vistas que se modifican en los disparadores `INSTEAD OF` no se consideran mutantes.

Tabla Mutante: Ejemplo

El trigger **CHECK_SALARY** intenta garantizar que los salarios de la tabla EMPLOYEES estén dentro del rango de sueldos establecido para el trabajo del empleado.

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE INSERT OR UPDATE OF salary, job_id
  ON employees
  FOR EACH ROW
  WHEN (NEW.job_id <> 'AD_PRES')
DECLARE
  v_minsalary employees.salary%TYPE;
  v_maxsalary employees.salary%TYPE;
BEGIN
  SELECT MIN(salary), MAX(salary)
    INTO v_minsalary, v_maxsalary
   FROM employees
  WHERE job_id = :NEW.job_id;
  IF :NEW.salary < v_minsalary OR :NEW.salary > v_maxsalary THEN
    RAISE_APPLICATION_ERROR(-20505,'Out of range');
  END IF;
END;
/
```

El código del trigger consulta la misma tabla que se está actualizando. Por lo tanto, se dice que la tabla EMPLOYEES es una tabla mutante.

Tabla Mutante: Ejemplo

```
UPDATE employees  
SET salary = 3400  
WHERE last_name = 'Stiles';
```

Script Output X

| Task completed in 0.016 seconds

Error starting at line 1 in command:

```
UPDATE employees  
SET salary = 3400  
WHERE last_name = 'Stiles'
```

Error report:

```
SQL Error: ORA-04091: table ORA61.EMPLOYEES is mutating, trigger/function may not see it  
ORA-06512: at "ORA61.CHECK_SALARY", line 5  
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY'  
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"  
*Cause: A trigger (or a user defined plsql function that is referenced in  
this statement) attempted to look at (or modify) a table that was  
in the middle of being modified by the statement which fired it.  
*Action: Rewrite the trigger (or function) so it does not read that table.
```

La tabla employees está cambiando, o en un estado de cambio; Por lo tanto, el disparador no puede leerla.

Tabla Mutante: Ejemplo Posibles Soluciones

- Las posibles soluciones a este problema de la tabla mutante son las siguientes:
 - Utilización de un trigger compuesto
 - Almacene los datos de resumen (los salarios mínimos y los salarios máximos) en otra tabla de resumen, que se mantiene actualizada con los triggers DML.
 - Almacene los datos de resumen en un paquete PL/SQL y acceda a los datos del paquete. Esto se puede hacer en un desencadenador de declaración [BEFORE](#).
 - Recupere estos datos y realice las acciones de verificación necesarias

Utilizar un trigger compuesto para resolver el error de tabla mutante

```
CREATE OR REPLACE TRIGGER check_salary
FOR INSERT OR UPDATE OF salary, job_id
ON employees
WHEN (NEW.job_id <> 'AD_PRES')
COMPOUND TRIGGER
```

```
TYPE salaries_t           IS TABLE OF employees.salary%TYPE;
min_salaries
max_salaries
```

```
TYPE department_ids_t     IS TABLE OF employees.department_id%TYPE;
department_ids
```

```
TYPE department_salaries_t IS TABLE OF employees.salary%TYPE
                             INDEX BY VARCHAR2(80);
department_min_salaries
department_max_salaries
```

```
-- ejemplo sigue en la siguiente diapositiva
```

Esto se logra almacenando los valores en colecciones PL/SQL en la sección "before statement" del trigger compuesto y luego realizando una inserción / actualización masiva

Definimos Variables

Utilizar un trigger compuesto para resolver el error de tabla mutante

```
...  
BEFORE STATEMENT IS  
BEGIN  
    SELECT MIN(salary), MAX(salary), NVL(department_id, -1)  
        BULK COLLECT INTO min_Salaries, max_salaries, department_ids  
    FROM employees  
    GROUP BY department_id;  
    FOR j IN 1..department_ids.COUNT() LOOP  
        department_min_salaries(department_ids(j)) := min_salaries(j);  
        department_max_salaries(department_ids(j)) := max_salaries(j);  
    END LOOP;  
END BEFORE STATEMENT;
```

Almacenamiento de Valores en Colecciones

```
AFTER EACH ROW IS
```

Si un salario es menor / mayor que el de su departamento → Error

```
BEGIN  
    IF :NEW.salary < department_min_salaries(:NEW.department_id)  
        OR :NEW.salary > department_max_salaries(:NEW.department_id) THEN  
        RAISE_APPLICATION_ERROR(-20505,'New Salary is out of acceptable  
                                range');  
    END IF;  
END AFTER EACH ROW;  
END check_salary;
```

Creación de triggers en las instrucciones DDL

- Se puede especificar uno o más tipos de instrucciones DDL que pueden lanzar un trigger.
- Estos trigger se pueden producir a nivel de DATABASE o SCHEMA.
- También puede especificar BEFORE y AFTER para la temporización del disparador.

```
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE | AFTER -- Timing
[ddl_event1 [OR ddl_event2 OR ...]]
ON {DATABASE | SCHEMA}
trigger_body
```

Creación de triggers en las instrucciones DDL

Ejemplos de eventos DDL	Se lanza cuando...
CREATE	Cualquier objeto de base de datos se crea utilizando el comando CREATE.
ALTER	Cualquier objeto de base de datos se altera mediante el comando ALTER.
DROP	Cualquier objeto de base de datos se elimina mediante el comando DROP.

- La sentencia DDL activa el trigger sólo si el objeto afectado es:
 - Tabla, Tablespace, Vista, Clúster
 - Función, Procedimiento, Paquete, Trigger
 - Sinónimo, Índice, Usuario
 - Secuencia

Creación de triggers de sistema

- Los triggers de sistema se pueden definir en el nivel de:
 - Base de Datos (DATABASE)
 - Es definido para todos los usuarios del Sistema
 - Logging on / Loggin off / Errores específicos Gestor
 - Esquema (SCHEMA)
 - Es definido sólo para los eventos de ese Esquema
 - CREATE, ALTER, o DROP

Creación de triggers con eventos del sistema

```
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE | AFTER -- timing
[database_event1 [OR database_event2 OR ...]]
ON {DATABASE | SCHEMA}
trigger_body
```

Evento de base de datos	Los Triggers se disparan cuando
AFTER SERVERERROR	Un error Oracle ocurre
AFTER LOGON	Un usuario inicia sesión en la base de datos
BEFORE LOGOFF	Un usuario cierra la sesión de la base de datos
AFTER STARTUP	Se abre la base de datos
BEFORE SHUTDOWN	La base de datos se cierra normalmente

Trigger LOGON y LOGOFF : Ejemplo

```
-- Create the log_trig_table shown in the notes page
-- first

CREATE OR REPLACE TRIGGER logon_trig
AFTER LOGON ON DATABASE
BEGIN
    INSERT INTO log_trig_table(user_id,log_date,action)
    VALUES (USER, SYSDATE, 'Logging on');
END;
/
```

```
CREATE OR REPLACE TRIGGER logoff_trig
BEFORE LOGOFF ON DATABASE
BEGIN
    INSERT INTO log_trig_table(user_id,log_date,action)
    VALUES (USER, SYSDATE, 'Logging off');
END;
/
```

Sentencia CALL en Triggers

- La sentencia `CALL` le permite llamar a un procedimiento almacenado, en lugar de codificar el cuerpo PL/SQL en el propio trigger.
- El procedimiento se puede implementar en **PL/SQL, C o Java**.

```
CREATE [OR REPLACE] TRIGGER trigger_name  
  timing  
  event1 [OR event2 OR event3]  
  ON table_name  
  [REFERENCING OLD AS old | NEW AS new]  
  [FOR EACH ROW]  
  [WHEN condition]  
CALL procedure name  
/
```

Sentencia CALL en Triggers

- La llamada puede hacer referencia a los atributos del disparador :NEW y :OLD como parámetros

```
CREATE OR REPLACE TRIGGER salary_check
BEFORE UPDATE OF salary, job_id ON employees
FOR EACH ROW
WHEN (NEW.job_id <> 'AD_PRES')
CALL sal_status(:NEW.job_id, :NEW.salary)
/
```

- Nota: No hay punto y coma al final de la instrucción CALL.

Sentencia CALL en Triggers . Ejemplo

```
CREATE OR REPLACE PROCEDURE log_execution IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('log_execution: Employee Inserted');
END;
/
CREATE OR REPLACE TRIGGER log_employee
BEFORE INSERT ON EMPLOYEES
CALL log_execution -- no semicolon needed
/
```

Beneficios de los triggers de base de datos

- Son alternativas a las características proporcionadas por el servidor Oracle
- Mejora de la seguridad de los datos:
 - Proporciona verificaciones de **seguridad**
 - Proporciona una **auditoría**
- Mejora de la integridad de los datos:
 - Impone **restricciones dinámicas de integridad** de datos
 - Impone **restricciones complejas de integridad** referencial
 - Asegura que las operaciones relacionadas se llevan a cabo conjuntamente implícitamente

Privilegios necesarios para administrar los triggers

Los siguientes privilegios del sistema son necesarios para administrar los disparadores:

- Los privilegios que le permiten crear, modificar y eliminar triggers en cualquier esquema:
 - GRANT CREATE TRIGGER TO ora61
 - GRANT ALTER ANY TRIGGER TO ora61
 - GRANT DROP ANY TRIGGER TO ora61
- TEI privilegio que le permite crear un disparador en la base de datos:
 - GRANT ADMINISTER DATABASE TRIGGER TO ora61
- El privilegio EXECUTE (si el disparador se refiere a cualquier objeto que no esté en su esquema)

Directrices para el diseño de disparadores

- Utilice los disparadores para:
 - Garantizar que se realizan acciones relacionadas para una operación específica
 - Operaciones globales centralizadas que deben activarse para la sentencia de activación, independientemente del usuario o la aplicación que emita la sentencia.
- NO utilice los disparadores para:
 - Duplicar o reemplazar la funcionalidad ya incorporada en la base de datos Oracle. (auditoría completa y no de valores)
- Puede crear procedimientos almacenados e invocarlos en un trigger , si el código PL/SQL es muy largo.
- El uso excesivo de trigger puede resultar en interdependencias complejas, que pueden ser difíciles de mantener en aplicaciones grandes.

Quiz

¿Cuál de las siguientes afirmaciones es verdadera para un trigger?

- a. Un trigger se define con una sentencia CREATE TRIGGER.
- b. El código fuente de un trigger está contenido en el diccionario de datos USER_TRIGGERS .
- c. Se invoca explícitamente un disparador.
- d. Un trigger es implícitamente invocado por DML.
- e. COMMIT , SAVEPOINT y ROLLBACK no están permitidos cuando se trabaja con un trigger .

Resumen

En esta lección, debes haber aprendido a:

- Describir los triggers compuestos
- Describir las tablas de mutantes
- Crear triggers de sentencias DDL
- Crear triggers de eventos del sistema
- Mostrar información sobre triggers

Práctica 10

En esta lección, realiza las siguientes prácticas:

- Creación de triggers avanzados para administrar reglas de integridad de datos
- Creación de triggers que causan una tabla mutante.
- Creación de triggers que usan paquetes para resolver el problema de la tabla mutante

11

Uso del compilador PL/SQL



ORACLE®

Objetivos

Después de completar esta lección, usted debería ser capaz de:

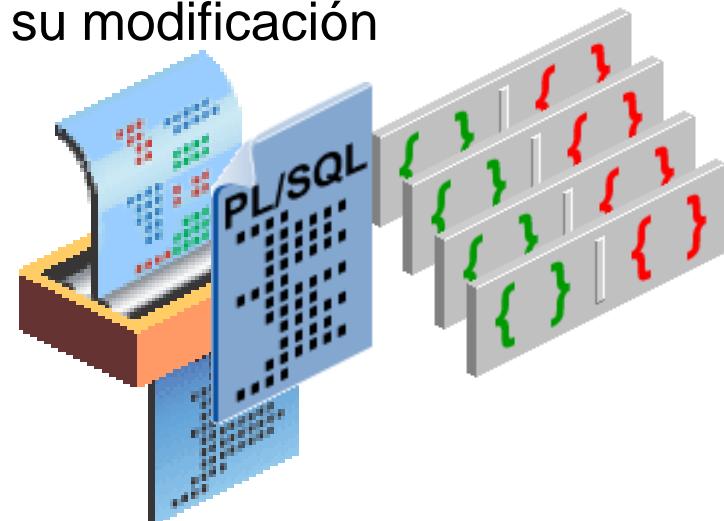
- Utilice los parámetros de inicialización del compilador PL/SQL
- Utilice las advertencias de tiempo de compilación PL/SQL
- Tenga en cuenta que el contenido de este capítulo es específico para 11g y versiones posteriores. Algunos de los ejemplos enumerados pueden o no funcionar en un entorno de 10g.

Agenda

- Utilización de los parámetros de inicialización de la compilación PLSQL_CODE_TYPE y PLSQL_OPTIMIZE_LEVEL PL/SQL
- Uso de las advertencias de tiempo de compilación de PL/SQL:
 - Utilización del parámetro de inicialización PLSQL_WARNING
 - Uso de los subprogramas del paquete DBMS_WARNINGS

Parámetros de inicialización para la compilación PL/SQL

- Desde Oracle Database 11g, PL/SQL utiliza un compilador optimizado.
- No es necesario hacer nada para obtener los beneficios de este nuevo optimizador; Está habilitado de forma predeterminada.
- Dispone de nuevos parámetros para su modificación
 - PLSQL_CODE_TYPE
 - PLSQL_OPTIMIZE_LEVEL
 - PLSQL_CCFLAGS
 - PLSQL_WARNINGS



Utilización de los parámetros de inicialización para la compilación PL/SQL

- **PLSQL_CODE_TYPE:**
 - Este parámetro especifica el modo de compilación para las unidades de biblioteca PL/SQL.

```
PLSQL_CODE_TYPE = { INTERPRETED | NATIVE }
```

- **INTERPRETED**
 - las unidades de la biblioteca PL/SQL se compilarán al formato de bytecode PL/SQL. (Defecto)
- **NATIVE**
 - Las unidades de biblioteca PL/SQL se compilarán en código nativo (**máquina**).
 - Dichos módulos se ejecutarán de forma nativa sin incurrir en ninguna sobrecarga de intérprete.
- Si una unidad de biblioteca PL/SQL se compila de forma nativa, todas las recompilaciones automáticas posteriores de esa unidad de biblioteca utilizarán compilación nativa.

Utilización de los parámetros de inicialización para la compilación PL/SQL

- **PLSQL_OPTIMIZE_LEVEL:**
 - Especifica el nivel de optimización que se utilizará para compilar unidades de biblioteca PL/SQL
- PLSQL_OPTIMIZE_LEVEL** = { 0 | 1 | 2 | 3 }
- **0** Mantiene el orden de evaluación y no realiza operaciones para obtener mejor rendimiento
 - **1** Aplica una amplia gama de optimizaciones a los programas PL/SQL incluyendo la eliminación de cálculos y excepciones innecesarios,
 - **2** Aplica una amplia gama de técnicas de optimización modernas más allá de las del nivel 1, incluyendo cambios dentro del código y su creación
 - **3** Ampliación de las técnicas de nivel 2, incluyendo automáticamente las técnicas no solicitadas específicamente. Esto permite procedimiento inlining,
- Cuanto mayor sea el valor de este parámetro, más esfuerzo tendrá el compilador para optimizar las unidades de librería PL/SQL

Configuración del compilador

Opciones Compilador	Descripción
PLSQL_CODE_TYPE	Especifica el modo de compilación para las unidades de biblioteca PL/SQL.
PLSQL_OPTIMIZE_LEVEL	Especifica el nivel de optimización que se utilizará para compilar unidades de biblioteca PL/SQL.
PLSQL_WARNINGS	Activa o desactiva la generación de informes de mensajes de advertencia por parte del compilador PL/SQL.
PLSQL_CCFLAGS	Controla la compilación condicional de cada unidad de biblioteca PL/SQL independientemente.

- En general, para obtener el rendimiento más rápido, utilice la siguiente configuración:

```
PLSQL_CODE_TYPE = NATIVE  
PLSQL_OPTIMIZE_LEVEL = 2
```

Visualización de los parámetros de inicialización de PL/SQL

Utilice las vistas USER | ALL | DBA_PLSQL_OBJECT_SETTINGS para mostrar la configuración de un objeto PL/SQL

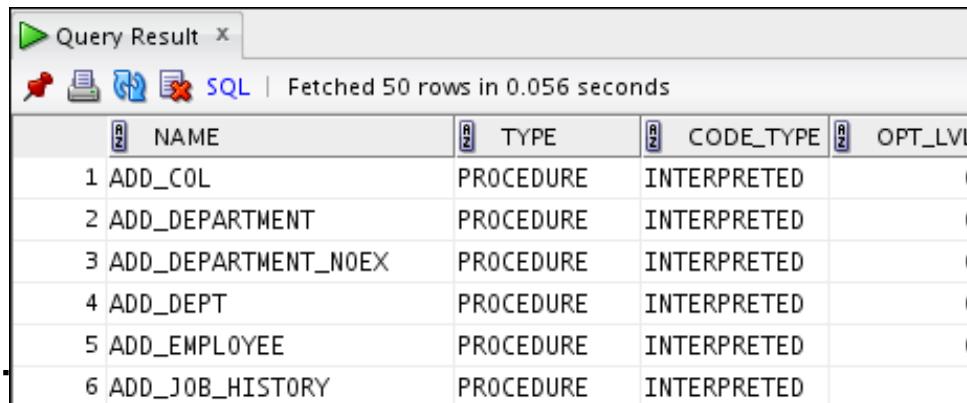
```
DESCRIBE USER_PLSQL_OBJECT_SETTINGS
```

DESCRIBE USER_PLSQL_OBJECT_SETTINGS		
Name	Null	Type
NAME	NOT NULL	VARCHAR2(128)
TYPE		VARCHAR2(12)
PLSQL_OPTIMIZE_LEVEL		NUMBER
PLSQL_CODE_TYPE		VARCHAR2(4000)
PLSQL_DEBUG		VARCHAR2(4000)
PLSQL_WARNINGS		VARCHAR2(4000)
NLS_LENGTH_SEMANTICS		VARCHAR2(4000)
PLSQL_CCFLAGS		VARCHAR2(4000)
PLSCOPE_SETTINGS		VARCHAR2(4000)

Name	El nombre del objeto
Type	Tipo del objeto
PLSQL_OPTIMIZE_LEVEL	Nivel de optimización
PLSQL_CODE_TYPE	Modo de compilación del objeto
PLSQL_DEBUG	Especifica si el objeto se compiló o no para la depuración
PLSQL_WARNINGS	Configuración de Warning
NLS_LENGTH_SEMANTICS	Semantica NLS de compilación
PLSQL_CCFLAGS	El indicador de compilación condicional
PLSCOPE_SETTINGS	Opciones de la recopilación en tiempo de compilación

Visualización y configuración de los parámetros de inicialización de PL/SQL

```
SELECT name, type, plsql_code_type AS CODE_TYPE,  
       plsql_optimize_level AS OPT_LVL  
FROM   user_plsql_object_settings;
```



The screenshot shows a 'Query Result' window from Oracle SQL Developer. The title bar says 'Query Result'. Below it, there are icons for Refresh, Print, Copy, Paste, and Close, followed by 'SQL' and 'Fetched 50 rows in 0.056 seconds'. The main area is a grid table with the following data:

	NAME	TYPE	CODE_TYPE	OPT_LVL
1	ADD_COL	PROCEDURE	INTERPRETED	0
2	ADD_DEPARTMENT	PROCEDURE	INTERPRETED	0
3	ADD_DEPARTMENT_NOEX	PROCEDURE	INTERPRETED	0
4	ADD_DEPT	PROCEDURE	INTERPRETED	0
5	ADD_EMPLOYEE	PROCEDURE	INTERPRETED	0
6	ADD_JOB_HISTORY	PROCEDURE	INTERPRETED	1

- Establezca el valor del parámetro de inicialización del compilador utilizando las instrucciones `ALTER SYSTEM` o `ALTER SESSION`.

Cambiar los parámetros de inicialización de PL/SQL: Ejemplo

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL = 1;
ALTER SESSION SET PLSQL_CODE_TYPE = 'NATIVE';
```

```
session SET altered.
session SET altered.
```

```
-- code displayed in the notes page
CREATE OR REPLACE PROCEDURE add_job_history
. . .
```

```
@code_11_09_sa.sql
```

NAME	TYPE	CODE_TYPE	OPT_LVL
1 ADD_COL	PROCEDURE	INTERPRETED	0
2 ADD_DEPARTMENT	PROCEDURE	INTERPRETED	0
3 ADD_DEPARTMENT_NOEX	PROCEDURE	INTERPRETED	0
4 ADD_DEPT	PROCEDURE	INTERPRETED	0
5 ADD_EMPLOYEE	PROCEDURE	INTERPRETED	0
6 ADD_JOB_HISTORY	PROCEDURE	NATIVE	1

Para aplicar la compilación nativa a todo el código PL/SQL, debe recompilar cada uno.

Los scripts del directorio rdmbs / admin)se proporcionan para lograr la conversión a compilación **nativa** ([dbmsupgnv.sql](#)) o compilación **interpretada** ([dbmsupgin.sql](#)).

Agenda

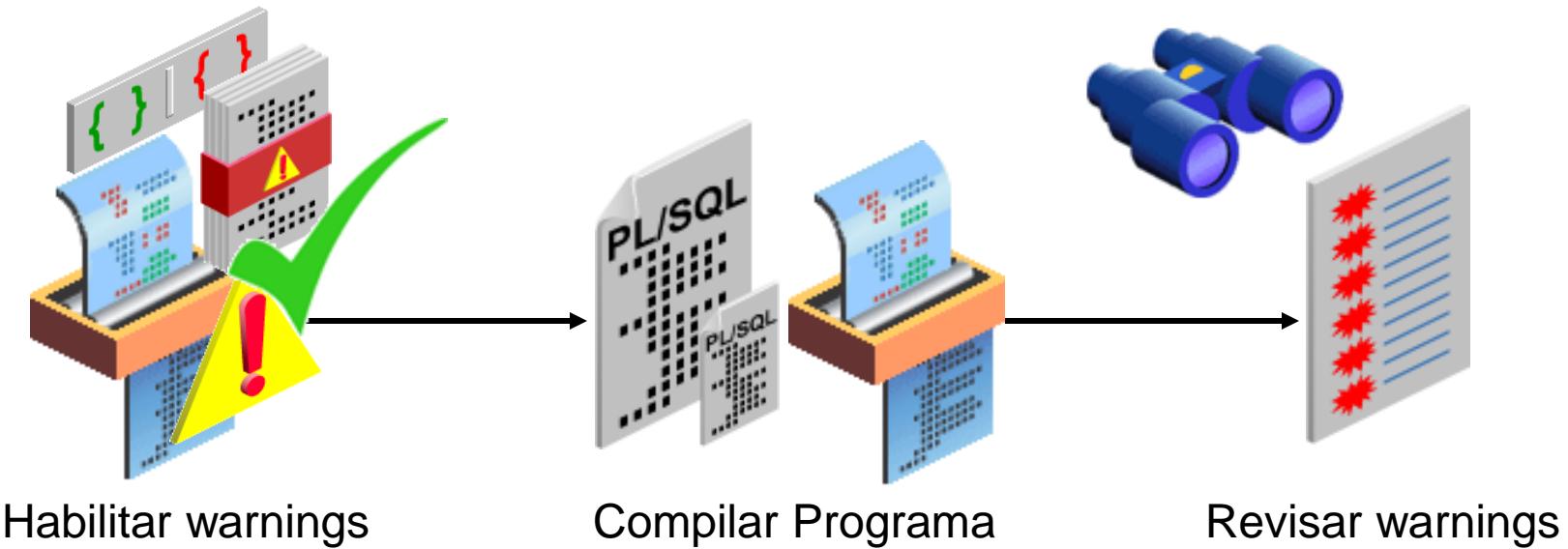
- Utilización de los parámetros de inicialización de la compilación PLSQL_CODE_TYPE y PLSQL_OPTIMIZE_LEVEL PL/SQL
- Uso de las advertencias de tiempo de compilación de PL/SQL:
 - Utilización del parámetro de inicialización PLSQL_WARNING
 - Uso de los subprogramas del paquete DBMS_WARNINGS

Introducción a los Warning de compilación de PL/SQL para subprogramas

- Para que sus programas sean más robustos y evitar problemas en tiempo de ejecución, puede activar la comprobación de ciertas condiciones de advertencia. (WARNINGS)
- Estos WARNINGS no son lo suficientemente graves como para producir un error y evitar que compile un subprograma.
 - Pueden señalar algo en el subprograma que produce un resultado indefinido
 - Puede crear un problema de rendimiento.
- En las versiones anteriores a Oracle 10g, la compilación de un programa PL/SQL tuvo dos resultados posibles:
 - Éxito, produciendo una unidad compilada válida
 - Fallo, con errores de compilación que indican que el programa tiene errores de sintaxis o semánticos

Introducción a los Warning de compilación de PL/SQL para subprogramas

- Las advertencias del compilador (WARNINGS) permiten a los desarrolladores evitar las trampas de codificación comunes, mejorando así la productividad.



Introducción a los Warning de compilación de PL/SQL para subprogramas

NOTAS

- Con la activación de `Warnings`, la compilación de un programa PL/SQL podría tener resultados adicionales posibles:
 - Éxito con las **advertencias de compilación**
 - Fallo con **errores** de compilación y **advertencias** de compilación
- El compilador puede emitir mensajes de advertencia incluso en una compilación correcta.
- Se debe corregir un **error** de compilación para poder utilizar el procedimiento almacenado, mientras que una advertencia tiene fines informativos.
- Ejemplos de mensajes de advertencia
 - SP2-0804:** Procedimiento creado con advertencias de compilación
 - PLW-07203:** Advertencia relacionada con el Hint del compilador `NOCOPY`.

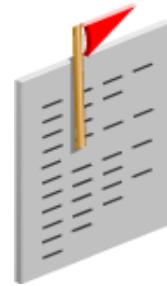
Beneficios de las advertencias del compilador

- Hacer programas más robustos y evitar problemas en tiempo de ejecución
- Identificar posibles problemas de rendimiento
- Identificar factores que producen resultados no definidos
- Los bloques **anónimos** no producen ninguna advertencia, solo los **subprogramas PL/SQL**.
- Todos los mensajes de advertencia de PL/SQL utilizan el prefijo `PLW`.



Categorías de mensajes de advertencia de compilación de PL/SQL

- Los mensajes de advertencia PL/SQL se dividen en categorías, de modo que puede suprimir o mostrar grupos de advertencias similares durante la compilación.



SEVERE

Mensajes para condiciones que pueden causar un comportamiento **inesperado o resultados incorrectos**



PERFORMANCE

Mensajes para condiciones que pueden causar problemas de **rendimiento**



INFORMATIONAL

Mensajes que no tienen un efecto sobre el rendimiento o la corrección, pero que desee cambiar para que el **código sea más fácil de mantener**,



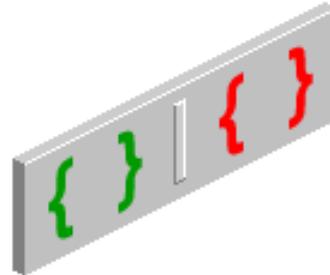
ALL

Todos los anteriores

Ajuste de los niveles de advertencia de los mensajes del compilador

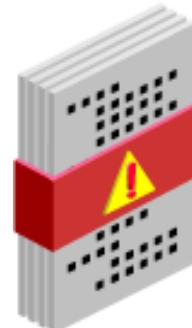
- Puede establecer los niveles de advertencia de los mensajes del compilador utilizando uno de los métodos siguientes:

- Utilización del parámetro de inicialización `PLSQL_WARNINGS`



`PLSQL_WARNINGS`
initialization parameter

- Programáticamente mediante el uso del paquete `DBMS_WARNING`

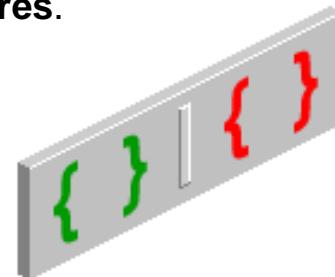


`DBMS_WARNING`
package

Ajuste de los niveles de advertencia de los mensajes del compilador

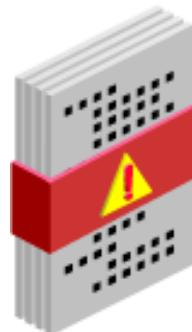
- Parámetro de inicialización `PLSQL_WARNINGS`
 - El parámetro `PLSQL_WARNINGS` habilita o deshabilita aparición de los mensajes de WARNING y qué mensajes de advertencia mostrar como errores. (`SEVERE`, `PERFORMANCE`, `INFORMACIÓN`, `ALL`)
 - Los parámetros del parámetro `PLSQL_WARNINGS` se almacenan junto con cada subprograma compilado.
 - Puede utilizar el parámetro de inicialización `PLSQL_WARNINGS` para realizar lo siguiente:
 - Activar o desactivar la notificación de todas las advertencias, advertencias de una categoría seleccionada o mensajes de advertencia específicos.
 - Trate todas las advertencias, una categoría de advertencia seleccionada o mensajes de advertencia específicos como errores.

`PLSQL_WARNINGS`
initialization parameter



Ajuste de los niveles de advertencia de los mensajes del compilador

- Uso del paquete `DBMS_WARNING`
 - El paquete `DBMS_WARNING` proporciona una forma de manipular el comportamiento de los mensajes de advertencia PL/SQL.
 - Mediante este paquete podemos leer y modificar la configuración del parámetro `PLSQL_WARNINGS` para controlar qué tipos de advertencias.
 - Este paquete proporciona la interfaz para consultar, modificar y eliminar la configuración actual del sistema o de la sesión.



`DBMS_WARNING`
package

Configuración de niveles de advertencia del compilador: Utilizar PLSQL_WARNINGS

- El valor del parámetro comprende una lista separada por comas de las cláusulas (Clausula = Cualificador : Valor)
 - Cualificador: ENABLE, DISABLE y ERROR.
 - Valores: ALL, SEVERE, INFORMATION, PERFORMANCE, Numero

```
ALTER [SESSION|SYSTEM]
PLSQL_WARNINGS = 'value_clause1'[ ,'value_clause2' ] ...
```

```
value_clause = Qualifier Value : Modifier Value
```

```
Qualifier Value = { ENABLE | DISABLE | ERROR }
```

```
Modifier Value = { ALL | SEVERE | INFORMATIONAL |
PERFORMANCE | Numeric_Value }
```

Configuración de niveles de advertencia de compilador: PLSQL_WARNINGS: Ejemplo

```
ALTER SESSION SET plsql_warnings = 'enable:severe',  
                  'enable:performance',  
                  'disable:informational';
```

Se habilitará advertencia SEVERE y PERFORMANCE
y deshabilitando advertencias INFORMATIONAL.

```
ALTER SESSION  
SET plsql_warnings = 'enable:severe';
```

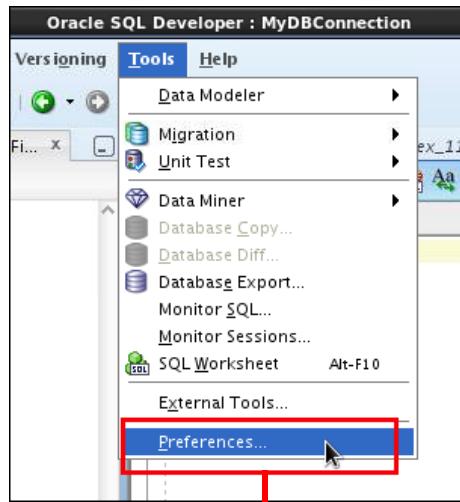
Se habilitará advertencia SEVERE

```
ALTER SESSION SET PLSQL_WARNINGS='ENABLE:SEVERE',  
                 'DISABLE:PERFORMANCE' , 'ERROR:05003';
```

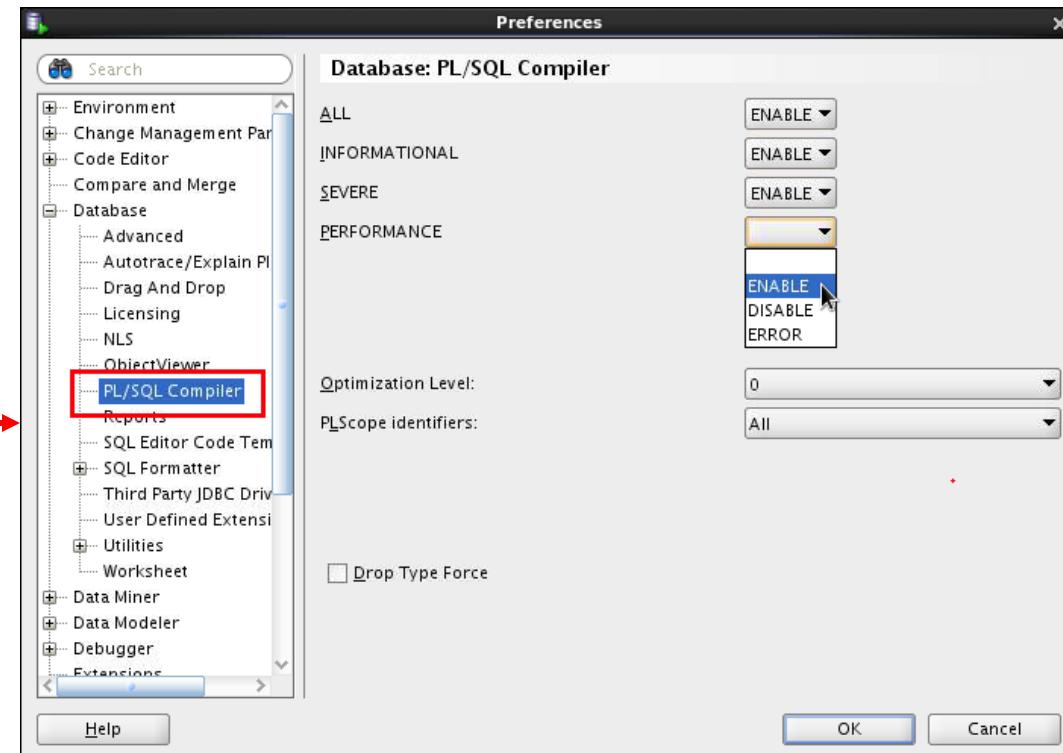
Podemos modificar el tratamiento de un WARNING habitual que se produzca, por ejemplo WARNING 05003' y que PLSQL_WARNINGS hace que la condición active un mensaje de error (PLS_05003)

Range 5000-5999 for severe
Range 6000-6249 for informational
Range 7000-7249 for performance

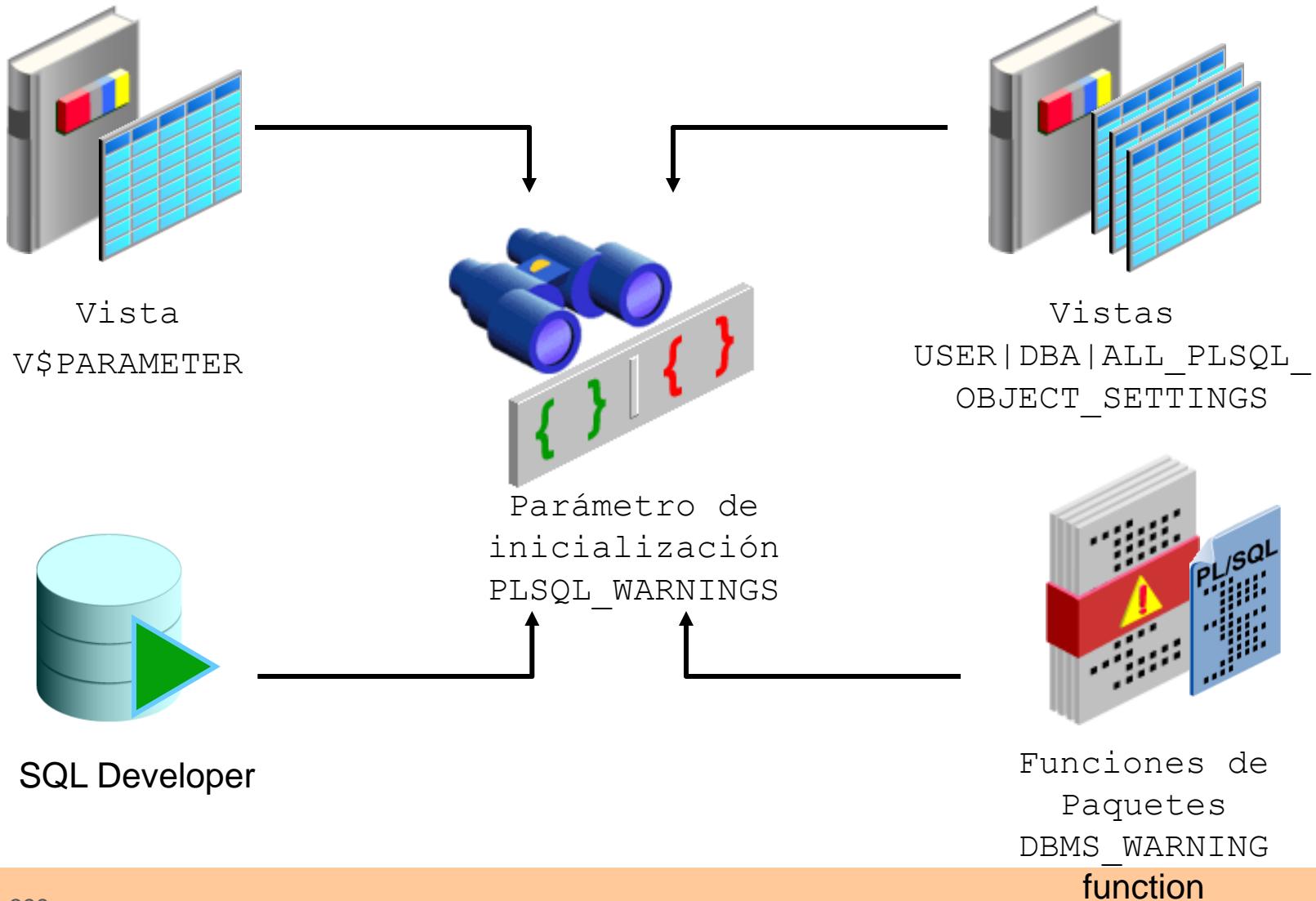
Configuración de PLSQL_WARNINGS en SQL Developer



Tools – Preferencias – PLSQL Compiler



Visualización de la configuración actual de PLSQL_WARNINGS



Visualización de la configuración actual de PLSQL_WARNINGS

- Puede examinar la configuración actual del parámetro PLSQL_WARNINGS emitiendo una instrucción `SELECT` en la vista V\$PARAMETER.
- Por ejemplo:

```
ALTER SESSION SET plsql_warnings = 'enable:severe',
               'enable:performance', 'enable:informational';
Session altered

SELECT value FROM v$parameter
WHERE name='plsql_warnings';

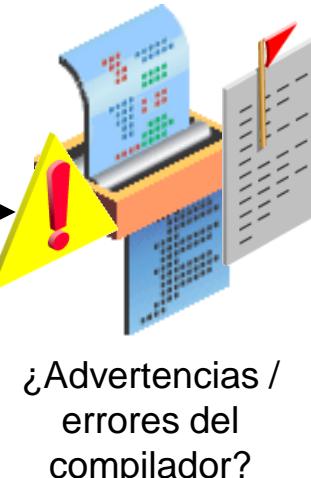
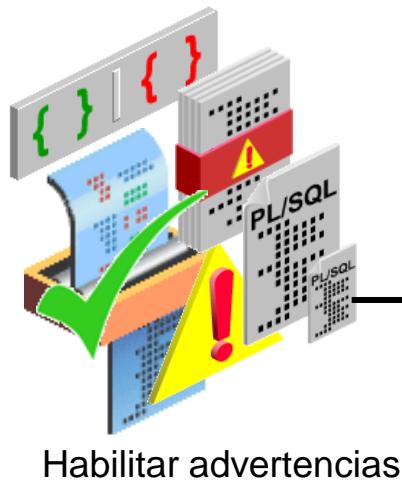
VALUE
-----
ENABLE:ALL
```

Visualización de la configuración actual de PLSQL_WARNINGS

- De forma alternativa, puede utilizar el paquete y el procedimiento DBMS_WARNING.GET_WARNING_SETTING_STRING para recuperar la configuración actual del parámetro PLSQL_WARNINGS:

```
DECLARE s VARCHAR2(1000);
BEGIN
    s := dbms_warning.>get_warning_setting_string();
    dbms_output.put_line (s);
END;
/
```

Visualización PLSQL_WARNINGS : SQL Developer, SQL*Plus, o Vistas del Diccionario Datos



Visualización PLSQL_WARNINGS : SQL Developer, SQL*Plus, o Vistas del Diccionario Datos

- **SQL*Plus**
 - SQL*Plus indica que se ha producido una advertencia de compilación, con el siguiente mensaje:
SP2-08xx: <object> created with compilation warnings.”
 - Puede mostrar los mensajes de advertencia del compilador utilizando uno de los métodos siguientes:
`SHOW ERRORS`
 - Este comando muestra los errores del compilador incluyendo las nuevas advertencias del compilador y los mensajes informativos.
 - Este comando se invoca inmediatamente después de utilizar un comando `CREATE [PROCEDURE | FUNCTION | PACKAGE] .`
 - Debe activar las advertencias del compilador antes de compilar el programa.

Visualización PLSQL_WARNINGS : SQL*Plus, o Vistas del Diccionario Datos

- Uso de las vistas del diccionario de datos
 - Puede seleccionar entre las vistas del diccionario de datos **USER_ | ALL_ | DBA_ERRORS** para mostrar las advertencias del compilador PL / SQL.
 - La columna **ATTRIBUTES** de estas vistas tiene un nuevo atributo llamado **WARNING**
 - El mensaje de advertencia aparece en la columna **TEXTO**.

Mensajes de advertencia SQL*Plus: Ejemplo

```
CREATE OR REPLACE PROCEDURE bad_proc(p_out ...) IS
BEGIN
. . . ;
END;
/
```

SP2-0804: Procedure created with compilation warnings.

SHOW ERRORS;

Errors for PROCEDURE BAD_PROC:

LINE/COL ERROR

6/24 PLW-07203: parameter 'p_out' may benefit
 from use of the NOCOPY compiler hint

Directrices para el uso de PLSQL_WARNINGS

- Los parámetros del parámetro PLSQL_WARNINGS se almacenan junto con cada subprograma compilado.
- Si vuelve a compilar el subprograma utilizando una de las siguientes instrucciones, se utilizarán las configuraciones actuales para esa sesión:
 - CREATE OR REPLACE
 - ALTER ... COMPILE
- Si vuelve a compilar el subprograma utilizando la instrucción ALTER ... COMPILE con la cláusula REUSE SETTINGS , se utiliza el ajuste original almacenado con el programa.

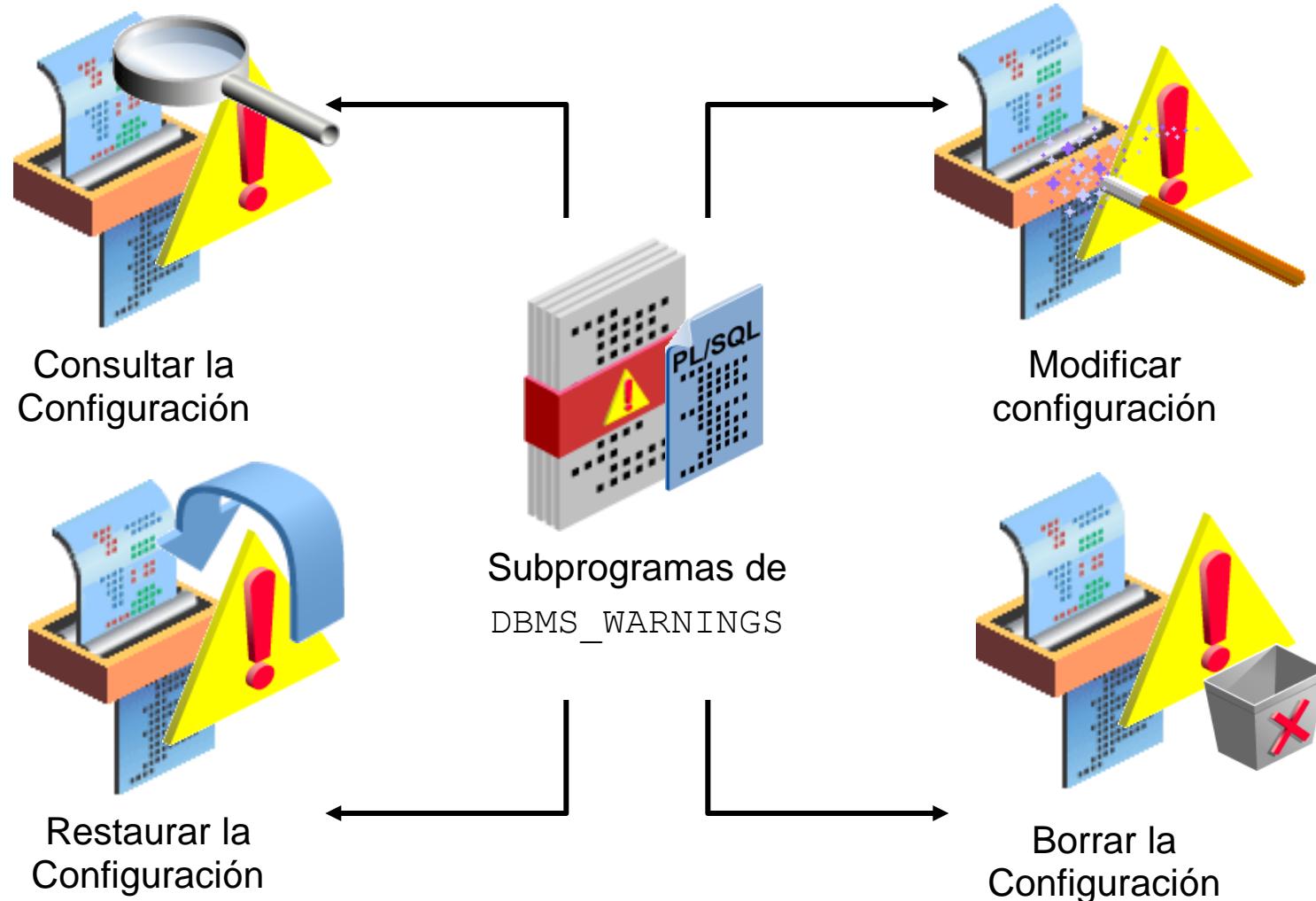
Agenda

- Utilización de los parámetros de inicialización de la compilación PLSQL_CODE_TYPE y PLSQL_OPTIMIZE_LEVEL PL/SQL
- Uso de las advertencias de tiempo de compilación de PL/SQL:
 - Utilización del parámetro de inicialización PLSQL_WARNING
 - Uso de los subprogramas del paquete DBMS_WARNINGS

Visualización PLSQL_WARNINGS : Usando el paquete DBMS_WARNING

- Podemos gestionar la configuración de los `WARNINGS` de PLSQL mediante el paquete `DBMS_WARNING`
- Esta gestión se puede realizar a nivel de Sistema o de Sesión PL/SQL
- Este paquete proporciona la interfaz para **consultar, modificar y eliminar** la configuración actual del sistema o de la sesión.
- Utilizando las rutinas de la interfaz del paquete, puede controlar los mensajes de advertencia de PL/SQL de forma programática para adaptarse a sus necesidades.

Visualización PLSQL_WARNINGS: Usando el paquete DBMS_WARNING



Uso de los subprogramas de paquete DBMS_WARNING

Escenario	Subprogramas a utilizar
Establecer advertencias	<code>ADD_WARNING_SETTING_CAT</code> (procedure) <code>ADD_WARNING_SETTING_NUM</code> (procedure)
Consultar advertencias	<code>GET_WARNING_SETTING_CAT</code> (function) <code>GET_WARNING_SETTING_NUM</code> (function) <code>GET_WARNING_SETTING_STRING</code> (function)
Modificar advertencias	<code>SET_WARNING_SETTING_STRING</code> (procedure)
Obtener los nombres de las categorías de advertencias	<code>GET_CATEGORY</code> (function)

Procedimientos DBMS_WARNING: sintaxis, parámetros y valores permitidos

```
EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_CAT (
```

warning_category	IN	VARCHAR2,	ALL / INFORMATIONAL SEVERE / PERFORMANCE .
warning_value	IN	VARCHAR2,	ENABLE / DISABLE / ERROR
scope	IN	VARCHAR2);	SYSTEM / SESSION

```
EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_NUM (
```

warning_number	IN	NUMBER,	Número de Error
warning_value	IN	VARCHAR2,	ENABLE / DISABLE / ERROR
scope	IN	VARCHAR2);	SYSTEM / SESSION

```
EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING (
```

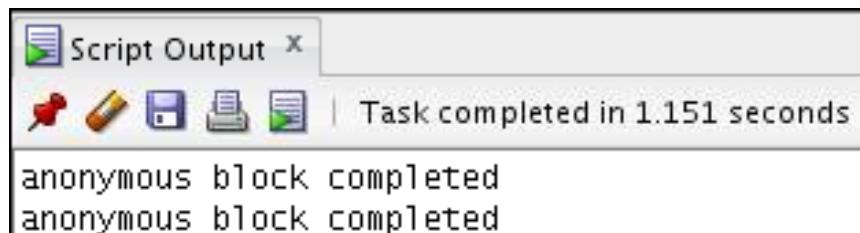
warning_value	IN	VARCHAR2,	ENABLE / DISABLE / ERROR
scope	IN	VARCHAR2);	SYSTEM / SESSION

Procedimientos DBMS_WARNING: Ejemplo

```
-- Establezca la siguiente cadena de ajuste de
-- advertencia en la sesión actual:
-- ENABLE:INFORMATIONAL,
-- DISABLE:PERFORMANCE,
-- ENABLE:SEVERE

EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING (
    'ENABLE:ALL', 'SESSION');

EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_CAT (
    'PERFORMANCE', 'DISABLE', 'SESSION');
```



Funciones DBMS_WARNING: sintaxis, parámetros y valores permitidos

```
DBMS_WARNING.GET_WARNING_SETTING_CAT (-  
    warning_category IN VARCHAR2) RETURN warning_value;
```

ALL/INFORMATIONAL
SEVERE/ PERFORMANCE.

ENABLE/DISABLE/ERROR

```
DBMS_WARNING.GET_WARNING_SETTING_NUM (-  
    warning_number IN NUMBER) RETURN warning_value;
```

Numero de Error

ENABLE/DISABLE/ERROR

```
DBMS_WARNING.GET_WARNING_SETTING_STRING  
    RETURN pls_integer;
```

Determinar los ajustes de
advertencia de la sesión actual

```
DBMS_WARNING.GET_CATEGORY (-  
    warning_number IN pls_integer) RETURN VARCHAR2;
```

Numero de Error

Las funciones DBMS_WARNING: Ejemplo

```
-- Determinar los ajustes de advertencia de la sesión  
actual  
SET SERVEROUTPUT ON  
EXECUTE DBMS_OUTPUT.PUT_LINE( -  
DBMS_WARNING.GET_WARNING_SETTING_STRING);
```

```
anonymous block completed  
ENABLE:INFORMATIONAL,DISABLE:PERFORMANCE,ENABLE:SEVERE
```

```
-- Determinar la categoría para el número de mensaje de  
-- advertencia PLW-07203  
SET SERVEROUTPUT ON  
EXECUTE DBMS_OUTPUT.PUT_LINE( -  
DBMS_WARNING.GET_CATEGORY(7203));
```

```
anonymous block completed  
PERFORMANCE
```

Uso de DBMS_WARNING: Ejemplo

```
CREATE OR REPLACE PROCEDURE compile_code(p_pkg_name VARCHAR2) IS
    v_warn_value    VARCHAR2(200);
    v_compile_stmt VARCHAR2(200) :=
        'ALTER PACKAGE ' || p_pkg_name || ' COMPILE';

BEGIN
    v_warn_value := DBMS_WARNING.GET_WARNING_SETTING_STRING;
    DBMS_OUTPUT.PUT_LINE('Current warning settings: ' ||
        v_warn_value);
    DBMS_WARNING.ADD_WARNING_SETTING_CAT(
        'PERFORMANCE', 'DISABLE', 'SESSION');
    DBMS_OUTPUT.PUT_LINE('Modified warning settings: ' ||
        DBMS_WARNING.GET_WARNING_SETTING_STRING);
    EXECUTE IMMEDIATE v_compile_stmt;
    DBMS_WARNING.SET_WARNING_SETTING_STRING(v_warn_value,
        'SESSION');
    DBMS_OUTPUT.PUT_LINE('Restored warning settings: ' ||
        DBMS_WARNING.GET_WARNING_SETTING_STRING);
END;
/
```

El código suprime las advertencias de PERFORMANCE

El código restaura las advertencias de PERFORMANCE

Uso de DBMS_WARNING: Ejemplo

```
EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING(-  
  'ENABLE:ALL', 'SESSION');
```

anonymous block completed

Habilita todas las advertencias del compilador

```
@/home/oracle/labs/plpu/code_ex/code_ex_scripts/code_11_33_s.sql
```

PROCEDURE compile_code compiled

```
EXECUTE compile_code('EMP_PKG');
```

anonymous block completed
Current warning settings: ENABLE:ALL
Modified warning settings: ENABLE:INFORMATIONAL,DISABLE:PERFORMANCE,ENABLE:SEVERE
Restored warning settings: ENABLE:ALL

Uso del mensaje de advertencia PLW 06009

- Como una buena práctica de programación, debe hacer que su manejador de excepciones OTHERS pase la excepción hacia arriba a la subrutina llamante.
 - Si no agrega esta funcionalidad, corre el riesgo de que las excepciones pasen desapercibidas.
- Para evitar este fallo en su código, puede activar las advertencias para su sesión y recompilar el código que desea verificar.
- Si el controlador OTHERS no maneja la excepción, la advertencia del **PLW 06009** le informará.
 - Tenga en cuenta que esta advertencia es específica de Oracle Database 11g y otras versiones.

Uso del mensaje de advertencia PLW 06009

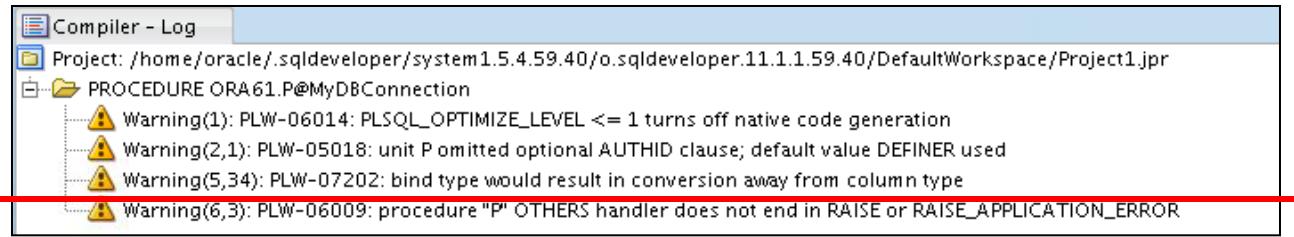
- La advertencia PLW 06009 indica que el controlador OTHERS de su subrutina PL/SQL puede salir sin ejecutar:
 - Alguna orden RAISE, or
 - Una llamada al procedimiento estándar RAISE_APPLICATION_ERROR
- Una buena práctica de programación sugiere que los manejadores de OTHERS siempre deben pasar una excepción hacia arriba.



Warning PLW 06009 : Ejemplo

```
CREATE OR REPLACE PROCEDURE p(i IN VARCHAR2)
IS
BEGIN
    INSERT INTO t(col_a) VALUES (i);
EXCEPTION
    WHEN OTHERS THEN null;
END p;
/
ALTER PROCEDURE P COMPILE
PLSQL_warnings = 'enable:all' REUSE SETTINGS;
```

Después de ejecutar y compilar el procedimiento, observamos que , la **ficha Compiler – Log** muestra la advertencia PLW-06009.



```
SELECT *
FROM user_errors
WHERE name = 'P'
```

Query Result						
SQL All Rows Fetched: 5 in 0.03 seconds						
#	NAME	TYPE	SEQUENCE	LINE	POSITION	TEXT
1	P	PROCEDURE	1	4	34	PLW-07202: bind type would result in conversion away from column type
2	P	PROCEDURE	2	1	1	PLW-05018: unit P omitted optional AUTHID clause; default value DEFINER used
3	P	PROCEDURE	3	0	0	PLW-06015: parameter PLSQL_DEBUG is deprecated; use PLSQL_OPTIMIZE_LEVEL = 1
4	P	PROCEDURE	4	0	0	PLW-06014: PLSQL_OPTIMIZE_LEVEL <= 1 turns off native code generation
5	P	PROCEDURE	5	5	3	PLW-06009: procedure "P" OTHERS handler does not end in RAISE or RAISE_APPLICATION_ERROR

Quiz

Las categorías de mensajes de advertencia de tiempo de compilación de PL/SQL son:

- a. SEVERE
- b. PERFORMANCE
- c. INFORMATIONAL
- d. All
- e. CRITICAL

Resumen

En esta lección, debes haber aprendido a:

- Utilice los parámetros de inicialización del compilador PL/SQL
- Utilice las advertencias de tiempo de compilación PL/SQL

Práctica 11

En esta lección, realiza las siguientes prácticas:

- Visualización de los parámetros de inicialización del compilador
- Habilitación de la compilación nativa para su sesión y compilación de un procedimiento
- Deshabilitar las advertencias del compilador ya continuación, restaurar la configuración de advertencia de sesión original
- Identificar las categorías para algunos números de mensaje de advertencia del compilador

12

Gestión de Dependencias



ORACLE®

Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Seguimiento de dependencias procedimentales
- Predecir el efecto de cambiar un objeto de base de datos en procedimientos y funciones
- Administrar dependencias procedurales

Descripción general de las dependencias de objetos de esquema

- Algunos tipos de objetos de esquema pueden hacer referencia a otros objetos en sus definiciones
 - Por Ejemplo, una vista se define mediante una consulta que hace referencia a tablas u otras vistas
- Si la definición del objeto A hace referencia al objeto B, entonces:
 - A es un objeto **dependiente** (con respecto a B) y
 - B es un objeto **referenciado** (con respecto a A).
- Problemas de dependencia
 - Si se modifica la definición de un objeto **referenciado**, los objetos **dependientes** pueden o no continuar funcionando correctamente
- El servidor Oracle registra automáticamente dependencias entre objetos (`USER_OBJECTS`) VALID – INVALID

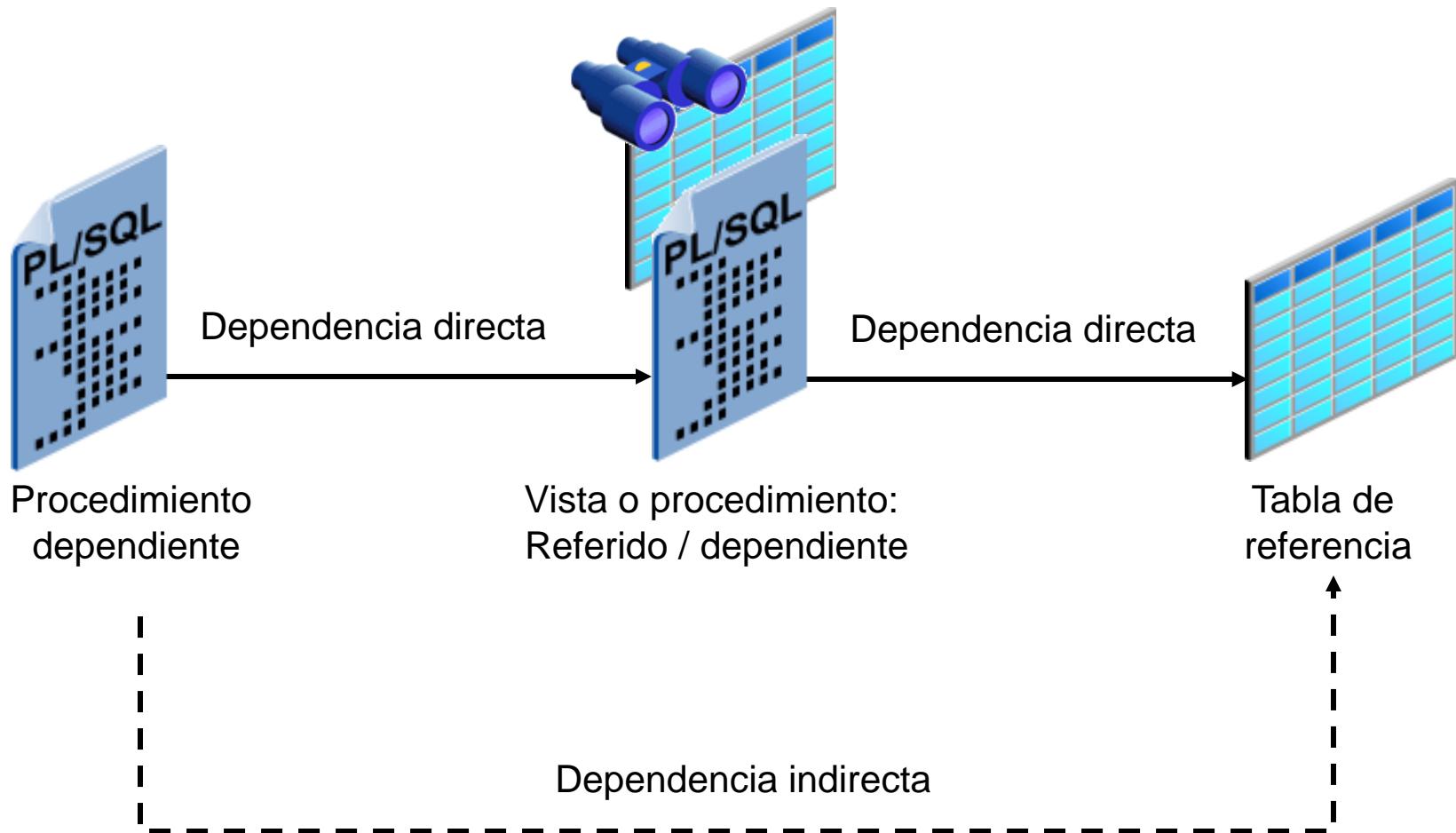
Descripción general de las dependencias de objetos de esquema

Tipo de objeto	Puede ser dependiente o referenciado
Package body	Dependiente sólo
Package specification	Ambos
Sequence	Referenciado sólo
Subprogram	Ambos
Synonym	Ambos
Table	Ambos
Trigger	Ambos
User-defined object	Ambos
User-defined collection	Ambos
View	Ambos

Descripción general de las dependencias de objetos de esquema

- Un procedimiento o función puede directa o indirectamente (a través de una vista intermedia, procedimiento, función o procedimiento o función empaquetada) hacer referencia a los siguientes objetos:
 - Tablas
 - Vistas
 - Secuencias
 - Procedimientos
 - Funciones
 - Procedimientos o Funciones Empaquetadas
- Si el estado de un objeto de esquema es VALID:
 - El objeto se ha compilado y puede utilizarse inmediatamente cuando se hace referencia.
- Si el estado de un objeto de esquema es INVALID
 - El objeto de esquema debe ser compilado antes de que pueda ser utilizado. Oracle lo hace de forma automática *si puede*

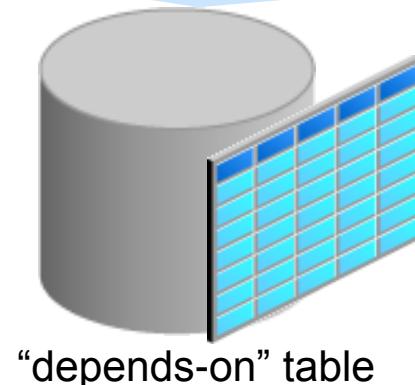
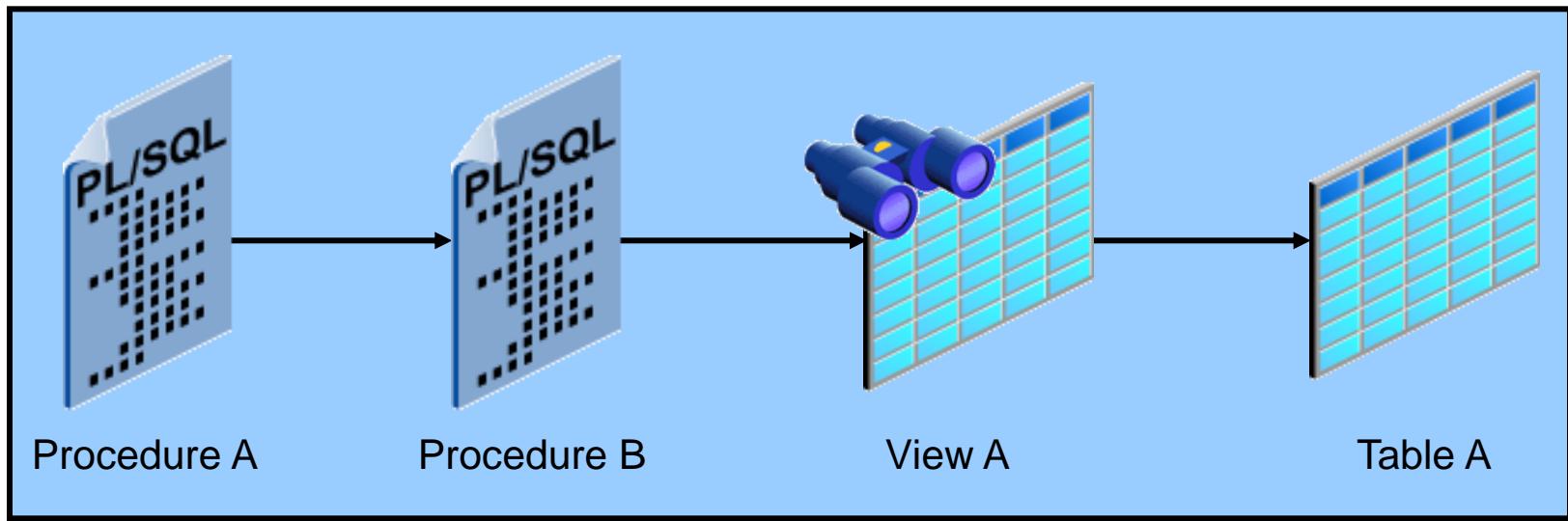
Dependencias



Dependencias locales directas

- En el caso de las dependencias locales, los objetos están en el **mismo nodo en la misma base de datos**
- El servidor Oracle administra automáticamente todas las dependencias locales, utilizando tablas internas.
- Cuando se modifica un objeto referenciado, los objetos dependientes a veces son invalidados.
 - La próxima vez que se llame un objeto invalidado, el servidor Oracle lo recompila automáticamente.
- Si **altera la definición** de un objeto referenciado, los objetos dependientes pueden o no continuar funcionando sin error, dependiendo del tipo de alteración.

Dependencias locales directas



Consultar dependencias de objetos directos: mediante la vista USER_DEPENDENCIES

- Mediante la tabla USER_DEPENDENCIES, se puede determinar qué objetos de base de datos se recompilan **manualmente**.
- También mediante las vistas ALL_DEPENDENCIES y DBA_DEPENDENCIES se podrá ver.
 - Contienen la columna OWNER adicional, que hace referencia al propietario del objeto.
- Columnas USER_DEPENDENCIES
 - **NAME:** El nombre del objeto dependiente
 - **TYPE:** El tipo del objeto dependiente (PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER, or VIEW)
 - **REFERENCED_OWNER:** El esquema del objeto referenciado
 - **REFERENCED_NAME:** El nombre del objeto referenciado
 - **REFERENCED_TYPE:** El tipo del objeto referenciado
 - **REFERENCED_LINK_NAME:** Database link usado

Consultar dependencias de objetos directos: mediante la vista USER_DEPENDENCIES

```
desc user_dependencies
Name          Null    Type
-----
NAME          NOT NULL VARCHAR2(128)
TYPE          VARCHAR2(18)
REFERENCED_OWNER  VARCHAR2(128)
REFERENCED_NAME   VARCHAR2(128)
REFERENCED_TYPE    VARCHAR2(18)
REFERENCED_LINK_NAME VARCHAR2(128)
SCHEMADID      NUMBER
DEPENDENCY_TYPE  VARCHAR2(4)
```

```
SELECT name, type, referenced_name, referenced_type
FROM   user_dependencies
WHERE  referenced_name IN ('EMPLOYEES', 'EMP_VW' );
```

NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE
1 EMP_PKG	PACKAGE	EMPLOYEES	TABLE
2 EMP_PKG	PACKAGE BODY	EMPLOYEES	TABLE
3 EMPLOYEE_INITJOBS_TRG	TRIGGER	EMPLOYEES	TABLE
4 CHECK_SALARY_TRG	TRIGGER	EMPLOYEES	TABLE
5 EMP_DETAILS	VIEW	EMPLOYEES	TABLE
6 EMPLOYEE_REPORT	PROCEDURE	EMPLOYEES	TABLE

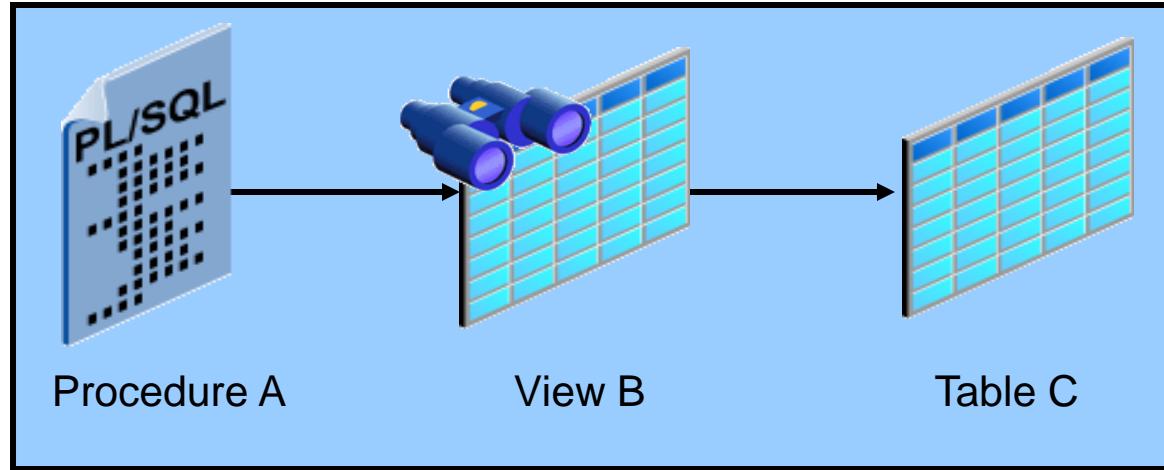
...

Consultar el estado de un objeto

Cada objeto de base de datos tiene uno de los siguientes valores de estado:

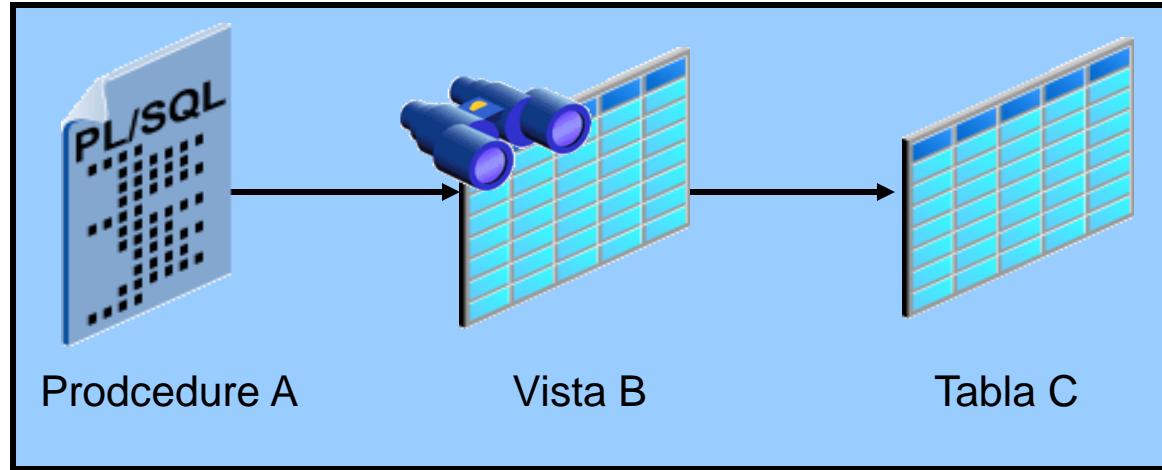
Status	Description
VALID	El objeto se compiló correctamente, utilizando la definición actual en el diccionario de datos.
COMPILED WITH ERRORS	El intento más reciente de compilar el objeto produjo errores.
INVALID	El objeto está marcado como no válido porque un objeto que hace referencia ha cambiado. (Sólo un objeto dependiente puede ser inválido.)
UNAUTHORIZED	Se anuló un privilegio de acceso en un objeto referenciado. (Sólo un objeto dependiente puede no estar autorizado.)

Invalidación de objetos dependientes



- Procedure A is a direct dependent of View B. View B is a direct dependent of Table C. Procedure A is an indirect dependent of Table C.
- Direct dependents are invalidated only by changes to the referenced object that affect them.
- Indirect dependents can be invalidated by changes to the reference object that do not affect them.

Invalidación de objetos dependientes



- El **procedimiento A** depende directamente de la **vista B**.
- La **vista B** depende directamente de la **Tabla C**.
- El **Procedimiento A** depende de forma indirecta de la **Tabla C**.
 - Las **dependencias directas**
 - Se invalidan sólo por cambios en el objeto referenciado.
 - Las **dependencias indirectas**
 - Pueden ser invalidados por cambios en el objeto de referencia no directo. *Invalidación en cascada*

Cambio en objeto que produce invalidaciones: Ejemplo

```
CREATE VIEW commissioned AS  
SELECT first_name, last_name, commission_pct FROM employees  
WHERE commission_pct > 0.00;
```

```
CREATE VIEW six_figure_salary AS  
SELECT * FROM employees  
WHERE salary >= 100000;
```

```
SELECT object_name, status  
FROM user_objects  
WHERE object_type = 'VIEW';
```

	OBJECT_NAME	STATUS
1	EMP_DETAILS_VIEW	VALID
2	EMP_DETAILS	VALID
3	COMMISSIONED	VALID
4	SIX FIGURE SALARY	VALID

Dos vistas creadas recientemente se basan en la tabla EMPLOYEES del esquema HR. El estado de las vistas creadas recientemente es VALID.

Cambio en objeto que produce invalidaciones: Ejemplo

```
ALTER TABLE employees MODIFY email VARCHAR2(50);

SELECT object_name, status
FROM user_objects
WHERE object_type = 'VIEW';
```

	OBJECT_NAME	STATUS
1	EMP_DETAILS_VIEW	VALID
2	EMP_DETAILS	VALID
3	COMMISSIONED	VALID
4	SIX FIGURE SALARY	INVALID

Modificamos la columna EMAIL
de la tabla EMPLOYEES de 25
a 50 caracteres

- La vista COMMISSIONED no se ha invalidado por no incluir la columna EMAIL, sin embargo, la vista SIXFIGURES se invalida porque se seleccionan todas las columnas de la tabla y entre ellas EMAIL

Visualización de dependencias directas e indirectas

- Para poder ver las dependencias directas e indirectas necesitamos disponer de las vistas DEPTREE y IDEPTREE

Para ello ejecutaremos las siguientes etapas

- Ejecute el script `utldtree.sql` que crea los objetos que le permiten mostrar las dependencias directas e indirectas

```
@/home/oracle/labs/plpu/labs/utldtree.sql
```

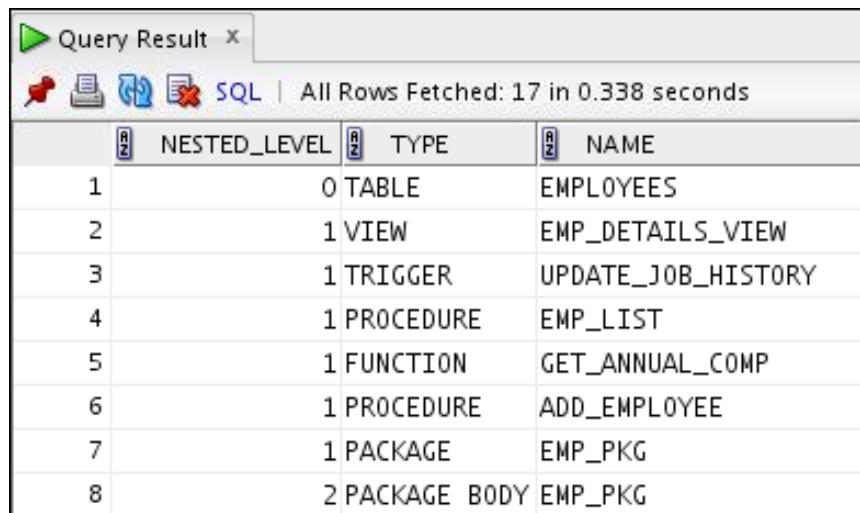
- Rellene la tabla DEPTREE_TEMP TAB con información para un objeto referenciado en particular invocando el procedimiento DEPTREE_FILL

```
EXECUTE deptree_fill('TABLE', 'ORA61', 'EMPLOYEES')
```

Visualización de dependencias mediante la vista DEPTREE

- vista [DEPTREE](#) representación tabular de todos los objetos dependientes.
- Vista [IDEPTREE](#) representación indentada de la misma información

```
SELECT      nested_level, type, name
FROM        deptree
ORDER BY    seq#;
```

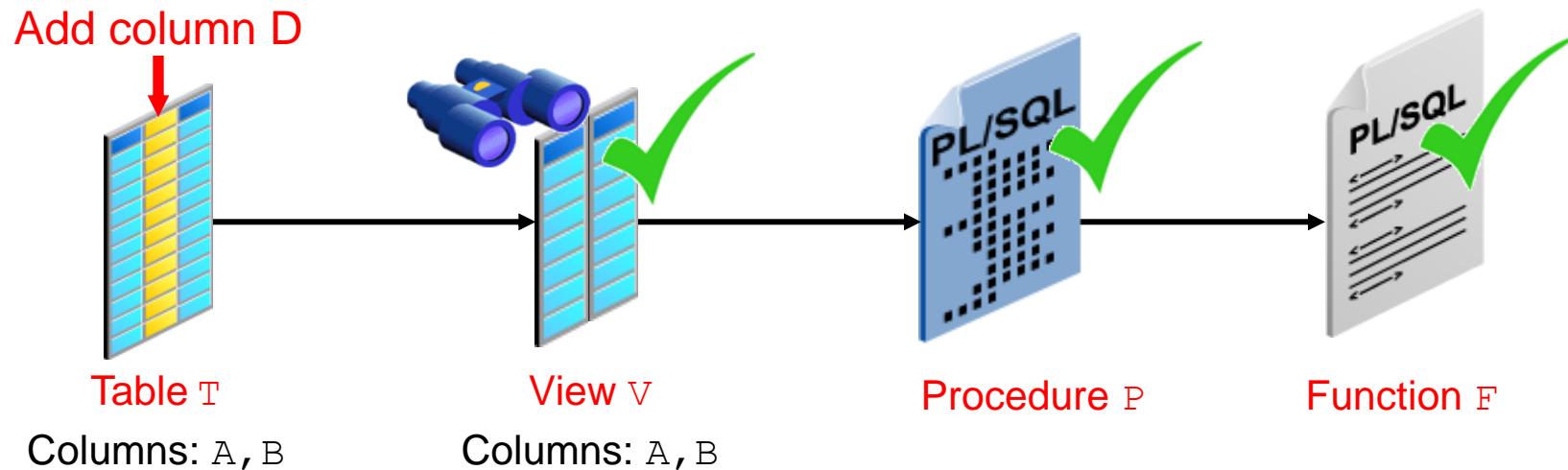


The screenshot shows a 'Query Result' window from Oracle SQL Developer. The title bar includes icons for Run, Stop, Refresh, and SQL, followed by 'All Rows Fetched: 17 in 0.338 seconds'. The main area displays a table with three columns: NESTED_LEVEL, TYPE, and NAME. The data is as follows:

NESTED_LEVEL	TYPE	NAME
1	0 TABLE	EMPLOYEES
2	1 VIEW	EMP_DETAILS_VIEW
3	1 TRIGGER	UPDATE_JOB_HISTORY
4	1 PROCEDURE	EMP_LIST
5	1 FUNCTION	GET_ANNUAL_COMP
6	1 PROCEDURE	ADD_EMPLOYEE
7	1 PACKAGE	EMP_PKG
8	2 PACKAGE BODY	EMP_PKG
...		

Metadatos de dependencia. Grado Fino

- Antes de 11g, la adición de la columna **D** a la tabla **T** invalidó los objetos dependientes.
- Oracle Database 11g registra una administración de dependencia adicional y de mayor precisión:
 - La adición de la **columna D a la tabla T** no afecta a la vista **V** y no invalida los objetos dependientes



Metadatos de dependencia. Grado Fino

- A partir de Oracle Database 11g, tiene acceso a registros que describen metadatos de dependencia más precisos.
- Esto se denomina dependencia de grano fino
 - y le permite a Oracle ver cuando los objetos dependientes no se invalidan sin necesidad lógica
 - En versiones anteriores, la adición de una columna implicaría la INVALIDACION de la vista
 - A partir de Oracle 11g se disponen de la **Dependencia de Grado Fina**, la cual, revisa la *Necesidad LOGICA de la Invalidación* de los objetos antes de producir la operación.

Gestión de dependencias de grano fino

- A partir de Oracle Database 11g, las dependencias se rastrean ahora en el *nivel de elemento dentro de la unidad*.
 - Es totalmente transparente para los desarrolladores y permiten una mejora de rendimiento transparente
 - La recompilación innecesaria sin duda consume CPU
- El seguimiento de dependencias de este tipo cubre los siguientes elementos:
 - Dependencia de una vista monotabla en su tabla base
 - Dependencia de una unidad de programa PL/SQL (cabecera del paquete, cuerpo del paquete o subprograma) en lo siguiente:
 - Otras unidades de programa PL/SQL
 - Tablas
 - Vistas



Gestión de dependencias de grano fino: Ejemplo1

Dependencia de una vista monotabla en su tabla base

```
CREATE TABLE t2 (col_a NUMBER, col_b NUMBER, col_c NUMBER);  
CREATE VIEW v AS SELECT col_a, col_b FROM t2;
```

```
SELECT ud.name, ud.type, ud.referenced_name,  
       ud.referenced_type, uo.status  
FROM user_dependencies ud, user_objects uo  
WHERE ud.name = uo.object_name AND ud.name = 'V';
```

La vista V
depende de
la tabla T

NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE	STATUS
V	VIEW	T2	TABLE	VALID

```
ALTER TABLE t2 ADD (col_d VARCHAR2(20));
```

Se agrega una
nueva columna
llamada COL_D

```
SELECT ud.name, ud.type, ud.referenced_name,  
       ud.referenced_type, uo.status  
FROM user_dependencies ud, user_objects uo  
WHERE ud.name = uo.object_name AND ud.name = 'V';
```

La vista V
depende de
la tabla T y es
VALID

NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE	STATUS
V	VIEW	T2	TABLE	VALID

Gestión de dependencias de grano fino: Ejemplo1

```
ALTER TABLE t2 MODIFY (col_a VARCHAR2(20));
SELECT ud.name, ud.referenced_name, ud.referenced_type,
       uo.status
FROM user_dependencies ud, user_objects uo
WHERE ud.name = uo.object_name AND ud.name = 'V';
```

Results:

	NAME	REFERENCED_NAME	REFERENCED_TYPE	STATUS
1	V	T2	TABLE	INVALID

En el ejemplo de la diapositiva, la vista se invalida porque su elemento (**COL_A**) se modifica en la tabla de la que depende la vista.

Gestión de dependencias de grano fino: Ejemplo2

```
CREATE OR REPLACE PACKAGE pkg IS
    PROCEDURE proc_1;
END pkg;
/
CREATE OR REPLACE PROCEDURE p IS
BEGIN
    pkg.proc_1();
END p;
/
CREATE OR REPLACE PACKAGE pkg
IS
    PROCEDURE proc_1;
    PROCEDURE unheard_of;
END pkg;
/
```

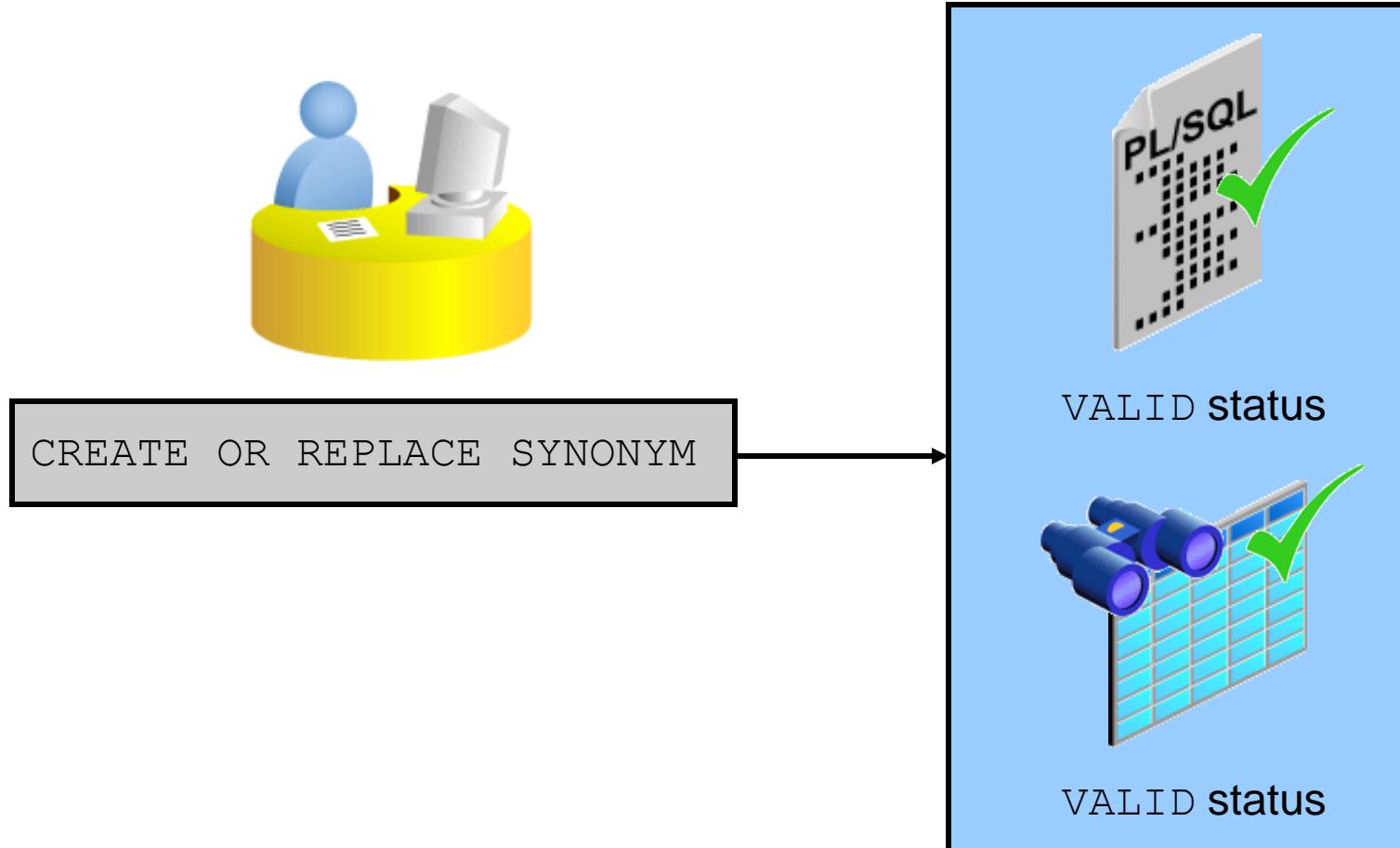
```
PACKAGE PKG compiled
PROCEDURE P compiled
PACKAGE PKG compiled
```

Se crea un paquete denominado **PKG** que tiene el procedimiento **PROC_1** declarado.
Un procedimiento llamado **P** invoca **PKG.PROC_1**.
La definición del paquete **PKG** se modifica y se agrega otra subrutina a la declaración del paquete.
El estado del procedimiento **P**, sigue siendo válido, pues el elemento que ha agregado no referencia a **proc_1**.

Cambios en las dependencias de Sinónimos

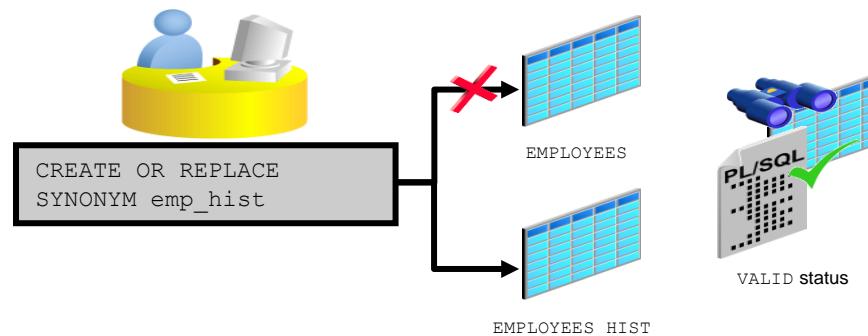
- Desde Oracle Database 10g, se ha mejorado el comando `CREATE OR REPLACE SYNONYM` para minimizar las invalidaciones de las unidades de programa PL/SQL dependientes en las referencias.
- No es necesario establecer ningún parámetro ni emitir ningún comando especial para habilitar esta funcionalidad;
- Las invalidaciones se minimizan automáticamente.
 - Sólo se aplican a sinónimos que se aplican a TABLAS

Cambios en las dependencias de Sinónimos



Mantenimiento de unidades de programa PL/SQL y vistas

- A partir de Oracle Database 10g Release 2, las **unidades de programa dependientes** de PL/SQL, **sinónimos** y **vistas** pueden cambiar y **no se invalidan** en las siguientes condiciones
 - El orden de las columnas, los nombres de columna y los tipos de datos de columna de las tablas son idénticos.
 - Los privilegios de la tabla recién referenciada y sus columnas son un superconjunto del conjunto de privilegios en la tabla original.
 - Los nombres y tipos de particiones y subparticiones son idénticos.
 - Las tablas son del mismo tipo de organización.
 - Las columnas de tipo de objeto son del mismo tipo.



Otro escenario de dependencias locales

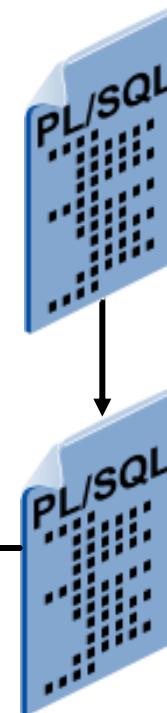
- Procedimiento `RAISE_SAL` actualiza la tabla `EMPLOYEES` directamente
- El procedimiento `REDUCE_SAL` actualiza indirectamente la tabla `EMPLOYEES` a través de `RAISE_SAL`
- Si se modifica la **lógica interna** del procedimiento `RAISE_SAL`, `REDUCE_SAL` volverá a compilar si `RAISE_SAL` ha compilado correctamente.
- Si se eliminan los **parámetros formales** para el procedimiento `RAISE_SAL`, `REDUCE_SAL` no volverá a compilar correctamente.

EMPLOYEES table

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
104	EFranz	IT_PROG	6000
...			

`REDUCE_SAL`
procedure

`RAISE_SAL`
procedure



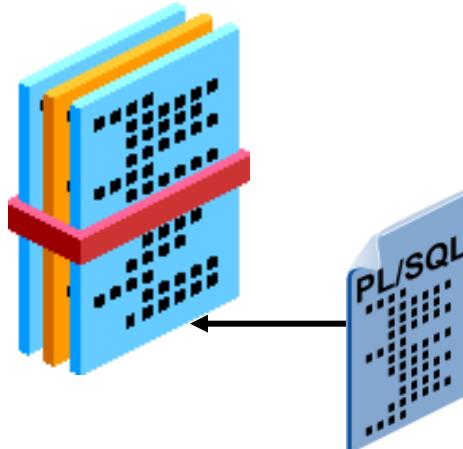
Cambio de Lógica - OK
Cambio parámetros - KO

Directrices para reducir la invalidación

Agregar nuevos elementos al final del paquete en la cabecera

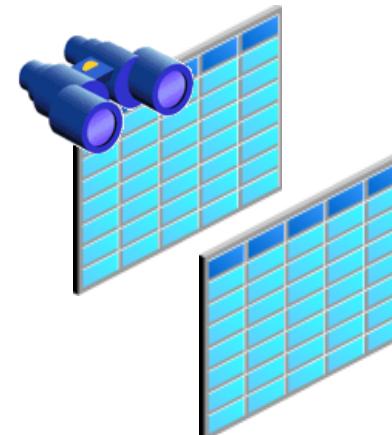
- Esto conserva los *slot numbers* and *entry-point* de los elementos superiores existentes en el paquete

```
CREATE OR REPLACE PACKAGE pkg1
IS
FUNCTION get_var RETURN
VARCHAR2;
PROCEDURE set_var (v VARCHAR2);
END;
```



Referencia de cada tabla a través de una vista

- Esto le permite hacer lo siguiente sin que se produzcan invalidaciones:
 - Agregue columnas a la tabla sin invalidar vistas dependientes
 - Modificar o eliminar columnas no referenciadas por la vista sin invalidar objetos dependientes



Volver a VALIDAR objetos (Revalidate)

- Un objeto que no es válido cuando se hace referencia debe ser validado antes de que pueda ser utilizado.
- La validación se produce automáticamente cuando se hace referencia a un objeto:
 - No requiere una acción explícita del usuario.
 - El compilador no puede revalidar automáticamente un objeto compilado con errores
- El compilador recompila el objeto:
 - Si compila sin errores, se vuelve VALIDO; De lo contrario, sigue siendo inválido.
- Si un objeto no es válido, su estado es COMPILED WITH ERRORS, UNAUTHORIZED, o INVALID.

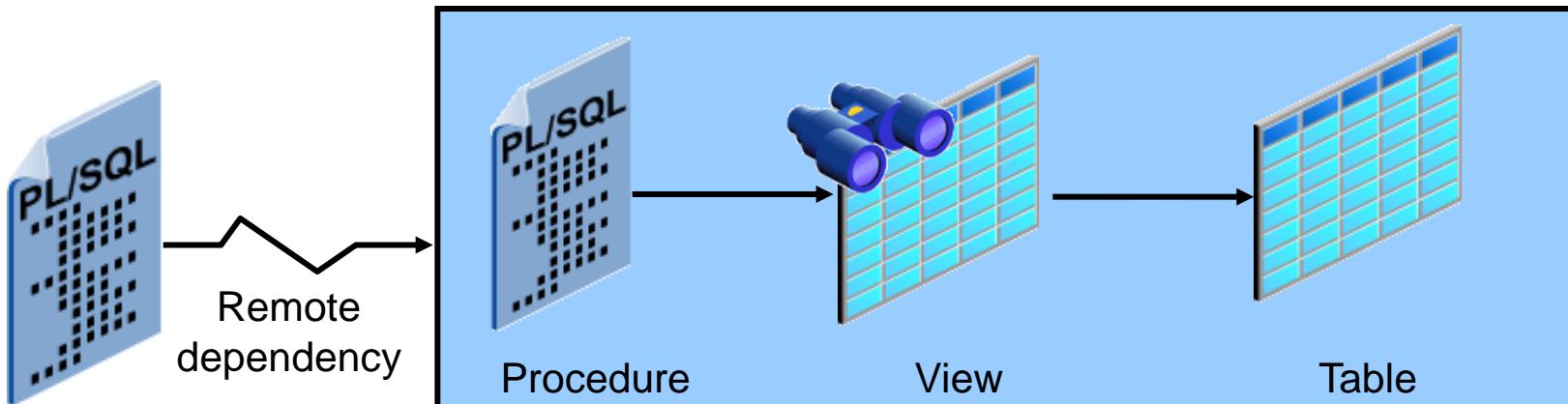
Dependencias remotas

- En el caso de las dependencias remotas, los objetos están en **nodos separados**.
- Oracle Database **no gestiona, en tiempo real**, las dependencias entre objetos de esquema remotos los elementos locales.
- Las dependencias entre los procedimientos almacenados (incluidas las funciones, los paquetes y los desencadenadores) en un sistema de base de datos distribuida se gestionan mediante la comprobación de:
 - **Marcas de tiempo** Tiempo modificado
 - **Firmas (prototipos)** Estructura de la Unidad PLSQL

Dependencias remotas

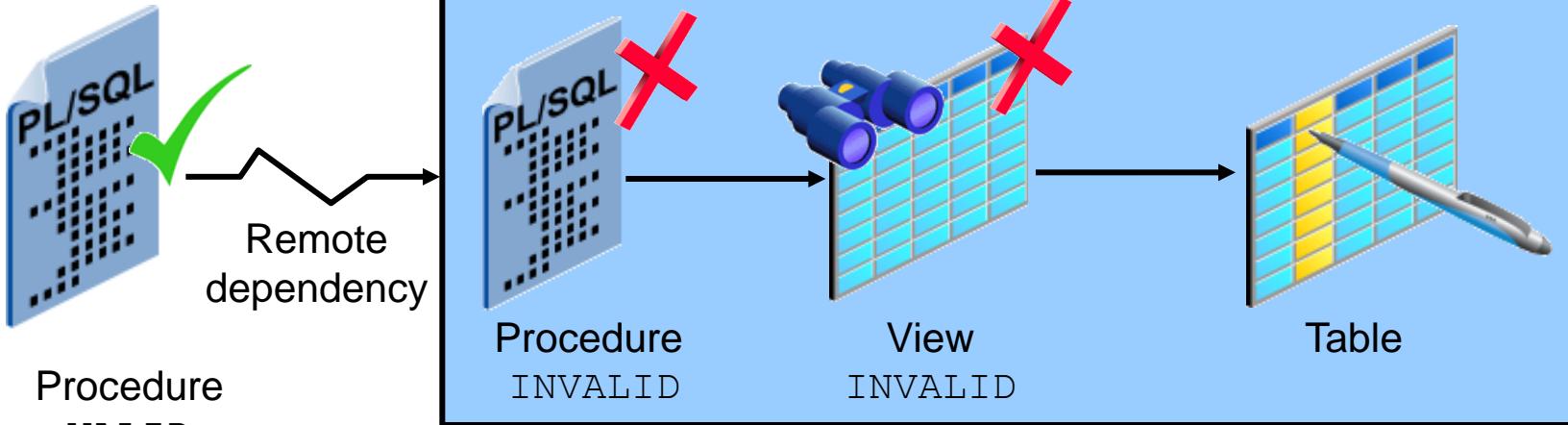
- Cuando el procedimiento almacenado local y todos sus objetos dependientes se llama por primera vez después de la modificación es cuando se invalidan.
- El problema aquí es;
 - Si cambia la **firma o marca de hora** del programa remoto no puede ver que el programa local está invalidado porque esperará a que se invalide la primera ejecución.
 - Si abre el sistema a los usuarios antes de corregirlo, puede enfrentarse a un problema serio de aplicación que provoca el tiempo de inactividad.
- En este caso, **la vista o el procedimiento deben modificarse manualmente para que no se devuelvan los errores.**
 - En tales casos, la falta de gestión de la dependencia es preferible realizar **recompilaciones explícitamente** de objetos dependientes.

Dependencias remotas



Procedure

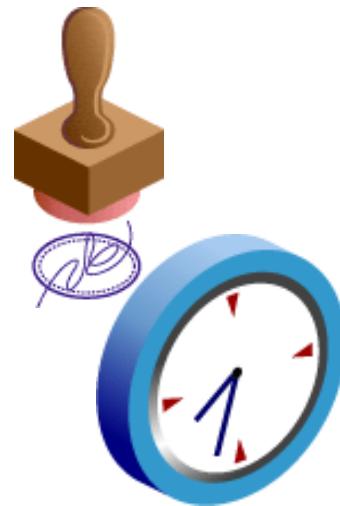
Local and remote references



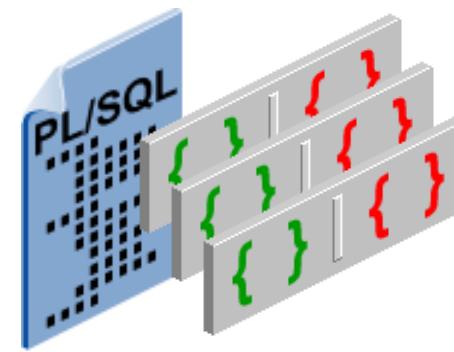
Procedure
VALID

Conceptos de dependencias remotas

Las dependencias remotas se rigen por el modo elegido por el usuario:



TIMESTAMP checking



SIGNATURE checking

- Cada unidad de programa PL/SQL lleva una marca de tiempo que se establece cuando se crea o se recompila
- La firma de una construcción PL/SQL (prototipo) contiene información sobre lo siguiente:
 - El nombre de la unida
 - Los tipos básicos de los parámetros
 - Los modos de los parámetros

Configuración del parámetro **REMOTE_DEPENDENCIES_MODE**

La utilización de **TIMESTAMP** o **SIGNATURE** dependen de la definición del parámetro **REMOTE_DEPENDENCIES_MODE**

- Como un parámetro init.ora:

```
REMOTE_DEPENDENCIES_MODE = TIMESTAMP | SIGNATURE
```

- A nivel de sistema:

```
ALTER SYSTEM SET REMOTE_DEPENDENCIES_MODE = TIMESTAMP | SIGNATURE
```

- En el nivel de la sesión:

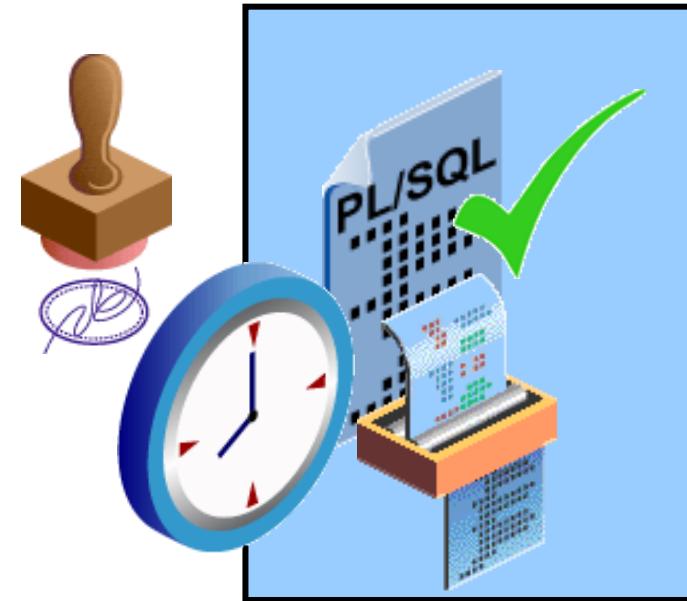
```
ALTER SESSION SET REMOTE_DEPENDENCIES_MODE = TIMESTAMP | SIGNATURE
```

Procedimiento remoto B

Compilado a las 8:00 AM

Procedimientos locales que se refieren a procedimientos remotos

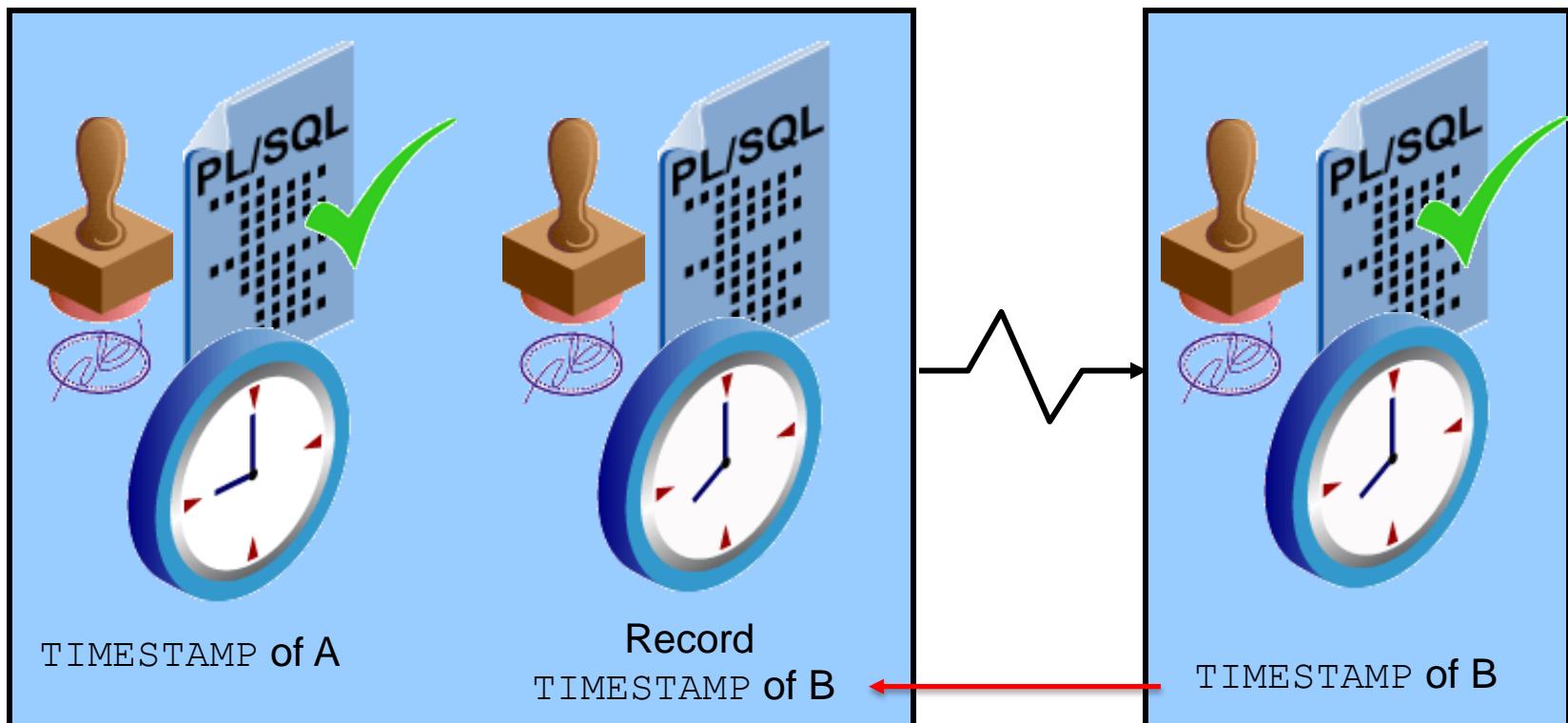
- Un procedimiento local que hace referencia a un procedimiento remoto es invalidado por el servidor Oracle si el procedimiento remoto se vuelve a compilar después de compilación del procedimiento local.
- Cuando se compila un procedimiento, el servidor de Oracle registra la **marca de tiempo** de esa compilación dentro del **código P** del procedimiento



Procedimiento remoto B:
Compila y es válido
A las 8:00 am

Procedimiento local A Compilado a las 9:00 AM

- Cuando se compila un **procedimiento local** que hace referencia a un **procedimiento remoto**, el servidor Oracle registra también la **marca de tiempo** del procedimiento remoto en el **código P** del procedimiento local.

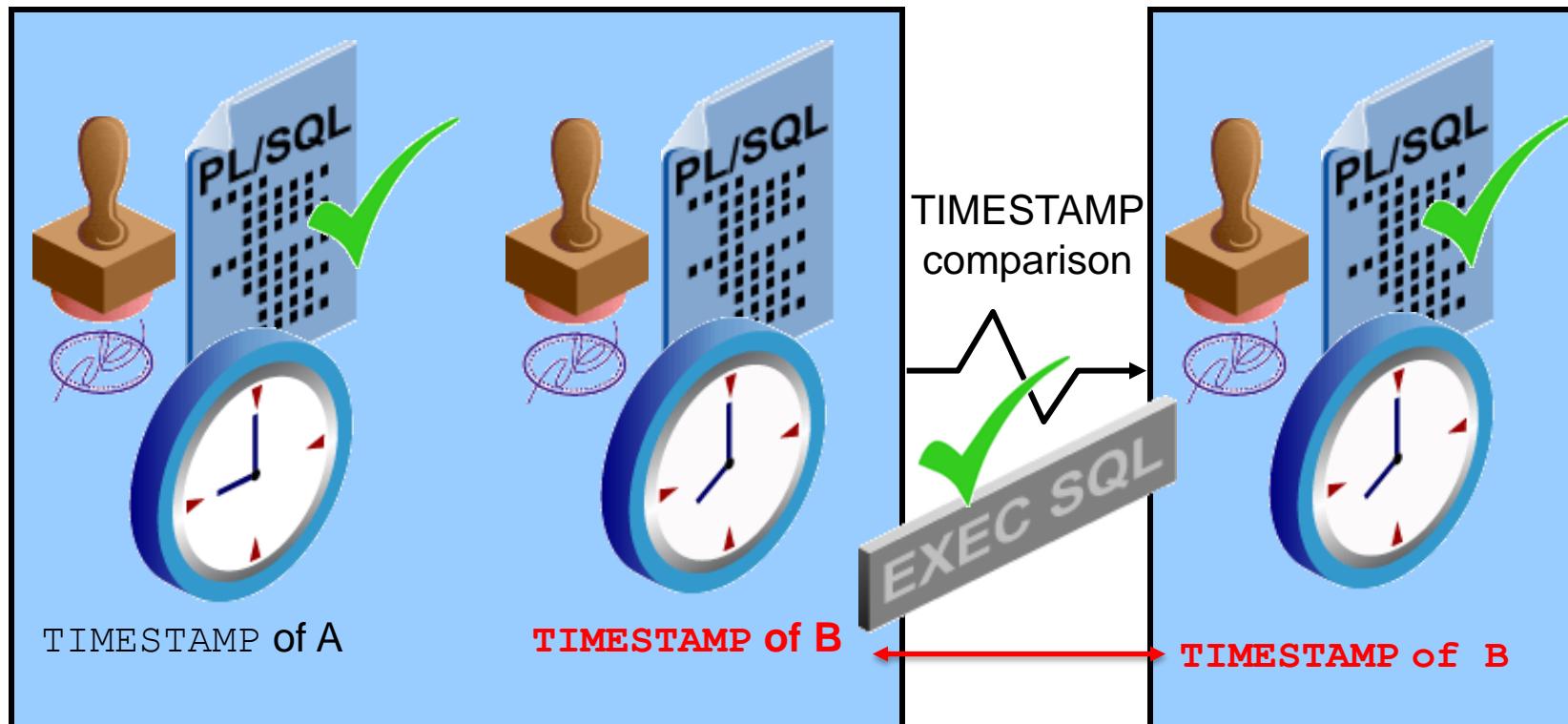


Local procedure A: VALID

Remote procedure B:
VALID

Ejecución del procedimiento A

- Cuando se invoca el procedimiento local en tiempo de ejecución, el servidor de Oracle **compara las dos marcas de tiempo del procedimiento remoto referenciado**.
- Si las **marcas de tiempo son iguales**, el procedimiento local se ejecuta



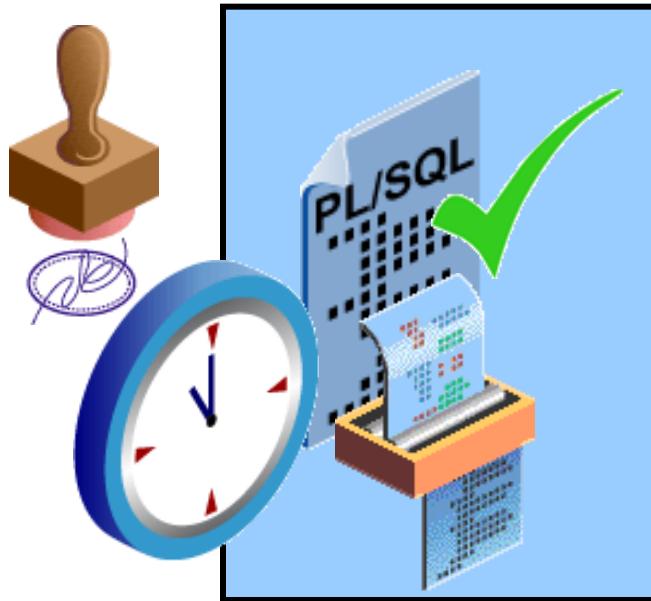
Local procedure A: VALID

Remote procedure B:
VALID

Procedimiento remoto B

Recompilado a las 11:00 AM

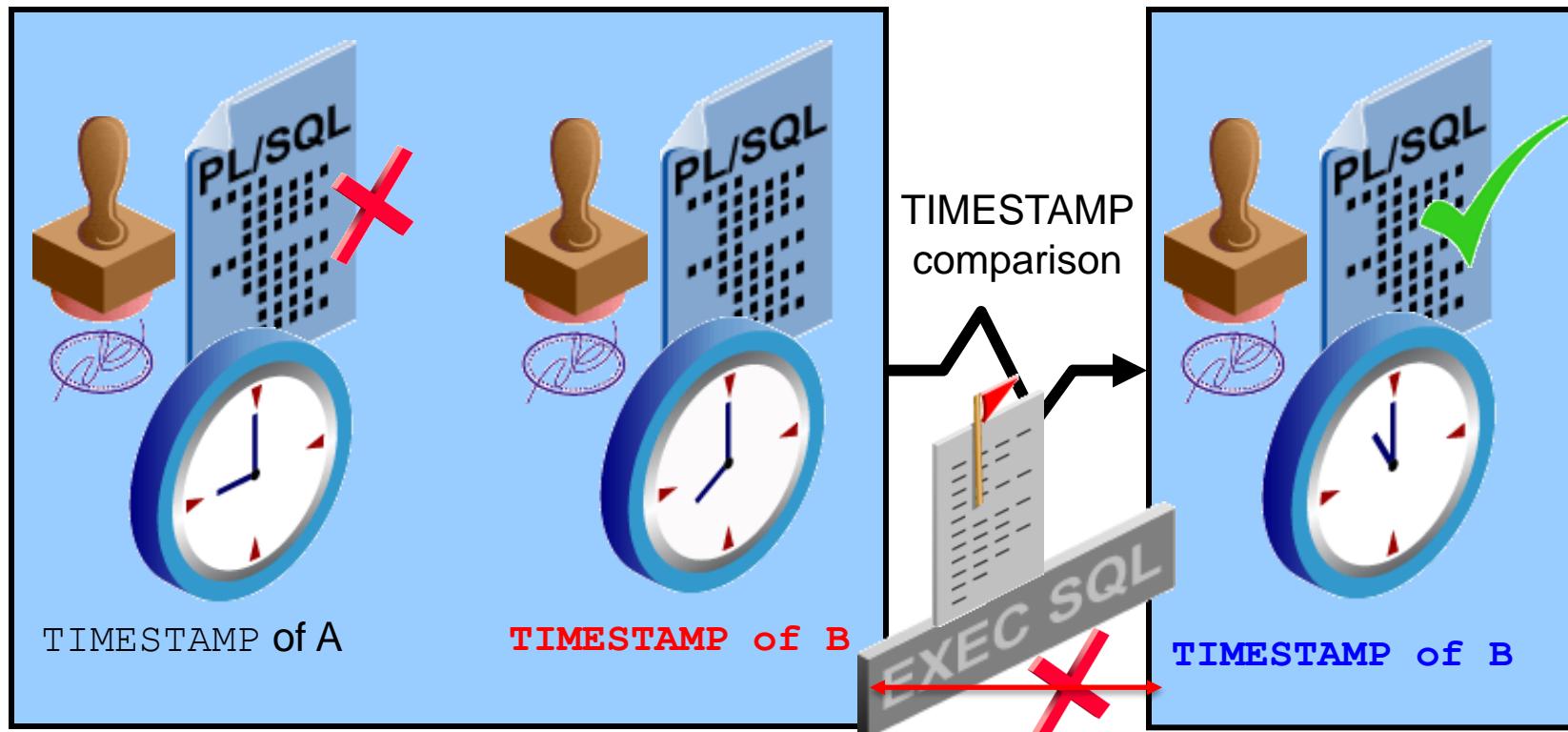
- El procedimiento remoto B se recompila con éxito a las 11:00 AM.
- **La nueva marca** de tiempo se graba junto con su **código P**.



Remote procedure B:
Recompiles and is VALID
at 11:00 AM

Ejecución del procedimiento A

- Si las **marcas de tiempo NO son iguales**, (lo que indica que el procedimiento remoto se ha vuelto a compilar), el servidor de Oracle **invalida** el procedimiento local y devuelve un **error en tiempo** de ejecución
- Si el procedimiento local se invoca una segunda vez, entonces el servidor Oracle lo recompila antes de ejecutar



Local procedure A: INVALID

Remote procedure B: VALID

Modo de firma o prototipo

- El modelo de firma, permite que el procedimiento remoto sea recompilado sin afectar los procedimientos locales.
 - Esto es importante si se distribuye la base de datos.
- La firma de un subprograma contiene la siguiente información:
 - El nombre del subprograma
 - Los tipos de datos de los parámetros
 - Los modos de los parámetros
 - El número de parámetros
 - El tipo de datos del valor devuelto para una función
- Si se cambia un programa remoto y se vuelve a compilar pero la firma no cambia, el procedimiento local puede ejecutar el procedimiento remoto.

Recompilación de una unidad de programa PL/SQL

- Si la recompilación tiene éxito, el objeto se convierte en válido.
- Si no, el servidor Oracle devuelve un error y el objeto sigue siendo inválido.
- Cuando recompila un objeto PL/SQL, el servidor Oracle primero recompila cualquier objeto no válido del que depende.
- La recompilación puede ser:
 - **Implicita**
 - Se maneja automáticamente a través de la compilación implícita en tiempo de ejecución
 - **Explícita**
 - Se maneja mediante recompilación explícita con la sentencia **ALTER**

Recompilación de una unidad de programa PL/SQL

```
ALTER PROCEDURE [SCHEMA.]procedure_name COMPILE;
```

```
ALTER FUNCTION [SCHEMA.]function_name COMPILE;
```

```
ALTER PACKAGE [SCHEMA.]package_name
    COMPILE [PACKAGE | SPECIFICATION | BODY];
```

```
ALTER TRIGGER trigger_name [COMPILE [DEBUG]] ;
```

Recompilación sin éxito

- A veces, una recompilación de procedimientos dependientes no tiene éxito
- La recompilación de procedimientos y funciones dependientes **no tiene éxito** cuando:
 - El objeto referenciado se **elimina** o se **cambia el nombre**
 - El **tipo de datos** de la columna referenciada se **cambia**
 - Se **borrar** la columna de referencia
 - Una vista referenciada es reemplazada por una **vista con diferentes columnas**
 - La **lista de parámetros** de un procedimiento referenciado se **modifica**

Recompilando procedimientos

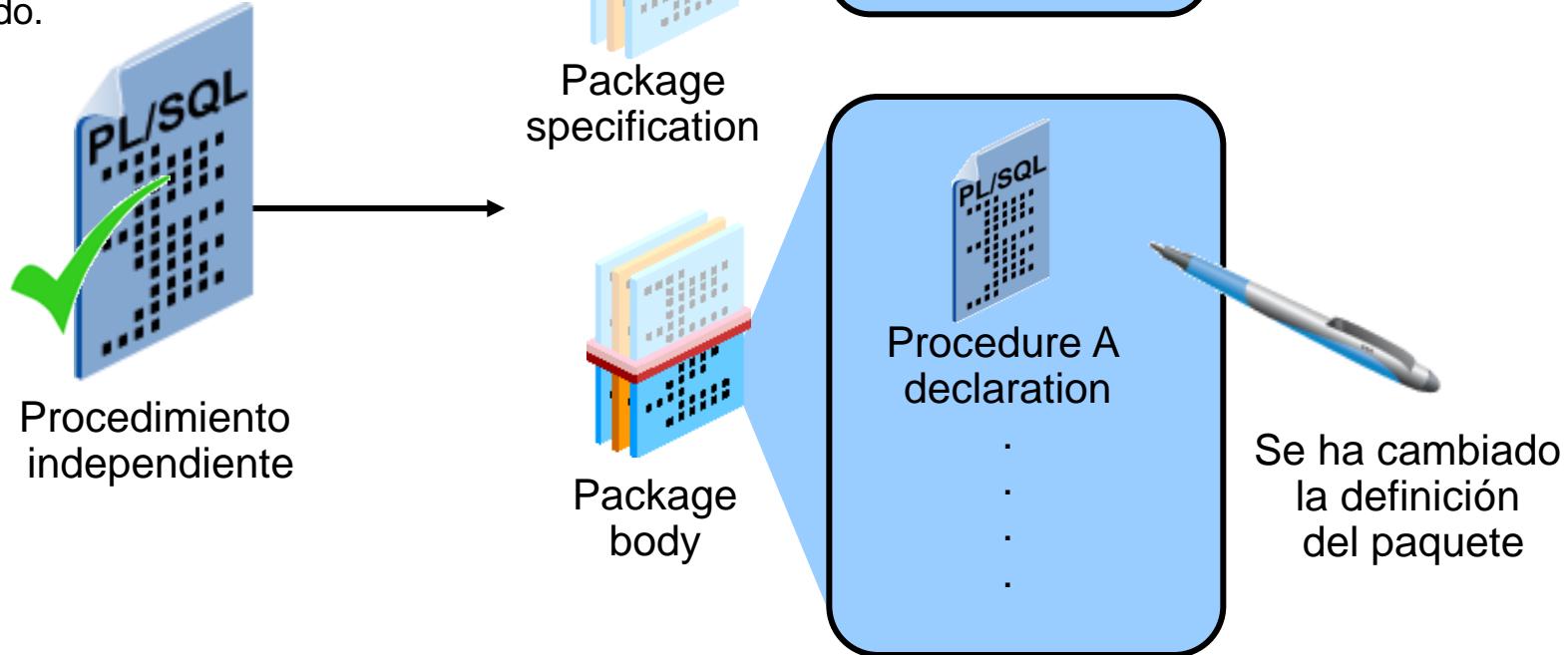
Puede minimizar el fracaso de recompilación siguiendo las directrices siguientes.

- Declaración de registros con el atributo %ROWTYPE
- Declaración de variables con el atributo %TYPE
- Consultando con la notación SELECT *
- Incluir una lista de columnas con instrucciones INSERT

Paquetes y dependencias: Referencias del subprograma el paquete

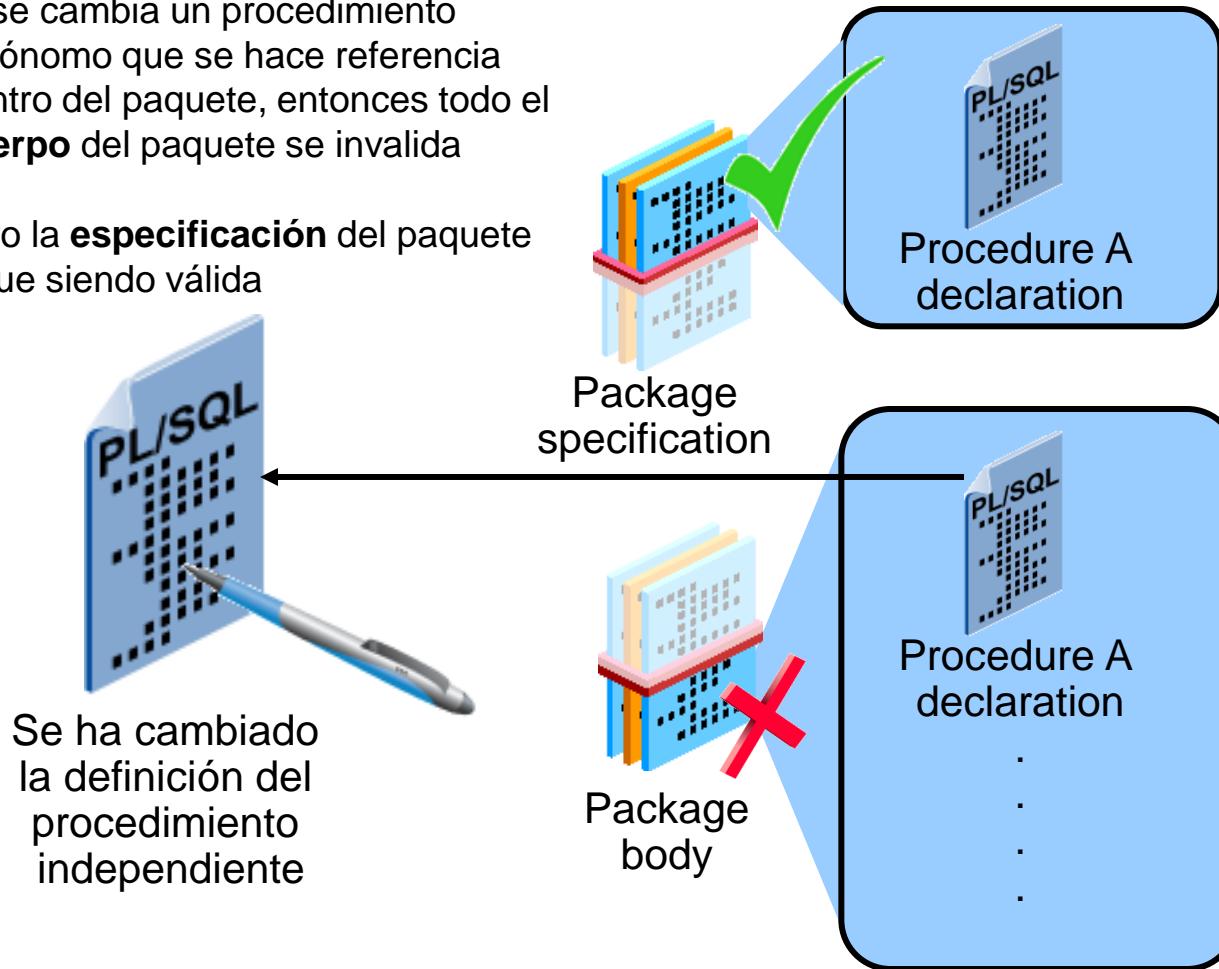
Utilice procedimientos independientes

- Si el cuerpo del paquete cambia y la especificación del paquete no cambia, entonces el procedimiento autónomo que hace referencia a una construcción de paquete sigue siendo válido.



Paquetes y dependencias: Referencias del subprograma el paquete

- Si se cambia un procedimiento autónomo que se hace referencia dentro del paquete, entonces todo el **cuerpo** del paquete se invalida
- pero la **especificación** del paquete sigue siendo válida



Quiz

Puede mostrar las dependencias directas e indirectas ejecutando el script `utldtree.sql`, rellenando la tabla `DEPTREE_TEMP TAB` con información para un objeto referenciado en particular y consultando las vistas `DEPTREE` o `I DEPTREE`.

- a. True
- b. False

Resumen

En esta lección, debes haber aprendido a:

- Seguimiento de dependencias procedimentales
- Predecir el efecto de cambiar un objeto de base de datos en procedimientos y funciones
- Administrar dependencias procedurales

Práctica 12

En esta lección, realiza las siguientes prácticas:

- Utilizar DEPTREE_FILL y IDEPTREE para ver las dependencias
- Recompilación de procedimientos, funciones y paquetes