9

Creación de Procedimientos

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Objetivos

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

- **Distinguir los bloques PL/SQL anónimos de los bloques PL/SQL denominados (subprogramas)**
- **Describir subprogramas**
- **Enumerar las ventajas de utilizar subprogramas**
- **Enumerar los diferentes entornos desde los cuales se pueden llamar a los subprogramas**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Finalidad de la Lección

PL/SQL admite muchas construcciones de programas diferentes. En esta lección aprenderá en qué se diferencian los bloques anónimos de los bloques PL/SQL denominados. A los bloques PL/SQL denominados también se les conoce como subprogramas o unidades de programa.

Objetivos

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

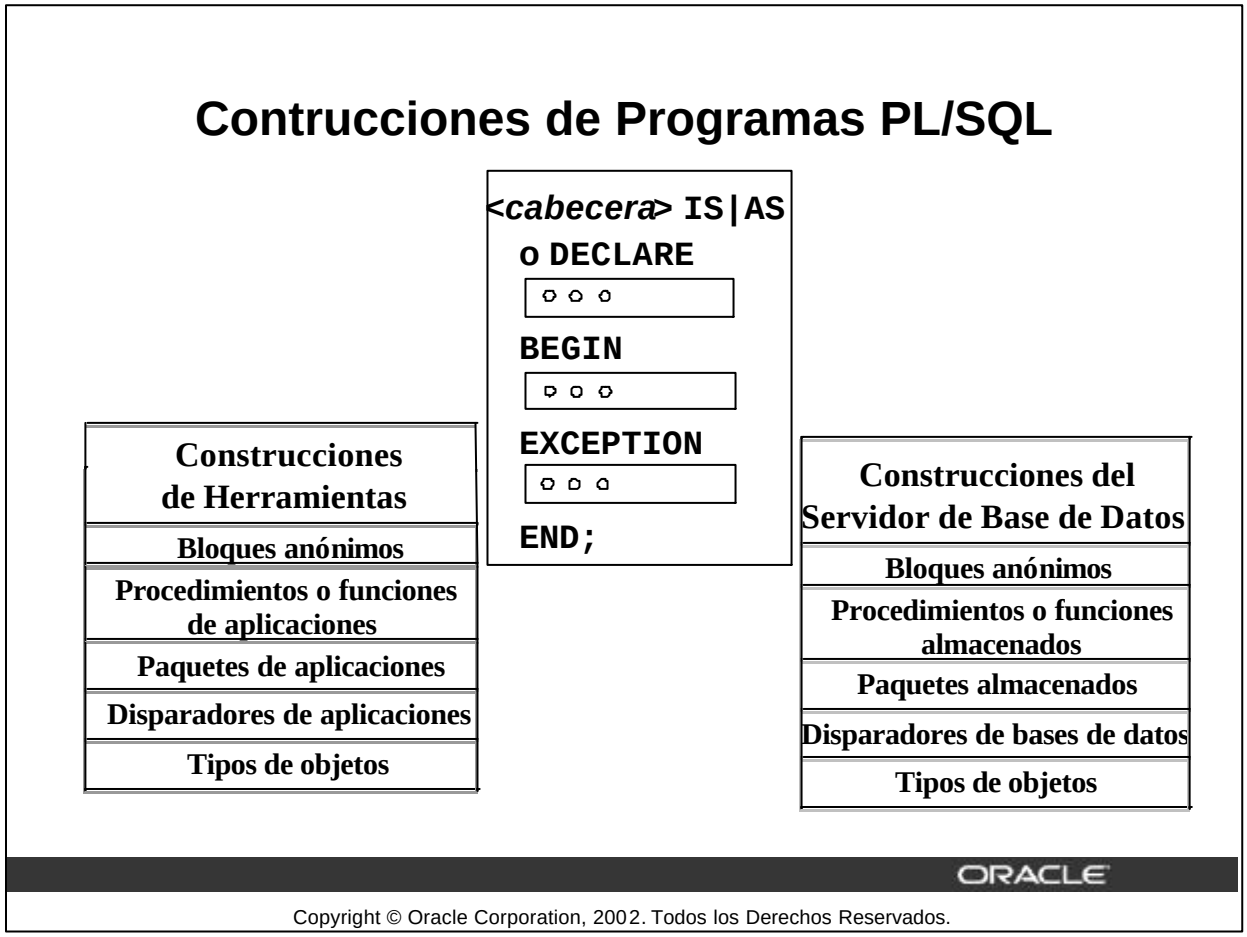
- **Describir bloques y subprogramas PL/SQL**
- **Describir los usos de los procedimientos**
- **Crear procedimientos**
- **Diferenciar los parámetros formales de los parámetros reales**
- **Enumerar las funciones de los diferentes modos de parámetros**
- **Crear procedimientos con parámetros**
- **Llamar a un procedimiento**
- **Manejar excepciones en procedimientos**
- **Eliminar un procedimiento**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Finalidad de la Lección

En esta lección aprenderá en qué se diferencian los bloques PL/SQL anónimos de los subprogramas. También aprenderá a crear, ejecutar y eliminar procedimientos.



Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Construcciones de Programas PL/SQL

El diagrama anterior describe una gran variedad de construcciones de programas PL/SQL diferentes que utilizan el bloque PL/SQL básico. En general, un bloque puede ser un bloque anónimo o un bloque denominado (un subprograma o una unidad de programa).

Estructura de un Bloque PL/SQL

Cada construcción PL/SQL se compone de uno o varios bloques. Estos bloques pueden estar completamente separados entre sí o estar anidados unos en otros. Por lo tanto, un bloque puede representar una pequeña parte de otro bloque que, a su vez, puede formar parte de una unidad de código completa.

Nota: En la transparencia, la palabra "o" situada antes de la palabra clave `DECLARE` no forma parte de la sintaxis. En el diagrama se utiliza únicamente para diferenciar el inicio de los subprogramas y los bloques anónimos.

En Oracle Server se pueden construir y utilizar bloques PL/SQL (unidades de programa PL/SQL almacenadas). También se pueden construir con las herramientas de Oracle Developer como, por ejemplo, Oracle Forms Developer, Oracle Report Developer, etc. (unidades de programa PL/SQL del cliente o aplicaciones).

Los tipos de objetos son tipos de dato compuestos y definidos por el usuario, que encapsulan una estructura de datos junto a las funciones y los procedimientos necesarios para manipular los datos. Se pueden crear tipos de objetos con Oracle Server o con las herramientas de Oracle Developer.

En este curso, aprenderá a escribir y gestionar funciones y procedimientos almacenados, disparadores de base de datos y paquetes. Sin embargo, no se explicará la creación de tipos de objeto.

Visión General de los Subprogramas

Un subprograma:

- **Es un bloque PL/SQL denominado que puede aceptar parámetros y que se puede llamar desde un entorno**
- **Existen dos tipos:**
  - **Un procedimiento que realiza una acción**
  - **Una función que calcula un valor**
- **Se basa en la estructura de bloques PL/SQL estándar**
- **Ofrece modularidad, capacidad de reutilización, extensibilidad y capacidad de mantenimiento**
- **Facilita el mantenimiento y mejora la seguridad, la integridad de los datos, el rendimiento y la claridad del código**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Visión General de los Subprogramas

Un subprograma se basa en la estructura PL/SQL estándar, que contiene una sección declarativa, una sección ejecutable y una sección de manejo de excepciones opcional.

Los subprogramas se pueden compilar y almacenar en la base de datos. Ofrecen modularidad, capacidad de reutilización, extensibilidad y capacidad de mantenimiento.

La modularización es el proceso mediante el cual los bloques de código de gran tamaño se dividen en grupos de código más pequeños llamados módulos. Una vez que el código se ha modularizado, los módulos pueden ser reutilizados por el mismo programa o se pueden compartir con otros programas. Es más fácil mantener y depurar el código de módulos pequeños que el de un solo programa de gran tamaño. Además, los módulos se pueden ampliar fácilmente para personalizarlos incorporando más funcionalidades, si es necesario, sin influir en los módulos restantes del programa.

Los subprogramas son fáciles de mantener, porque el código se encuentra en un solo lugar y, por lo tanto, cualquier modificación que haya que hacer en un subprograma, se puede realizar en esta única ubicación. Los subprogramas mejoran la seguridad y la integridad de los datos. Para acceder a los objetos de datos se utiliza el subprograma, y un usuario sólo puede llamar al subprograma si se le otorga el privilegio de acceso adecuado.

Estructura de los Bloques PL/SQL Anónimos

DECLARE (opcional)
Declare los objetos PL/SQL que se van a utilizar en este bloque

BEGIN (obligatorio)
Defina las sentencias ejecutables

EXCEPTION (opcional)
Defina las acciones que tendrán lugar si se produce un error o una excepción

END; (obligatorio)

ORACLE

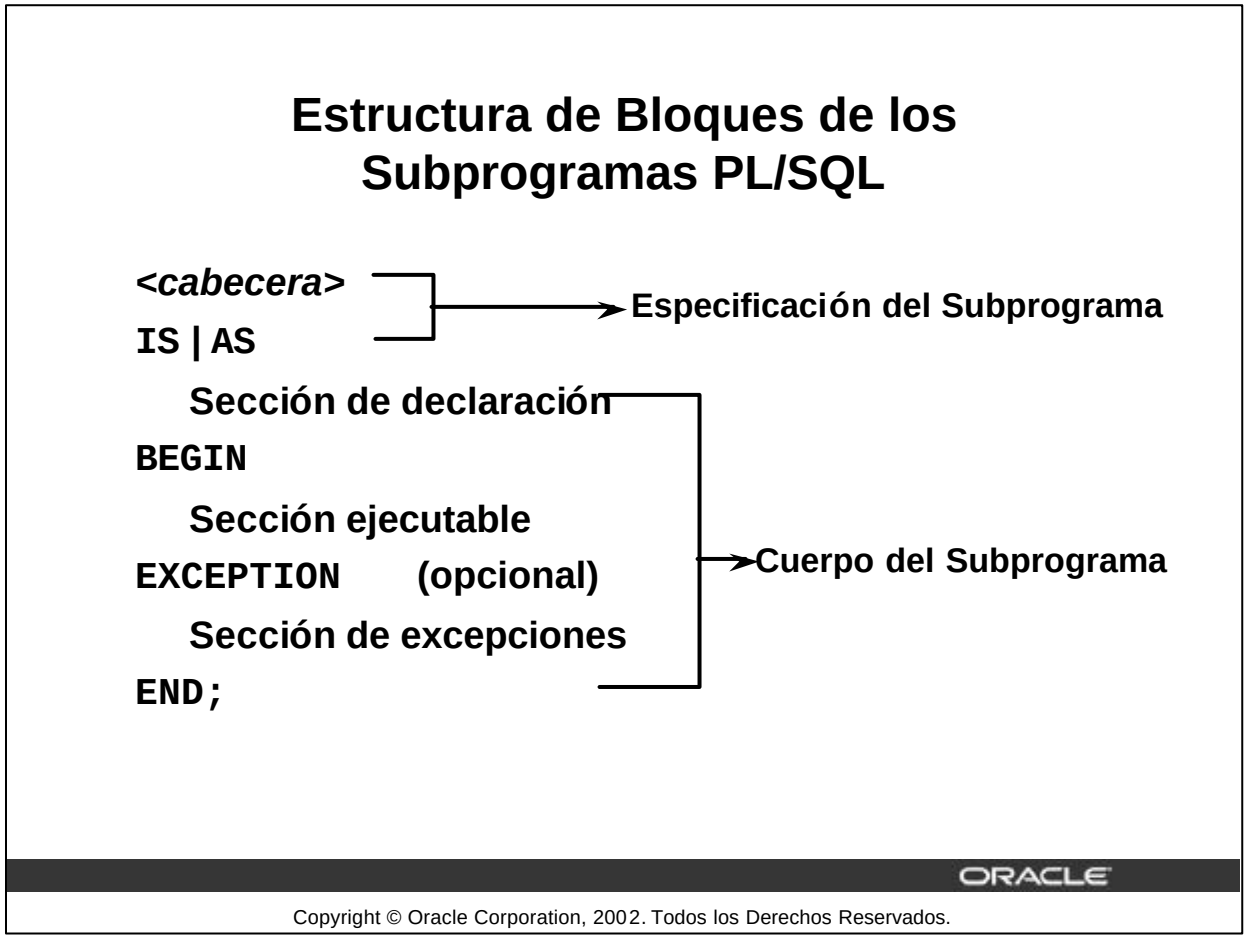
Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Bloques Anónimos

Los bloques anónimos no tienen nombre. Se declaran en el punto de aplicación en el que tienen que ser ejecutados y se envían al motor PL/SQL para que los ejecute en tiempo de ejecución.

- La sección situada entre las palabras clave `DECLARE` y `BEGIN` se denomina sección de declaración. En la sección de declaración se definen objetos PL/SQL como, por ejemplo, las variables, las constantes, los cursores y las excepciones definidas por el usuario a las que quiera hacer referencia en el interior del bloque. La palabra clave `DECLARE` es opcional si no se declara ningún objeto PL/SQL.
- Las palabras clave BEGIN y END son obligatorias y delimitan **el cuerpo de acciones que se** van a realizar. Esta sección se denomina sección ejecutable **del bloque.**
- La sección situada entre las palabras clave `EXCEPTION` y `END` se conoce como sección de excepciones. La sección de excepciones interrumpe las condiciones de error. En ella, se definen las acciones que hay que realizar si se produce la condición especificada. Esta sección es opcional.

Las palabras clave `DECLARE`, `BEGIN` y `EXCEPTION` no van seguidas de punto y coma (;), pero `END` y el resto de las sentencias PL/SQL sí que necesitan puntos y comas.



Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Subprogramas

Los subprogramas son bloques PL/SQL denominados que pueden aceptar parámetros y que se pueden llamar desde un entorno. PL/SQL posee dos tipos de subprogramas: *procedimientos* y *funciones*.

Especificación del Subprograma

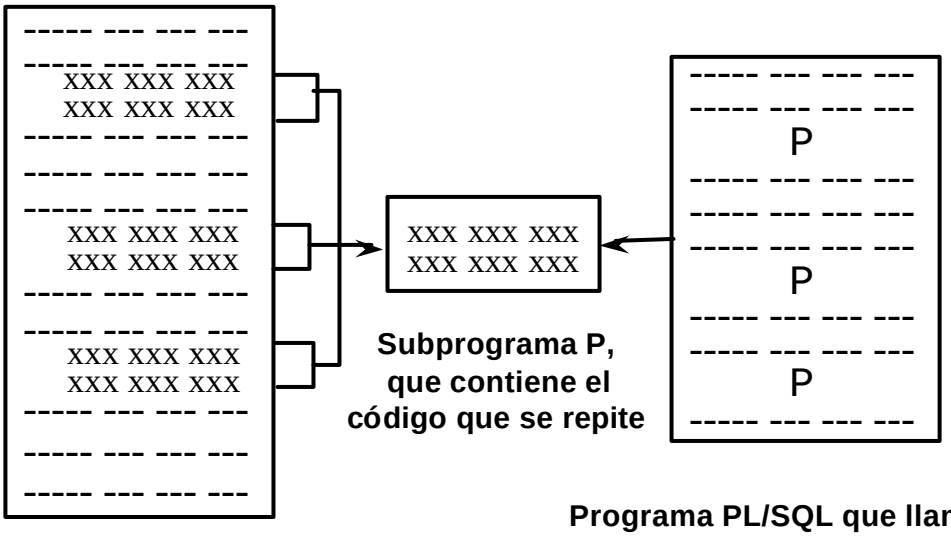
- La cabecera sólo es importante para los bloques denominados y determina la forma de llamar a la unidad de programa.

  La cabecera determina:
  - El tipo de subprograma PL/SQL, es decir, si es un procedimiento o una función
  - El nombre del subprograma
  - La lista de parámetros, si la hubiera
  - La cláusula `RETURN`, que sólo se aplica a las funciones
- La palabra clave `IS` o `AS` es obligatoria.

Cuerpo del Subprograma

- Es la sección de declaración del bloque situada entre `IS|AS` y `BEGIN`. Aquí no se utiliza la palabra clave `DECLARE`, que sirve para indicar el comienzo de la sección de declaración de un bloque anónimo.
- La sección ejecutable situada entre las palabras clave `BEGIN` y `END` es obligatoria y delimita el cuerpo de acciones que se van a realizar. En esta sección debe haber al menos una sentencia. Además, debería haber al menos una sentencia `NULL;` sentencia, que se considera ejecutable.
- La sección de excepciones situada entre `EXCEPTION` y `END` es opcional. Esta sección interrumpe las condiciones de error predefinidas. En ella, se definen las acciones que hay que realizar si se produce la condición de error especificada.

Subprogramas PL/SQL



Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Subprogramas

El diagrama de la transparencia explica cómo se puede sustituir por un subprograma una secuencia de sentencias PL/SQL que se repite en un bloque PL/SQL.

Si una secuencia de sentencias se repite más de una vez en un subprograma PL/SQL, puede crear un subprograma con el código repetido. Puede llamar al subprograma desde múltiples ubicaciones de un bloque PL/SQL. Después de crear y almacenar el subprograma en la base de datos, se le puede llamar todas las veces necesarias y desde múltiples aplicaciones.

Ventajas de los Subprogramas

- **Fácil mantenimiento**
- **Mayor integridad y seguridad de los datos**
- **Mejor rendimiento**
- **Mayor claridad del código**

ORACLE

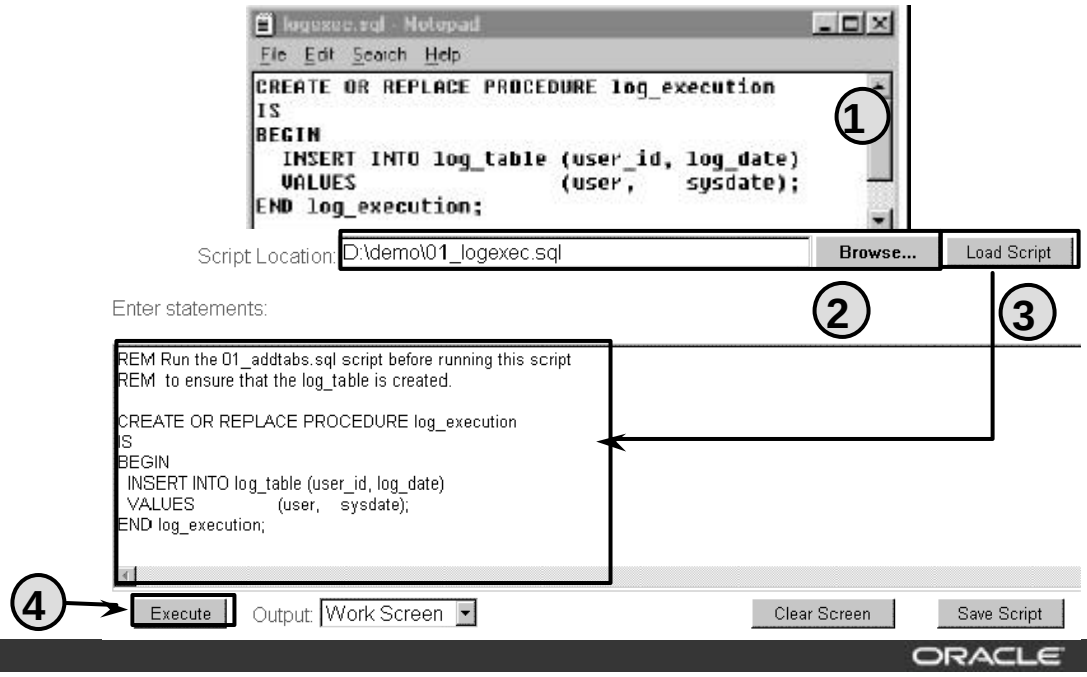
Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Ventajas de los Subprogramas

Las funciones y los procedimientos almacenados poseen muchas ventajas, además de permitir el desarrollo de aplicaciones modulares:

- Fácil mantenimiento
  - Modificación de las rutinas en línea sin interferir con otros usuarios
  - Modificación de una rutina que afecta a varias aplicaciones
  - Modificación de una rutina para no duplicar las pruebas
- Mayor integridad y seguridad de los datos
  - Control del acceso indirecto a los objetos de base de datos por parte de los usuarios sin privilegios utilizando privilegios de seguridad.
  - Garantía de que las acciones relacionadas se realizan juntas o no se realizan, al canalizar la actividad de las tablas relacionadas a través de una sola ruta de acceso
- Mejor rendimiento
  - Se evitan las repeticiones del análisis para múltiples usuarios utilizando el área de SQL compartido
  - Se evita el análisis de PL/SQL en tiempo de ejecución al realizarlo en tiempo de compilación
  - Se reduce el número de llamadas a la base de datos y se disminuye el tráfico de red agrupando comandos
- Mayor claridad del código: El uso de nombres de identificadores adecuados para describir la acción de las rutinas, reduce la necesidad de incluir comentarios y mejora la claridad del código.

Desarrollo de Subprogramas con *i*SQL*Plus



Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Desarrollo de Subprogramas con *i*SQL*Plus

*iSQL*Plus es una interfaz con SQL*Plus apta para Internet. Puede utilizar un navegador Web para conectarse a una base de datos de Oracle y realizar las mismas acciones que a través de otras interfaces de SQL*Plus.*

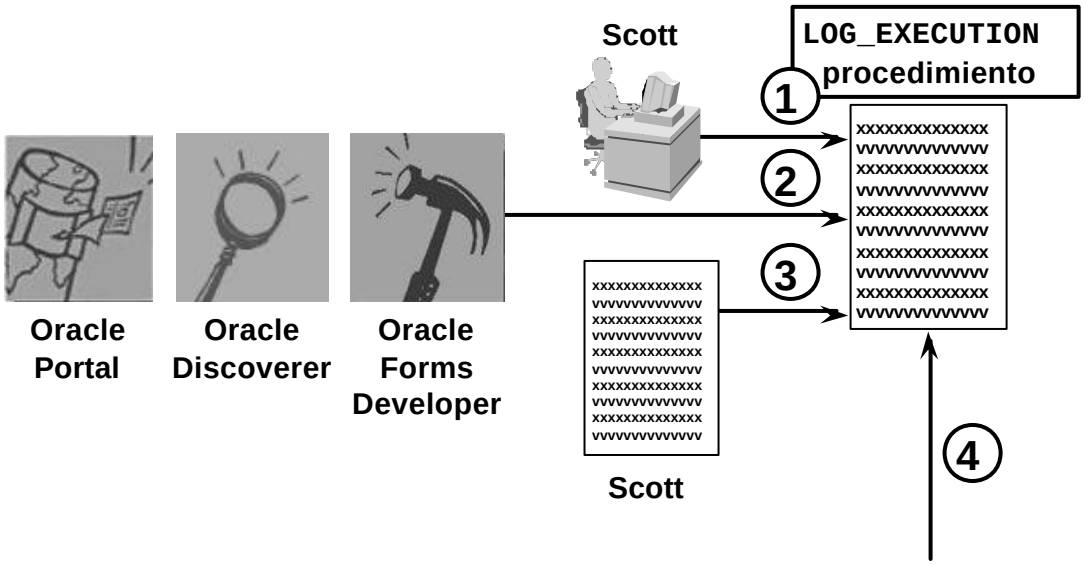
1. Utilice un editor de texto para crear un archivo de comandos SQL que defina el subprograma. El ejemplo de la transparencia crea el procedimiento almacenado `LOG_EXECUTION` sin ningún parámetro. El procedimiento registra el usuario y la fecha actual en una tabla de base de datos llamada `LOG_TABLE`.

Desde la ventana del navegador de *i*SQL*Plus:

2. Haga clic en el botón Browse para localizar el archivo de comandos SQL.
3. Haga clic en el botón Load Script para cargar el archivo de comandos en el buffer de *i*SQL*Plus.
4. Haga clic en el botón Execute para ejecutar el código. Por defecto, el resultado del código aparece en la pantalla.

También se pueden crear subprogramas PL/SQL utilizando las herramientas de desarrollo de Oracle como, por ejemplo, Oracle Forms Developer.

Llamada a Funciones y Procedimientos Almacenados



ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Cómo Llamar a Funciones y Procedimientos Almacenados

Se puede llamar a un procedimiento o a una función que se haya creado previamente desde una gran variedad de entornos como *i*SQL*Plus, Oracle Forms Developer, Oracle Discoverer, Oracle Portal, otro procedimiento almacenado y muchas otras herramientas de Oracle y aplicaciones de precompilador. La siguiente tabla explica cómo se puede llamar a un procedimiento que se ha creado previamente, `log_execution`, desde diversos entornos.

*i*SQL*Plus	`EXECUTE log_execution`
Herramientas de desarrollo de Oracle como, por ejemplo, Oracle Forms Developer	`log_execution;`

Otro procedimiento	`CREATE OR REPLACE PROCEDURE leave_emp`
	`(p_id IN employees.employee_id%TYPE)`
	`IS`
	`BEGIN`
	`DELETE FROM employees`
	`WHERE employee_id = p_id;`
	`log_execution;`
	`END`
	`leave_emp;`

¿Qué es un procedimiento?

- **Un procedimiento es un tipo de subprograma que realiza una acción.**
- **Los procedimientos se pueden almacenar en la base de datos, como objetos de esquema, para repetir su ejecución.**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Definición de un Procedimiento

Un procedimiento es un bloque PL/SQL denominado que puede aceptar parámetros (que en ocasiones se denominan argumentos) y que se puede llamar. En general, los procedimientos sirven para realizar una acción. Los procedimientos poseen una cabecera, una sección de declaración, una sección ejecutable y una sección de manejo de excepciones opcional.

Los procedimientos se pueden compilar y almacenar en la base de datos como objetos de esquema.

Los procedimientos mejoran la capacidad de reutilización y de mantenimiento. Una vez validados, se pueden utilizar en cualquier número de aplicaciones. Si los requisitos cambian, sólo hay que actualizar el procedimiento.

Sintaxis de los Procedimientos

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter1 [mode1] datatype1,
  parameter2 [mode2] datatype2,
  . . .)]
IS|AS
PL/SQL Block;
```

- **La opción REPLACE indica que, si el procedimiento existe, se borrará y se sustituirá por la nueva versión que ha creado la sentencia.**
- **El bloque PL/SQL comienza con BEGIN o con la declaración de variables locales, y termina con END o con END *procedure_name.***

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Sintaxis de los Procedimientos

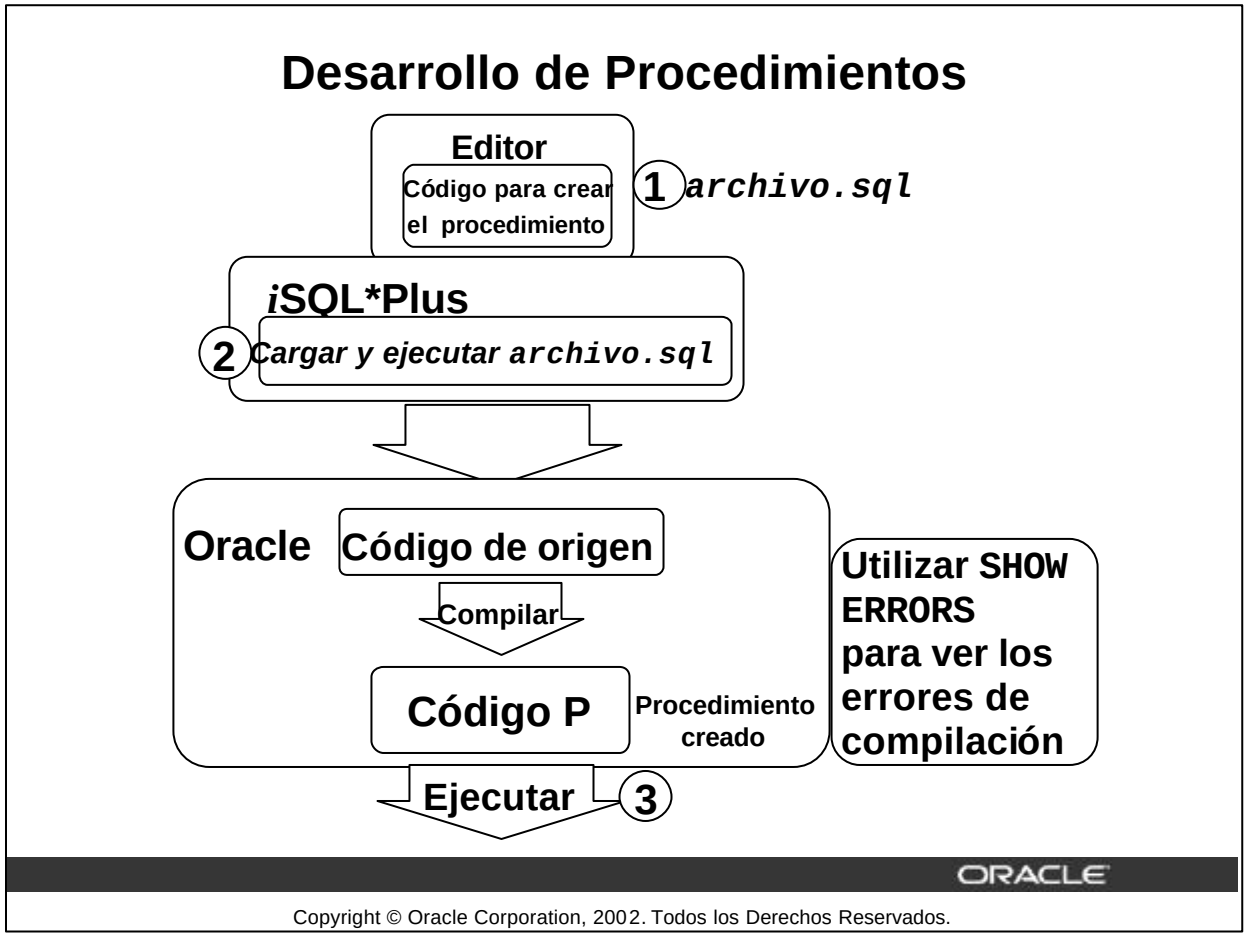
Definiciones de la Sintaxis

Parámetro	Descripción
nombre_procedimiento	Nombre del procedimiento
parámetro	Nombre de una variable PL/SQL cuyo valor se rellena en el entorno de llamada o se transfiere a él, o ambas cosas, dependiendo del *modo* que se utilice
modo	Tipo de argumento: IN (por defecto) OUT IN OUT
Tipo de dato	Tipo de dato del argumento; puede ser cualquier tipo de dato SQL / PL/SQL. Puede ser el tipo de dato TYPE, %ROWTYPE, o cualquier tipo de dato escalar o compuesto.
Bloque PL/SQL	Cuerpo procedural que define la acción que realiza el procedimiento

Sintaxis de los Procedimientos (continuación)

Para crear nuevos procedimientos, hay que utilizar la sentencia `CREATE PROCEDURE`, que puede declarar una lista de parámetros, y debe definir las acciones que va a realizar el bloque PL/SQL estándar. La cláusula `CREATE` le permite crear procedimientos autónomos, que se almacenan en una base de datos de Oracle.

- Los bloques PL/SQL comienzan con `BEGIN` o con la declaración de variables locales, y terminan con `END` o con `END` *`procedure_name`*. No puede hacer referencia a variables ligadas o del host en un bloque PL/SQL de un procedimiento almacenado.
- La opción `REPLACE` indica que, si el procedimiento existe, se borrará y se sustituirá por la nueva versión que ha creado la sentencia.
- No se puede limitar el tamaño del tipo de dato en los parámetros.



Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Desarrollo de Procedimientos

Estos son los principales pasos para desarrollar un procedimiento almacenado. En las próximas dos páginas encontrará más detalles acerca de la creación de procedimientos.

1. Escriba la sintaxis: Introduzca el código para crear un procedimiento (sentencia `CREATE PROCEDURE`) en un editor del sistema o en un procesador de textos y guárdelo como un archivo de comandos SQL (con extensión `.sql`).
2. Compile el código: Utilice *i*SQL*Plus para cargar y ejecutar el archivo de comandos SQL. El código de origen se compila en el *código P* y se crea el procedimiento.

   Los archivos de comandos con la sentencia `CREATE PROCEDURE` (o `CREATE OR REPLACE PROCEDURE`) permiten cambiar la sentencia en el caso de que se produzca algún error de compilación o de tiempo de ejecución, o realizar cambios posteriores en la sentencia. No se puede llamar a un procedimiento que contiene un error de compilación o de tiempo de ejecución. En *i*SQL*Plus, utilice `SHOW ERRORS` para ver los errores de compilación. Si ejecuta la sentencia `CREATE PROCEDURE`, el código de origen se almacenará en el diccionario de datos, aunque el procedimiento contenga errores de compilación. Solucione los errores del código utilizando el editor y recompile el código.
3. Ejecute el procedimiento para realizar la acción deseada. Después de compilar el código de origen y de haber creado el procedimiento, dicho procedimiento se puede ejecutar todas las veces necesarias utilizando el comando `EXECUTE` desde *i*SQL*Plus. El compilador PL/SQL genera el *pseudo código* o el código P basándose en el código analizado. El motor PL/SQL lo ejecuta cuando se llama al procedimiento.

Nota: Si hay algún error de compilación y se realizan cambios posteriores en la sentencia `CREATE PROCEDURE`, primero debe utilizar `DROP` en el procedimiento o la sintaxis `OR REPLACE`.

Puede crear procedimientos del cliente que se utilicen en aplicaciones del cliente con herramientas como, por ejemplo, el IDE (entorno de desarrollo integrado) de Oracle Forms y Reports. Consulte el Apéndice F para saber cómo se pueden crear subprogramas del cliente utilizando la herramienta Oracle Procedure Builder.

Parámetros Formales y Parámetros Reales

- **Parámetros formales: variables declaradas en la lista de parámetros de la especificación de un subprograma**

  Ejemplo:

```
CREATE PROCEDURE raise_sal(p_id NUMBER, p_amount
NUMBER)
...
END raise_sal;
```

- **Parámetros reales: variables o expresiones a las que se hace referencia en la lista de parámetros de la llamada de un subprograma**

  Ejemplo:

```
raise_sal(v_id, 2000)
```

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Parámetros Formales y Parámetros Reales

Los parámetros formales son variables declaradas en la lista de parámetros de la especificación de un subprograma. Por ejemplo, en el procedimiento `RAISE_SAL`, las variables `P_ID` y `P_AMOUNT` son parámetros formales.

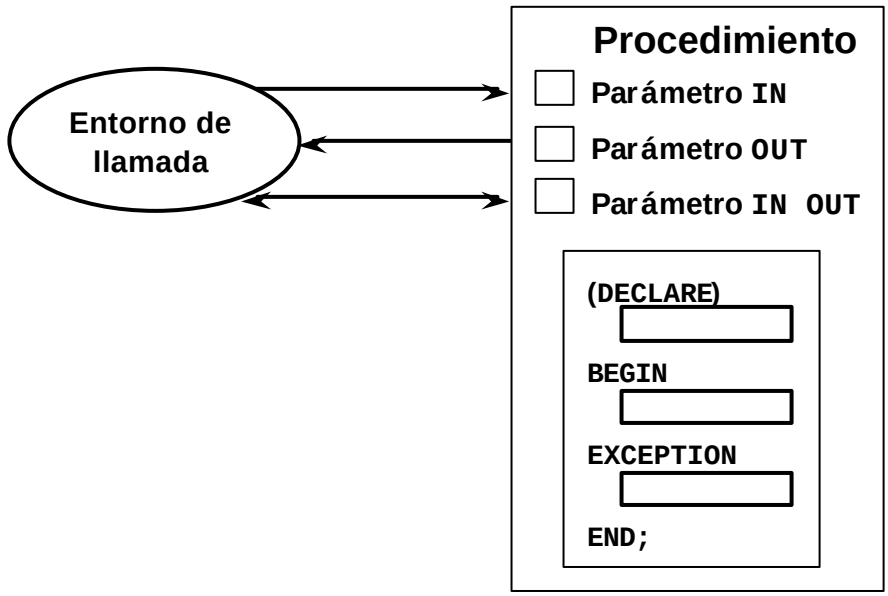
Los parámetros reales son variables o expresiones a las que se hace referencia en la lista de parámetros de la llamada de un subprograma. Por ejemplo, en la llamada `raise_sal(v_id , 2000)` al procedimiento `RAISE_SAL`, las variables `V_ID` y `2000` son parámetros reales.

- Los parámetros reales se evalúan y los resultados se asignan a parámetros formales durante la llamada del subprograma.
- Los parámetros reales también pueden ser expresiones como, por ejemplo:

```
raise_sal(v_id, raise+100);
```

- Es conveniente utilizar diferentes nombres para los parámetros formales y los reales. En este curso, los parámetros formales tienen el prefijo `p_`.
- Los parámetros formales y reales deberían tener tipos de datos compatibles. Si es necesario, antes de asignar el valor, PL/SQL convierte el tipo de dato del valor del parámetro real en el del parámetro formal.

Modos de Parámetros Procedurales



ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Modos de Parámetros Procedurales

Se pueden transferir valores hasta y desde el entorno de llamada por medio de parámetros. Hay que seleccionar uno de los tres modos para cada parámetro: `IN`, `OUT` o `IN OUT`.

Cualquier intento de cambiar el valor de un parámetro `IN`, producirá un error.

Nota: `DATATYPE` sólo puede ser la definición `%TYPE`, la definición `%ROWTYPE` o un tipo de dato explícito sin especificación de tamaño.

Tipo de Parámetro	Descripción
`IN` (por defecto)	Transfiere un valor constante desde el entorno de llamada al procedimiento
`OUT`	Transfiere un valor desde el procedimiento al entorno de llamada
`IN OUT`	Transfiere un valor desde el entorno de llamada al procedimiento y un valor posiblemente diferente desde el procedimiento al entorno de llamada utilizando el mismo parámetro

Creación de Procedimientos con Parámetros

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

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Creación de Procedimientos con Parámetros

Cuando se crea el procedimiento, el parámetro formal define el valor utilizado en la sección ejecutable del bloque PL/SQL, mientras que se hace referencia al parámetro real al llamar al procedimiento.

El modo de parámetro `IN` es el modo por defecto. Es decir, si no se especifica ningún modo con un parámetro, se considera que el parámetro es `IN`. Los modos `OUT` e `IN OUT` se deben especificar explícitamente delante de dichos parámetros.

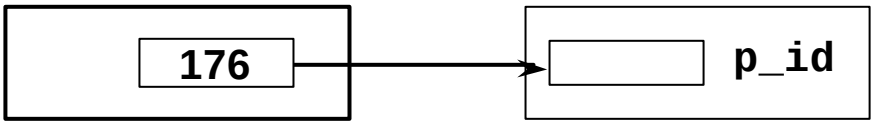
A un parámetro formal `IN` no se le puede asignar un valor. Es decir, los parámetros `IN` no se pueden modificar en el cuerpo del procedimiento.

Hay que asignar un valor a los parámetros `OUT` o `IN OUT` antes de que regresen al entorno de llamada.

A los parámetros `IN` se les puede asignar un valor por defecto en la lista de parámetros. A los parámetros `OUT` e `IN OUT` no se les pueden asignar valores por defecto.

Por defecto, los parámetros `IN` se pasan por referencia y los parámetros `OUT` y `IN OUT` por valor. Para mejorar el rendimiento de los parámetros `OUT` e `IN OUT`, se puede utilizar el indicador del compilador `NOCOPY` para solicitar un paso por referencia. El uso de `NOCOPY` se explica en profundidad en el curso *Advanced PL/SQL*.

Parámetros IN: Ejemplo




```
CREATE OR REPLACE PROCEDURE raise_salary
  (p_id IN employees.employee_id%TYPE)
IS
BEGIN
  UPDATE employees
  SET    salary = salary * 1.10
  WHERE   employee_id = p_id;
END raise_salary;
/
```

```
Procedure created.
```

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Parámetros IN: Ejemplo

El ejemplo de la transparencia muestra un procedimiento con un parámetro `IN`. Si se ejecuta esta sentencia en *i*SQL*Plus, se creará el procedimiento `RAISE_SALARY`. Una vez llamado, `RAISE_SALARY` acepta el parámetro para el identificador del empleado y actualiza el registro de dicho empleado con un aumento de sueldo del 10 por ciento.

Para llamar a un procedimiento en *i*SQL*Plus, hay que utilizar el comando `EXECUTE`.

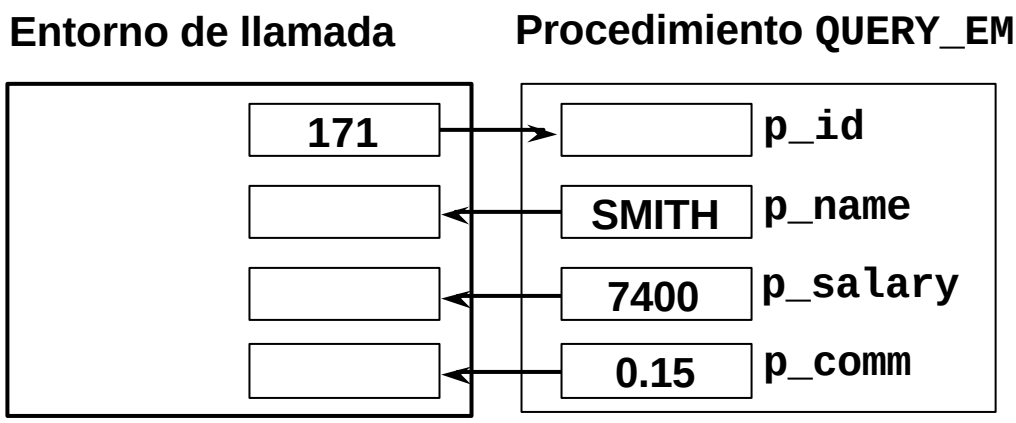
```
EXECUTE raise_salary (176)
```

Para llamar a un procedimiento desde otro procedimiento, hay que utilizar una llamada directa. Introduzca el nombre del procedimiento y los parámetros reales en el lugar de llamada del nuevo procedimiento.

```
raise_salary (176);
```

Los parámetros `IN` se transfieren como constantes desde el entorno de llamada al procedimiento. Cualquier intento de cambiar el valor de un parámetro `IN`, producirá un error.

Parámetros OUT: Ejemplo



ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Parámetros OUT: Ejemplo

En este ejemplo, se crea un procedimiento con parámetros OUT para recuperar información acerca de un empleado. El procedimiento acepta el valor 171 para el identificador de empleado y recupera en los tres parámetros de salida el nombre, el sueldo y el porcentaje de comisión del empleado con el identificador 171. En la siguiente transparencia se muestra el código para crear el procedimiento QUERY_EMP.

Parámetros OUT: Ejemplo

emp_query.sql

```
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,
   p_comm    OUT  employees.commission_pct%TYPE)
IS
BEGIN
  SELECT   last_name, salary, commission_pct
   INTO    p_name, p_salary, p_comm
   FROM    employees
   WHERE   employee_id = p_id;
END query_emp;
/
```

Procedure created.

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Parámetros OUT: Ejemplo (continuación)

Ejecute el archivo de comandos que se muestra en la transparencia para crear el procedimiento QUERY_EMP. Este procedimiento posee cuatro parámetros formales. Tres de ellos son parámetros OUT que devuelven valores al entorno de llamada.

El procedimiento acepta un valor EMPLOYEE_ID para el parámetro P_ID. Los valores del nombre, el sueldo y el porcentaje de comisión correspondientes al identificador del empleado se recuperan en tres parámetros OUT cuyos valores se devuelven al entorno de llamada.

Observe que el nombre del archivo de comandos no tiene que ser necesariamente igual que el del procedimiento. (El archivo de comandos se encuentra en el cliente y el procedimiento se almacena en el esquema de base de datos).

Visualización de Parámetros OUT

- Cargue y ejecute el archivo de comandos `emp_query.sql` para crear el procedimiento `QUERY_EMP`.
- Declare variables del host, ejecute el procedimiento `QUERY_EMP` e imprima el valor de la variable global `G_NAME`.

```
VARIABLE g_name     VARCHAR2(25)
VARIABLE g_sal      NUMBER
VARIABLE g_comm     NUMBER

EXECUTE query_emp(171, :g_name, :g_sal, :g_comm)

PRINT g_name
```

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

G_NAME
Smith

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Cómo Ver el Valor de los Parámetros OUT con *i*SQL*Plus

1. Ejecute el archivo de comandos SQL para generar y compilar el código de origen.
2. Cree variables del host en *i*SQL*Plus utilizando el comando `VARIABLE`.
3. Llame al procedimiento `QUERY_EMP`, proporcionando estas variables del host como parámetros `OUT`. Tenga en cuenta el uso de los dos puntos (:) para hacer referencia a las variables del host en el comando `EXECUTE`.
4. Para ver los valores que se transfieren desde el procedimiento al entorno de llamada, utilice el comando PRINT.

El ejemplo de la transparencia muestra cómo se ha transferido el valor de la variable `G_NAME` al entorno de llamada. El resto de las variables se pueden ver individualmente, como en el ejemplo anterior, o con un solo comando `PRINT`.

```
PRINT g_name g_sal g_comm
```

No especifique un tamaño para la variable del host de tipo de dato `NUMBER` cuando utilice el comando `VARIABLE`. Las variables del host de tipo de dato `CHAR` o `VARCHAR2` tienen una longitud de uno por defecto, a menos que se proporcione un valor entre paréntesis.

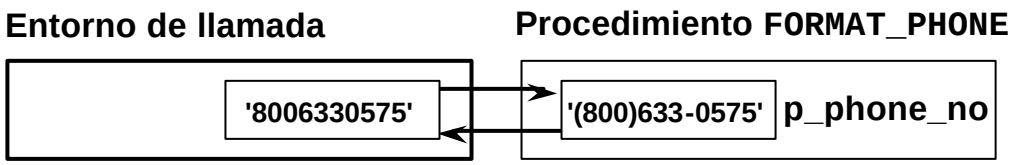
`PRINT` y `VARIABLE` son comandos de *i*SQL*Plus.

Nota: Si se transfiere una constante o una expresión como un parámetro real a la variable `OUT`, se producirán errores de compilación. Por ejemplo:

```
EXECUTE query_emp(171, :g_name, raise+100, :g_comm)
```

produce un error de compilación.

Parámetros IN OUT




```
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;
/
```

Procedure created.

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Uso de Parámetros IN OUT

Con los parámetros IN OUT, puede transferir valores a un procedimiento y devolver un valor al entorno de llamada. El valor que se devuelve es el original, un valor que no ha cambiado o un nuevo valor definido en el interior del procedimiento.

Los parámetros IN OUT funcionan como variables inicializadas.

Ejemplo

Cree un procedimiento con un parámetro IN OUT que acepte una cadena de caracteres de 10 dígitos y devuelva un número de teléfono con el formato (800) 633-0575.

Ejecute la sentencia para crear el procedimiento FORMAT_PHONE.

Visualización de Parámetros IN OUT

```
VARIABLE g_phone_no VARCHAR2(15)
BEGIN
  :g_phone_no := '8006330575';
END;
/
PRINT g_phone_no
EXECUTE format_phone (:g_phone_no)
PRINT g_phone_no
```

PL/SQL procedure successfully completed.

G_PHONE_NO
8006330575

PL/SQL procedure successfully completed.

G_PHONE_NO
(800)633-0575

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Cómo Ver los Parámetros IN OUT con *i*SQL*Plus

1. Cree una variable del host utilizando el comando VARIABLE.
2. Rellene la variable del host con un valor, utilizando un bloque PL/SQL anónimo.
3. Llame al procedimiento FORMAT_PHONE, proporcionando las variables del host como parámetros IN OUT. Tenga en cuenta el uso de los dos puntos (:) para hacer referencia a la variable del host en el comando EXECUTE.
4. Para ver el valor que se ha transferido al entorno de llamada, utilice el comando PRINT.

Métodos para Transferir Parámetros

- **Posicional: Enumera los parámetros reales en el mismo orden que los parámetros formales.**
- **Denominado: Enumera los parámetros reales en un orden arbitrario asociando cada uno a su parámetro formal correspondiente.**
- **Combinación: Enumera algunos de los parámetros reales como el método posicional y algunos como el método denominado.**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Métodos para Transferir Parámetros

En los procedimientos que contienen múltiples parámetros, se pueden utilizar una serie de métodos para especificar los valores de los parámetros.

Método	Descripción
Posicional	Enumera los valores en el orden en el que se declaran los parámetros
Asociación con nombre	Enumera los valores en un orden arbitrario asociando cada uno de ellos a su nombre de parámetro utilizando una sintaxis especial (=>)
Combinación	Enumera los primeros valores con el método posicional y el resto, utilizando la sintaxis especial del método con nombre

Opción DEFAULT de los Parámetros

```
CREATE OR REPLACE PROCEDURE add_dept
  (p_name  IN departments.department_name%TYPE
                          DEFAULT 'unknown',
   p_loc   IN 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;
/
```

```
Procedure created.
```

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Ejemplo de Valores por Defecto de los Parámetros

Se pueden inicializar parámetros `IN` con valores por defecto. De esa manera, se pueden transferir diferentes números de parámetros reales a un subprograma, aceptando o sustituyendo a su gusto los valores por defecto. Asimismo, puede agregar nuevos parámetros formales sin tener que cambiar cada llamada al subprograma.

Ejecute la sentencia de la transparencia para crear el procedimiento `ADD_DEPT`. Observe el uso de la cláusula `DEFAULT` en la declaración del parámetro formal.

Sólo puede asignar valores por defecto a los parámetros del modo `IN`. No se permite asignar valores por defecto a los parámetros `OUT` e `IN OUT`. Si se transfieren valores por defecto a estos tipos de parámetros, se producirá el siguiente error de compilación.

```
PLS-00230: OUT and IN OUT formal parameters may not have default
expressions
```

Si no se transfiere un parámetro real, se utiliza el valor por defecto de su parámetro formal correspondiente. Observe las llamadas al procedimiento anterior que se ilustran en la siguiente página.

Ejemplos de Transferencias de Parámetros

```
BEGIN
  add_dept;
  add_dept ('TRAINING', 2500);
  add_dept ( p_loc => 2400, p_name =>'EDUCATION');
  add_dept ( p_loc => 1200) ;
END;
/
SELECT department_id, department_name, location_id
FROM departments;
```

PL/SQL procedure successfully completed.

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
30	Purchasing	1700
40	Human Resources	2400
290	TRAINING	2500
300	EDUCATION	2400
310	unknown	1200

31 rows selected.

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Ejemplo de Valores por Defecto de los Parámetros (continuación)

El bloque anónimo anterior muestra las diferentes formas de llamar al procedimiento ADD_DEPT y el resultado de cada una de esas formas que se utiliza para llamar al procedimiento.

Normalmente, se puede utilizar la notación posicional para sustituir los valores por defecto de los parámetros formales. Sin embargo, no se puede ignorar un parámetro formal excluyendo su parámetro real.

Nota: Todos los parámetros posicionales deberían estar situados antes de los parámetros denominados en una llamada del subprograma. De lo contrario, recibirá un mensaje de error, tal y como se muestra en el siguiente ejemplo:

```
BEGIN add_dept(p_name=>'new dept', 'new location'); END;
                       *
ERROR at line 1:
ORA-06550: line 1, column 36:
PLS-00312: a positional parameter association may not follow a named association
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
```

Declaración de Subprogramas

leave_emp2.sql

```
CREATE OR REPLACE PROCEDURE leave_emp2
  (p_id  IN  employees.employee_id%TYPE)
IS
  PROCEDURE log_exec
  IS
  BEGIN
    INSERT INTO log_table (user_id, log_date)
    VALUES (USER, SYSDATE);
  END log_exec;
BEGIN
  DELETE FROM employees
  WHERE   employee_id = p_id;
  log_exec;
END leave_emp2;
/
```

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Declaración de Subprogramas

Puede declarar subprogramas en cualquier bloque PL/SQL. Esta es una alternativa a la creación del procedimiento autónomo `LOG_EXEC`. Los subprogramas que se declaran de esta manera se denominan subprogramas locales (o módulos locales). Dado que están definidos en la sección de declaración de otro programa, el ámbito de los subprogramas locales se limita al bloque principal (delimitador) en el que están definidos. Esto significa que no se puede llamar a los subprogramas locales desde el exterior del bloque en el que están declarados. La declaración de subprogramas locales mejora la claridad del código, al asignar los identificadores de regla de negocio adecuados a los bloques de código.

Nota: Debe declarar el subprograma en la sección de declaración del bloque y debe ser el último elemento de dicha sección, después del resto de los elementos del programa. Por ejemplo, si declarara una variable después del final del subprograma, pero antes del `BEGIN` del procedimiento, se produciría un error de compilación.

Si hay que acceder al código por varias aplicaciones, coloque el subprograma en un paquete o cree un subprograma autónomo con el código. Los paquetes se explicarán más adelante, en este mismo curso.

Llamada a un Procedimiento desde un Bloque PL/SQL Anónimo

```
DECLARE
  v_id NUMBER := 163;
BEGIN
  raise_salary(v_id);   --llama al procedimiento
  COMMIT;
...
END;
```

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Llamada a un Procedimiento desde un Bloque PL/SQL Anónimo

Llame al procedimiento `RAISE_SALARY` desde un bloque PL/SQL anónimo, tal y como se muestra en la transparencia.

Los procedimientos se pueden llamar desde *cualquier* herramienta o lenguaje que sea compatible con PL/SQL.

Ya ha visto cómo se llama a un procedimiento independiente desde *i*SQL*Plus.

Llamada a un Procedimiento desde Otro Procedimiento

process_emps.sql

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

ORACLE

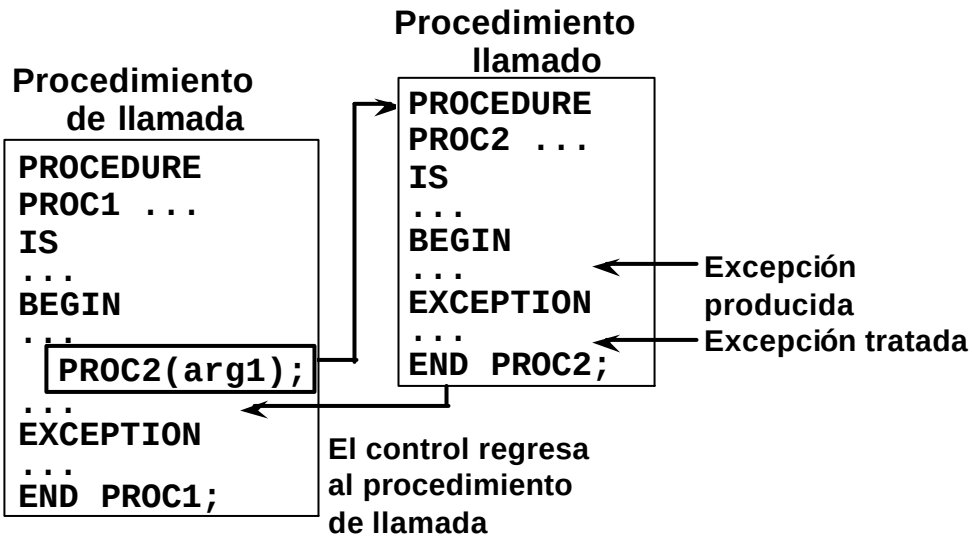
Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Llamada a un Procedimiento desde Otro Procedimiento

Este ejemplo muestra cómo se llama a un procedimiento desde otro procedimiento almacenado. El procedimiento almacenado PROCESS_EMPS utiliza un cursor para procesar todos los registros de la tabla EMPLOYEES y transfiere el identificador de cada empleado al procedimiento RAISE_SALARY, lo que produce un aumento de sueldo del 10 por ciento en toda la empresa.

.

Excepciones Tratadas



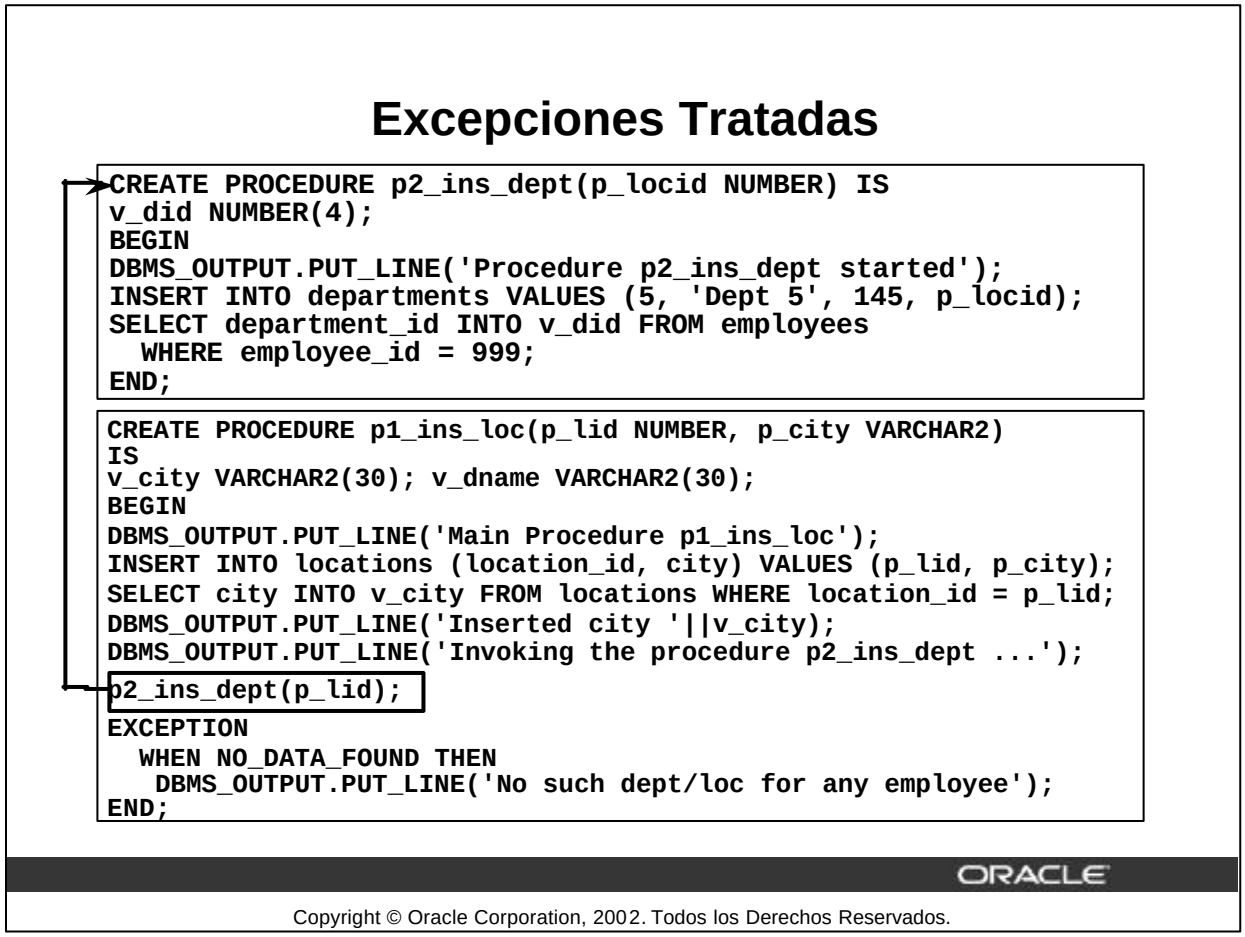
ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Cómo Afectan las Excepciones Tratadas al Procedimiento de Llamada

Cuando se desarrollan procedimientos que se llaman desde otros procedimientos, deberían tener en cuenta los efectos que tienen las excepciones tratadas y las excepciones no tratadas en la transacción y en el procedimiento de llamada.

Cuando se produce una excepción en un procedimiento llamado, el control se transfiere inmediatamente a la sección de excepciones de ese bloque. Si la excepción está tratada, el bloque termina y el control pasa al programa de llamada. Todas las sentencias DML (Lenguaje de Manipulación de Datos) que se emitan antes de la excepción se mantienen como parte de la transacción.



Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Cómo Afectan las Excepciones Tratadas al Procedimiento de Llamada (continuación)

En el ejemplo de la transparencia se muestran dos procedimientos. El procedimiento `P1_INS_LOC` inserta una nueva ubicación (que se ha proporcionado a través de los parámetros) en la tabla `LOCATIONS`. El procedimiento `P2_INS_DEPT` inserta un nuevo departamento (con el identificador 5) en la nueva ubicación que se ha insertado a través del procedimiento `P1_INS_LOC`. El procedimiento `P1_INS_LOC` llama al procedimiento `P2_INS_DEPT`.

El procedimiento `P2_INS_DEPT` posee una sentencia `SELECT` que selecciona el identificador `DEPARTMENT_ID` de un empleado que no existe y produce la excepción `NO_DATA_FOUND`. Dado que esta excepción no está tratada en el procedimiento `P2_INS_DEPT`, el control vuelve al procedimiento de llamada `P1_INS_LOC`, donde la excepción está tratada. Dado que la excepción está tratada, no se realiza un rollback en la sentencia DML del procedimiento `P2_INS_DEPT` y forma parte de la transacción del procedimiento `P1_INS_LOC`.

El siguiente código muestra que las sentencias `INSERT` de los dos procedimientos son correctas:

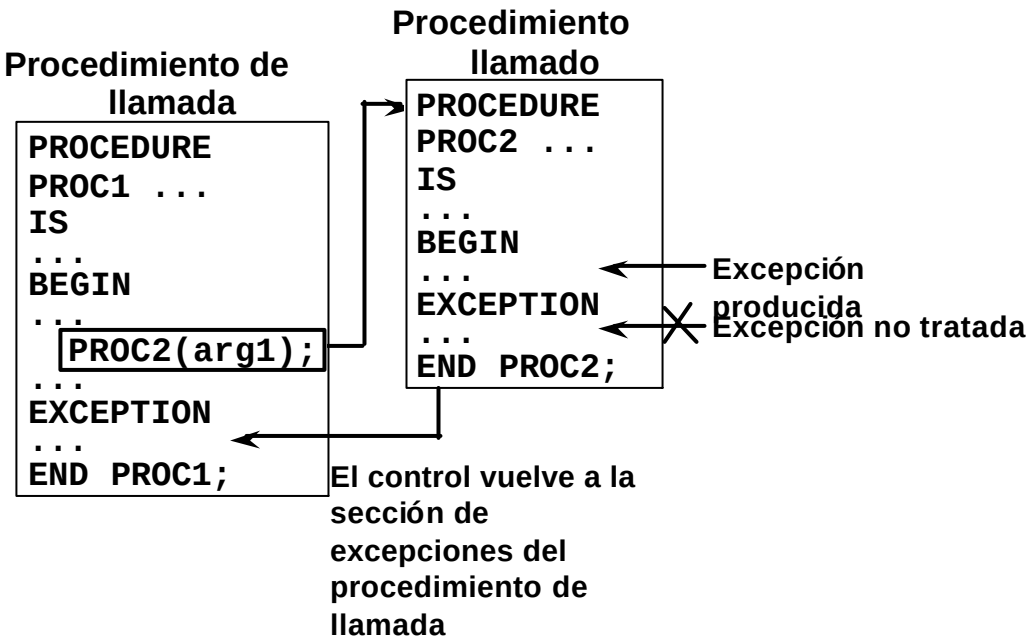
```
EXECUTE p1_ins_loc(1, 'Redwood Shores')
SELECT location_id, city FROM locations
WHERE  location_id = 1;
SELECT * FROM departments WHERE department_id = 5;
```

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

LOCATION_ID	CITY
1	Redwood Shores

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
5	Dept 5	145	1

Excepciones No Tratadas



ORACLE

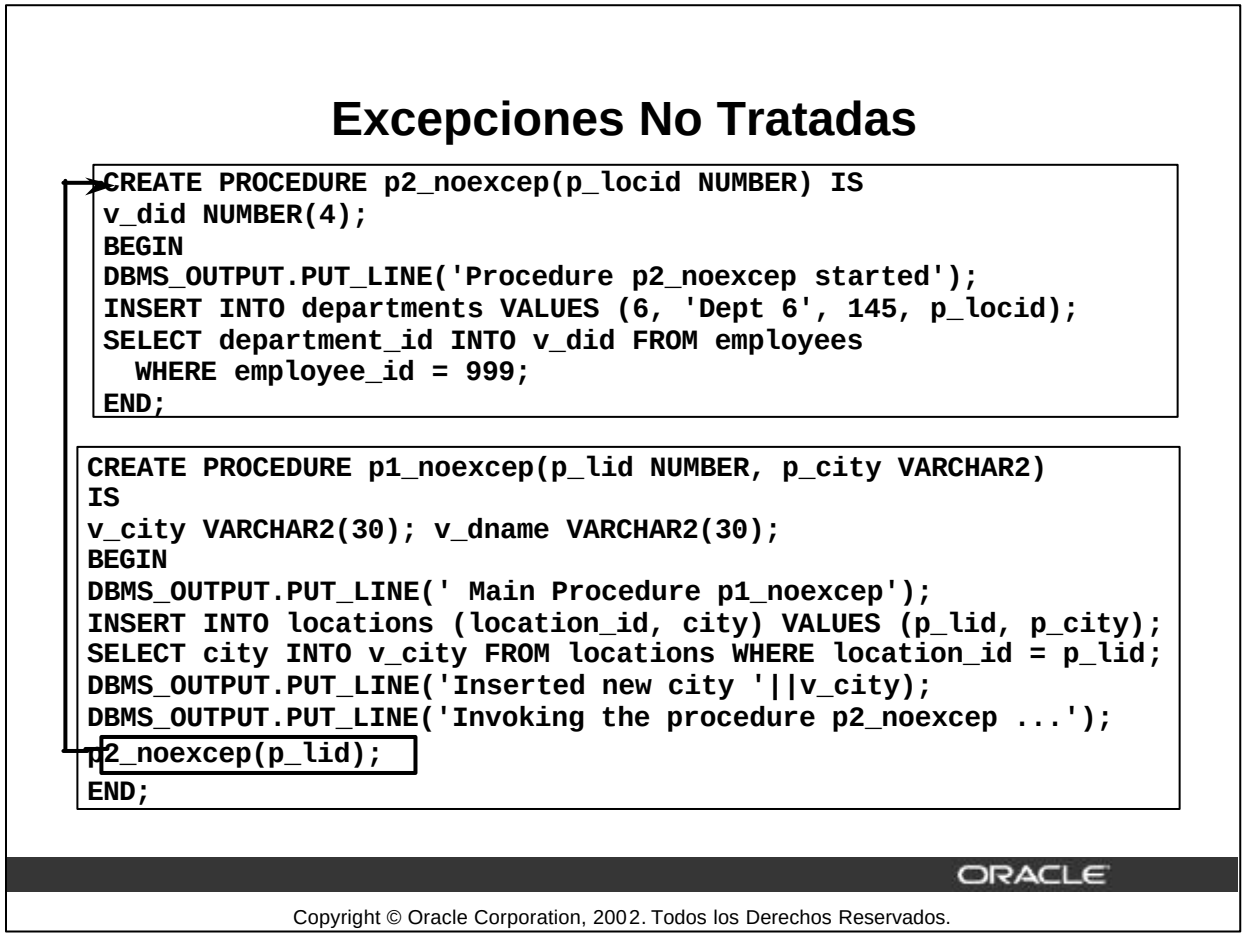
Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Cómo Afectan las Excepciones No Tratadas al Procedimiento de Llamada

Cuando se produce una excepción en un procedimiento llamado, el control se transfiere inmediatamente a la sección de excepciones de ese bloque. Si la excepción no está tratada, el bloque termina y el control pasa a la sección de excepciones del procedimiento de llamada. PL/SQL no realiza un rollback en el trabajo de la base de datos que ha hecho el subprograma.

Si la excepción está tratada en el procedimiento de llamada, todas las sentencias DML del procedimiento de llamada y del procedimiento llamado se mantienen como parte de la transacción.

Si la excepción no está tratada en el procedimiento de llamada, este procedimiento termina y la excepción se propaga al entorno de llamada. Se realiza un rollback en todas las sentencias DML del procedimiento de llamada y del procedimiento llamado, junto con cualquier cambio en cualquier variable del host. El entorno del host determina el resultado de la excepción no tratada.



Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Cómo Afectan las Excepciones No Tratadas al Procedimiento de Llamada (continuación)

En el ejemplo de la transparencia se muestran dos procedimientos. El procedimiento `P1_NOEXCEP` inserta una nueva ubicación (que se ha proporcionado a través de los parámetros) en la tabla `LOCATIONS`. El procedimiento `P2_NOEXCEP` inserta un nuevo departamento (con el identificador 5) en la nueva ubicación que se ha insertado a través del procedimiento `P1_NOEXCEP`. El procedimiento `P1_NOEXCEP` llama al procedimiento `P2_NOEXCEP`.

El procedimiento `P2_NOEXCEP` posee una sentencia `SELECT` que selecciona el identificador `DEPARTMENT_ID` de un empleado que no existe y produce la excepción `NO_DATA_FOUND`. Dado que esta excepción no está tratada en el procedimiento `P2_NOEXCEP`, el control vuelve al procedimiento de llamada `P1_NOEXCEP`. La excepción no está tratada. Dado que la excepción no está tratada, se realiza un rollback en la sentencia DML del procedimiento `P2_NOEXCEP` junto con la transacción del procedimiento `P1_NOEXCEP`.

El siguiente código muestra que las sentencias DML de los dos procedimientos fallan:

```
EXECUTE p1_noexcep(3, 'New Delhi')
SELECT location_id, city FROM locations
WHERE location_id = 3;
SELECT * FROM departments WHERE department_id = 6;
```

```
BEGIN p1_noexcep(3,'New Delhi'); END;
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at "PLSQL.P2_NOEXCEP", line 7
ORA-06512: at "PLSQL.P1_NOEXCEP", line 12
ORA-06512: at line 1
```

Eliminación de Procedimientos

Borre un procedimiento almacenado en la base de datos.

Sintaxis:

```
DROP PROCEDURE procedure_name
```

Ejemplo:

```
DROP PROCEDURE raise_salary;
```

```
Procedure dropped.
```

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Eliminación de Procedimientos

Cuando un procedimiento almacenado ya no sea necesario, puede utilizar una sentencia SQL para borrarlo.

Para eliminar un procedimiento del servidor utilizando *i*SQL*Plus, hay que ejecutar el comando SQL `DROP PROCEDURE`.

La emisión de rollbacks no tiene ningún efecto si ya se ha ejecutado un comando DDL (Lenguaje de Definición de Datos) como `DROP PROCEDURE`, que valida cualquier transacción pendiente.

Resumen

En esta lección, ha aprendido que:

- **Un procedimiento es un subprograma que realiza una acción.**
- **Para crear procedimientos, se utiliza el comando `CREATE PROCEDURE`.**
- **Puede compilar y guardar los procedimientos en la base de datos.**
- **Los parámetros sirven para transferir datos desde el entorno de llamada al procedimiento.**
- **Existen tres modos de parámetros: `IN`, `OUT` e `IN OUT`.**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Resumen

Un procedimiento es un subprograma que realiza una acción específica. Puede compilar y guardar los procedimientos en la base de datos como procedimientos almacenados. Los procedimientos pueden devolver cero o más valores a su entorno de llamada a través de sus parámetros. Existen tres modos de parámetros: `IN`, `OUT` e `IN OUT`.

Resumen

- **Los subprogramas locales son programas que están definidos en la sección de declaración de otro programa.**
- **Los procedimientos se pueden llamar desde cualquier herramienta o cualquier lenguaje que sea compatible con PL/SQL.**
- **Debería tener en cuenta el efecto de las excepciones tratadas y no tratadas en las transacciones y en los procedimientos de llamada.**
- **Para eliminar procedimientos de la base de datos, utilice el comando DROP PROCEDURE.**
- **Los procedimientos son como bloques de construcción de la aplicación.**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Resumen (continuación)

Los subprogramas que están definidos en la sección de declaración de otro programa se denominan subprogramas locales. El ámbito de los subprogramas locales es la unidad de programa en la que están definidos.

Debería tener en cuenta el efecto de las excepciones tratadas y no tratadas en las transacciones y en los procedimientos de llamada. Las excepciones se manejan en la sección de excepciones de un subprograma.

Se pueden modificar y eliminar los procedimientos. También puede crear procedimientos del cliente que pueden utilizar las aplicaciones del cliente.

Visión General de la Práctica 9

Esta práctica cubre los siguientes temas:

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

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Visión General de la Práctica 9

En esta práctica, tendrá que crear procedimientos que emitan comandos de consulta y DML.

Si encuentra errores de compilación mientras está utilizando *i*SQL*Plus, use el comando `SHOW ERRORS`. El uso del comando `SHOW ERRORS` se explica más detalladamente en la lección *Gestión de Subprogramas.*

Si tiene que corregir algún error de compilación en *i*SQL*Plus, hágalo en el archivo de comandos original, no en el buffer, y luego vuelva a ejecutar la nueva versión del archivo. De esta manera, guardará una nueva versión del procedimiento en el diccionario de datos.

Práctica 9

Nota: En el Apéndice D "Descripciones de Tablas y Datos", encontrará una serie de descripciones de tablas y datos de ejemplo.

Guarde los subprogramas como archivos `.sql`, haciendo clic en el botón `Save Script`.

No se olvide de definir `SERVEROUTPUT ON` si lo había desactivado anteriormente.

1. Cree y llame al procedimiento `ADD_JOB` y observe los resultados.
   a. Cree un procedimiento denominado `ADD_JOB` para insertar un nuevo puesto de trabajo en la tabla `JOBS`. Proporcione el identificador y el nombre del puesto de trabajo utilizando dos parámetros.
   b. Compile el código y llame al procedimiento utilizando `IT_DBA` como identificador del puesto de trabajo, y `Database Administrator` como nombre del puesto de trabajo. Consulte la tabla `JOBS` para ver los resultados.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Database Administrator		

   c. Vuelva a llamar al procedimiento y transfiera el identificador del puesto de trabajo `ST_MAN` y el nombre del puesto de trabajo `Stock Manager`. ¿Qué ocurre y por qué?

      _____

      _____

2. Cree un procedimiento denominado `UPD_JOB` para modificar un puesto de trabajo en la tabla `JOBS`.
   a. Cree un procedimiento denominado `UPD_JOB` para actualizar el nombre del puesto de trabajo. Proporcione el identificador del puesto de trabajo y el nuevo nombre del puesto de trabajo utilizando dos parámetros. Si no se produce la actualización, incluya el manejo de excepciones necesario.
   b. Compile el código; llame al procedimiento para cambiar el nombre del puesto de trabajo que tiene el identificador `IT_DBA` por `Data Administrator`. Consulte la tabla `JOBS` para ver los resultados.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Data Administrator		

      Compruebe también el manejo de excepciones intentando actualizar un trabajo que no existe (puede utilizar el identificador del puesto de trabajo `IT_WEB` y el nombre del puesto de trabajo `Web Master`).

Práctica 9 (continuación)

3. Cree un procedimiento denominado DEL_JOB para suprimir un puesto de trabajo de la tabla JOBS.
   a. Cree un procedimiento denominado DEL_JOB para suprimir un puesto de trabajo. Si el puesto de trabajo no se suprime, incluya el manejo de excepciones necesario.
   b. Compile el código; llame al procedimiento utilizando el identificador IT_DBA. Consulte la tabla JOBS para ver los resultados.

      ```
      no rows selected
      ```

      Compruebe también el manejo de excepciones intentando suprimir un puesto de trabajo que no existe (puede utilizar el identificador del puesto de trabajo IT_WEB). Como resultado, debería recibir el mensaje que haya utilizado en la sección de manejo de excepciones del procedimiento.
4. Cree un procedimiento denominado QUERY_EMP para consultar la tabla EMPLOYEES y recuperar el sueldo y el identificador del puesto de trabajo de un empleado cuando se proporcione el identificador de dicho empleado.
   a. Cree un procedimiento que devuelva un valor desde las columnas SALARY y JOB_ID con respecto a un identificador de empleado específico.

      Utilice variables del host para los dos parámetros OUT del sueldo y el identificador del puesto de trabajo.
   b. Compile el código, llame al procedimiento para que muestre el sueldo y el identificador del puesto de trabajo del empleado que posee el identificador 120.

G_SAL
8000

G_JOB
ST_MAN

   c. Llame de nuevo al procedimiento y transfiera un identificador EMPLOYEE_ID de 300. ¿Qué ocurre y por qué?

      _____

      _____

10

Creación de Funciones

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Objetivos

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

- **Describir los usos de las funciones**
- **Crear procedimientos almacenados**
- **Llamar a una función**
- **Eliminar una función**
- **Diferenciar un procedimiento de una función**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Finalidad de la Lección

En esta lección aprenderá a crear y a llamar funciones.

Visión General de las Funciones Almacenadas

- **Una función es un bloque PL/SQL denominado que devuelve un valor.**
- **Se puede almacenar en la base de datos como objeto de esquema para repetir su ejecución.**
- **Una función se denomina como parte de una expresión.**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Funciones Almacenadas

Una función es un bloque PL/SQL denominado que puede aceptar parámetros y que se puede llamar. En términos generales, las funciones sirven para calcular un valor. Las funciones y los procedimientos tienen una estructura similar. Las funciones deben devolver un valor al entorno de llamada, mientras que los procedimientos devuelven cero o más valores a su entorno de llamada. Al igual que los procedimientos, las funciones poseen una cabecera, una sección declarativa, una sección ejecutable y una sección de manejo de excepciones opcional. Las funciones deben tener una cláusula RETURN en la cabecera y al menos una sentencia RETURN en la sección ejecutable.

Las funciones se pueden almacenar en la base de datos como objetos de esquema para repetir su ejecución. Las funciones que están almacenadas en la base de datos se denominan funciones almacenadas. También se pueden crear funciones en aplicaciones del cliente. Esta lección explica la creación de funciones almacenadas. Consulte el apéndice "Creación de Unidades de Programa con Procedure Builder" para crear aplicaciones del cliente.

Las funciones aumentan la capacidad de reutilización y de mantenimiento. Una vez validadas, se pueden utilizar en cualquier número de aplicaciones. Si los requisitos del procesamiento cambian, sólo hay que actualizar la función.

La función se denomina como parte de una expresión SQL o como parte de una expresión PL/SQL. En una expresión SQL, la función debe obedecer una serie de reglas específicas para controlar los efectos secundarios. En una expresión PL/SQL, el identificador de la función actúa como una variable cuyo valor depende de los parámetros que se le transfieran.

Sintaxis de la Creación de Funciones

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter1 [mode1] datatype1,
  parameter2 [mode2] datatype2,
  . . .)]
RETURN datatype
IS|AS
PL/SQL Block;
```

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

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

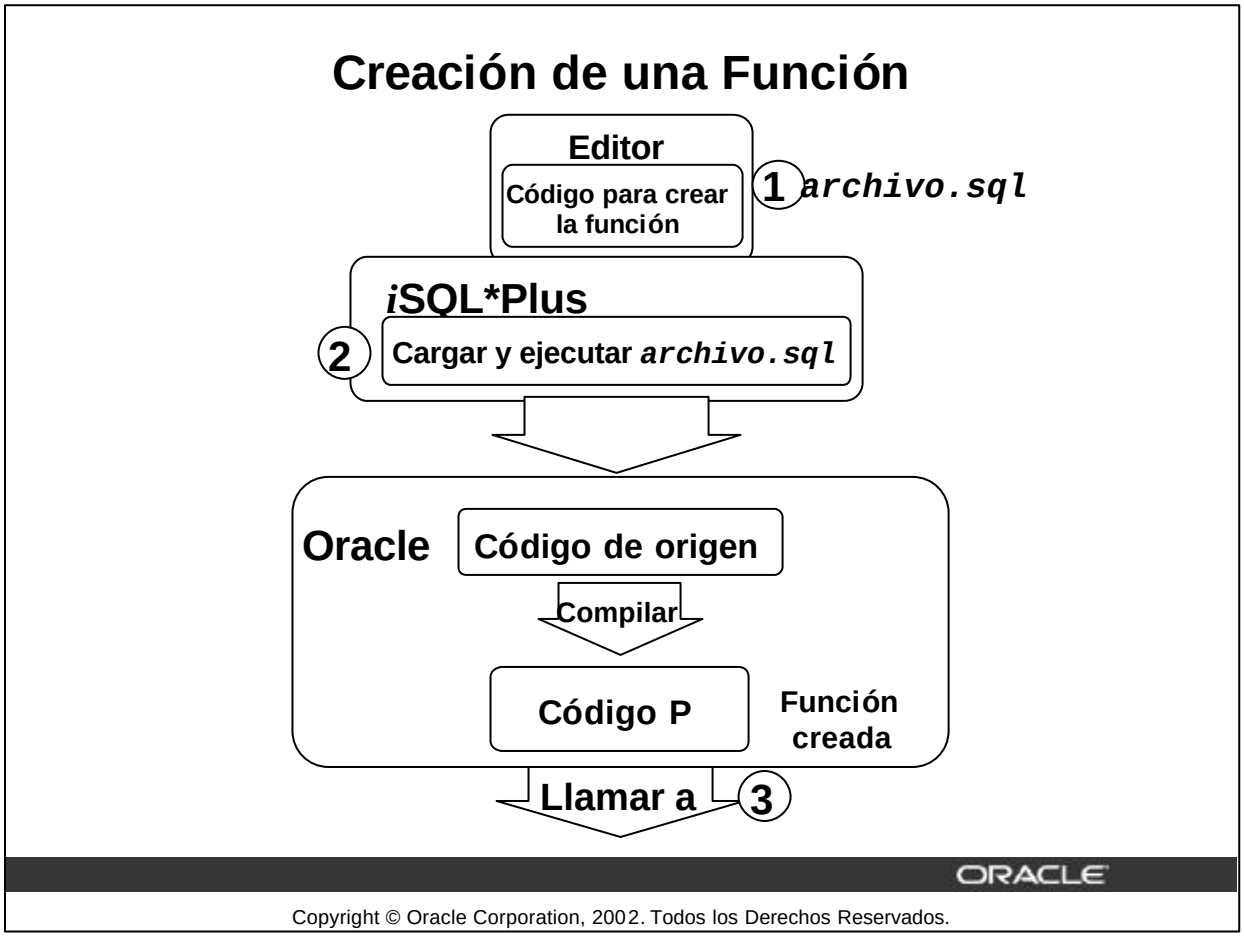
Sintaxis de la Creación de Funciones

Una función es un bloque PL/SQL que devuelve un valor. Para crear nuevas funciones, se utiliza la sentencia `CREATE FUNCTION`, que puede declarar una lista de parámetros y, a su vez, debe devolver un valor y definir las acciones que va a realizar el bloque PL/SQL estándar.

- La opción `REPLACE` indica que, si la función existe, se borrará y se sustituirá por la nueva versión que ha creado la sentencia.
- El tipo de dato `RETURN` no debe incluir una especificación de tamaño.
- Los bloques PL/SQL comienzan con `BEGIN` o con la declaración de variables locales y terminan con `END` o con `END` *`function_name`*. Debe haber al menos una sentencia `RETURN` (*`expression`*). No se puede hacer referencia a variables ligadas o del host en el bloque PL/SQL de una función almacenada.

Definiciones de la Sintaxis

Parámetro	Descripción
nombre_función	Nombre de la función
parámetro	Nombre de una variable PL/SQL cuyo valor se transfiere a la función
modo	El tipo de parámetro; sólo se deberían declarar parámetros `IN`
tipo_de_dato	Tipo de dato del parámetro
tipo_de_dato `RETURN`	Tipo de dato del valor `RETURN` que debe dar como resultado la función
bloque PL/SQL	Cuerpo procedural que define la acción que realiza la función



Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Cómo Desarrollar Funciones Almacenadas

Estos son los pasos básicos que hay que seguir para desarrollar una función almacenada. En las próximas dos páginas encontrará más detalles acerca de la creación de funciones.

1. Escriba la sintaxis: Introduzca el código necesario para crear una función en un editor de texto y guárdelo como un archivo de comandos SQL.
2. Compile el código: Utilice *i*SQL*Plus para cargar y ejecutar el archivo de comandos SQL. El código de origen se compila en el código P. Ya se ha creado la función.
3. Llame a la función desde un bloque PL/SQL.

Devolución de un Valor

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

Aunque en una función está permitido utilizar múltiples sentencias `RETURN` (normalmente en el interior de una sentencia `IF`), sólo se ejecutará una sentencia `RETURN`, ya que el procesamiento del bloque termina una vez devuelto el valor.

Nota: El compilador PL/SQL genera el *pseudo código* o el código P basándose en el código analizado. El motor PL/SQL lo ejecuta cuando se llama al procedimiento.

Creación de una Función Almacenada Utilizando *i*SQL*Plus

1. **Introduzca el texto de la sentencia `CREATE FUNCTION` en un editor y guárdelo como un archivo de comandos SQL.**
2. **Ejecute el archivo de comandos para almacenar el código de origen y compilar la función.**
3. **Utilice `SHOW ERRORS` para ver los errores de compilación.**
4. **Una vez compilada con éxito, llame a la función.**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Cómo Crear una Función Almacenada

1. Introduzca el texto de la sentencia `CREATE FUNCTION` en un editor del sistema o un procesador de textos y guárdela como un archivo de comandos (con la extensión `.sql`).
2. Desde *i*SQL*Plus, cargue y ejecute el archivo de comandos para guardar el código de origen y compilarlo en código P.
3. Utilice `SHOW ERRORS` para ver cualquier error de compilación.
4. Cuando haya conseguido compilar el código, la función ya estará lista para ser ejecutada. Llame a dicha función desde un entorno de Oracle Server.

Los archivos de comandos con la sentencia `CREATE FUNCTION` permiten cambiar la sentencia en el caso de que se produzca algún error de compilación o de tiempo de ejecución, o realizar cambios posteriores en dicha sentencia. No se puede llamar a una función que contiene cualquier error de compilación o de tiempo de ejecución. En *i*SQL*Plus, utilice `SHOW ERRORS` para ver los errores de compilación.

Si ejecuta la sentencia `CREATE FUNCTION`, el código de origen se almacenará en el diccionario de datos, aunque la función contenga errores de compilación.

Nota: Si hay algún error de compilación y realiza cambios posteriores en la sentencia `CREATE FUNCTION`, primero debe borrar la función o utilizar la sintaxis `OR REPLACE`.

Creación de una Función Almacenada Utilizando *i*SQL*Plus: Ejemplo

get_salary.sql

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

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Ejemplo

Cree una función con un parámetro `IN` que devuelva un número.

Ejecute el archivo de comandos para crear la función `GET_SAL`. Llame a una función como parte de una expresión PL/SQL, porque la función devolverá un valor al entorno de llamada.

Es una buena práctica de programación asignar un valor de retorno a una variable y utilizar una sola sentencia `RETURN` en la sección ejecutable del código. También puede haber una sentencia `RETURN` en la sección de excepciones del programa.

Ejecución de Funciones

- **Llame a una función como parte de una expresión PL/SQL.**
- **Cree una variable que guarde el valor devuelto.**
- **Ejecute la función. La variable se rellenará con el valor que se ha devuelto a través de una sentencia `RETURN`.**

ORACLE

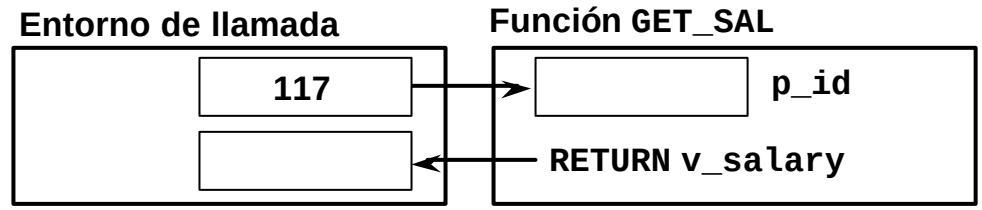
Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Ejecución de la Función

Una función puede aceptar uno o varios parámetros, pero debe devolver un solo valor. Para llamar a funciones como parte de las expresiones PL/SQL, hay que utilizar variables que guarden el valor devuelto.

Aunque los tres modos de parámetros, `IN` (por defecto), `OUT` e `IN OUT`, se pueden utilizar con cualquier subprograma, evite el uso de los modos `OUT` e `IN OUT` con funciones. El objetivo de una función es aceptar cero o más argumentos (parámetros reales) y devolver un solo valor. No es una buena práctica de programación hacer que una función devuelva varios valores. Además, las funciones no deben tener efectos secundarios que cambien los valores de las variables que no son locales del subprograma. Más adelante, en esta misma lección, se explicarán los efectos secundarios.

Ejecución de Funciones: Ejemplo



1. Cargue y ejecute el archivo `get_salary.sql` para crear la función

```
2 → VARIABLE g_salary NUMBER

3 → EXECUTE :g_salary := get_sal(117)

4 → PRINT g_salary
```

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

G_SALARY
2800

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Ejemplo

Ejecute la función `GET_SAL` desde *i*SQL*Plus:

1. Cargue y ejecute el archivo de comandos `get_salary.sql` para crear la función almacenada `GET_SAL`.
2. Cree una variable del host que rellenará la sentencia `RETURN (variable)` en el interior de la función.
3. Utilice el comando `EXECUTE` en *i*SQL*Plus para llamar a la función `GET_SAL` creando una expresión PL/SQL. Proporcione un valor al parámetro (el identificador del empleado, en este ejemplo). El valor que devuelve la función se guardará en la variable del host, `G_SALARY`. Tenga en cuenta el uso de los dos puntos (:) para hacer referencia a la variable del host.
4. Vea el resultado de la llamada de la función utilizando el comando `PRINT`. El empleado Tobias, que posee el identificador de empleado 117, gana un sueldo de 2800 al mes.

En una función, debe haber al menos una ruta de acceso de ejecución que lleve a una sentencia `RETURN`. De lo contrario, se produciría el error `Function returned without value` en tiempo de ejecución.

Ventajas de las Funciones Definidas por el Usuario en las Expresiones SQL

- **Amplie SQL donde las actividades sean muy complejas, demasiado incómodas o no estén disponibles en SQL**
- **Pueden mejorar la eficacia si se utilizan en la cláusula `WHERE` para filtrar los datos, en lugar de filtrar los datos en la aplicación**
- **Pueden manipular cadenas de caracteres**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Llamada a Funciones Definidas por el Usuario desde Expresiones SQL

Las expresiones SQL pueden hacer referencia a las funciones PL/SQL definidas por el usuario. Las funciones definidas por el usuario se pueden colocar en los mismos lugares que las funciones SQL incorporadas.

Ventajas

- Permiten realizar cálculos que son demasiado complejos, incómodos o no están disponibles en SQL
- Aumentan la independencia de los datos al procesar el análisis de datos complejo en el interior de Oracle Server, en lugar de recuperar los datos en una aplicación
- Aumentan la eficicacia de las consultas realizando las funciones en la consulta en lugar de realizarlas en la aplicación
- Manipulan nuevos tipos de datos (por ejemplo, la latitud y la longitud) codificando cadenas de caracteres y utilizando funciones en las cadenas

Llamada a Funciones en Expresiones SQL: Ejemplo

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

Function created.

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

6 rows selected.

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Ejemplo

La transparencia muestra cómo se crea la función `tax`, que se llama desde una sentencia `SELECT`. La función acepta un parámetro `NUMBER` y devuelve el impuesto después de multiplicar el valor del parámetro por 0.08.

En *i*SQL*Plus, llame a la función `TAX` en el interior de una consulta que muestre el identificador, el nombre, el sueldo y el impuesto del empleado.

Lugares para Llamar a Funciones Definidas por el Usuario

- Lista de selección de un comando **SELECT**
- Condición de las cláusulas **WHERE** y **HAVING**
- Cláusulas **CONNECT BY**, **START WITH**, **ORDER BY** y **GROUP BY**
- Cláusula **VALUES** del comando **INSERT**
- Cláusula **SET** del comando **UPDATE**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Uso de las Funciones Definidas por el Usuario

Las funciones PL/SQL definidas por el usuario se pueden llamar desde cualquier expresión SQL desde la que se pueda llamar a una función incorporada.

Ejemplo:

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

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

...

10 rows selected.

Restricciones en las Llamadas a Funciones desde Expresiones SQL

Para poder llamar a una función definida por el usuario desde las expresiones SQL, dicha función debe:

- **Ser una función almacenada**
- **Aceptar sólo parámetros `IN`**
- **Aceptar sólo tipos de dato SQL válidos, no tipos específicos de PL/SQL como, por ejemplo, los parámetros**
- **Devolver tipos de dato SQL válidos, no tipos específicos de PL/SQL**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Restricciones al Llamar a Funciones desde Expresiones SQL

Para poder llamar a una función PL/SQL definida por el usuario desde las expresiones SQL, esa función debe cumplir ciertos requisitos.

- Los parámetros de una función PL/SQL que se llaman desde una sentencia SQL deben utilizar la notación posicional. No soporta notación denominada.
- Las funciones PL/SQL almacenadas no se pueden llamar desde la cláusula de restricción `CHECK` de un comando `CREATE` o `ALTER TABLE` ni se pueden utilizar para especificar un valor por defecto para una columna.
- Debe poseer o tener el privilegio `EXECUTE` sobre la función para poder llamarla desde una sentencia SQL.
- Las funciones deben devolver tipos de dato SQL válidos. No pueden ser tipos de dato específicos de PL/SQL como, por ejemplo, `BOOLEAN`, `RECORD` o `TABLE`. Esta misma restricción se aplica a los parámetros de la función.

Nota: Desde las sentencias SQL sólo se puede llamar a funciones almacenadas. No se puede llamar a procedimientos almacenados.

En PL/SQL 2.1 o posterior se puede utilizar una función PL/SQL definida por el usuario en una expresión SQL. Las herramientas que utilicen versiones anteriores de PL/SQL no son compatibles con esta funcionalidad. Antes de Oracle9*i*, las funciones definidas por el usuario sólo podían ser funciones de una sola fila. A partir de Oracle9*i*, las funciones definidas por el usuario también se pueden definir como funciones agregadas.

Nota: Las funciones que se pueden llamar desde expresiones SQL no pueden contener parámetros `OUT` e `IN OUT`. Otras funciones pueden contener parámetros con estos modos, pero no es recomendable.

Restricciones en las Llamadas a Funciones desde Expresiones SQL

- **Las funciones que se llaman desde expresiones SQL no pueden contener sentencias DML.**
- **Las funciones que se llaman desde sentencias `UPDATE/DELETE` en una tabla T, no pueden contener sentencias DML en la misma tabla T.**
- **Las funciones que se llaman desde una sentencia `UPDATE` o `DELETE` en una tabla T, no pueden consultar la misma tabla.**
- **Las funciones que se llaman desde sentencias SQL no pueden contener sentencias que terminen las transacciones.**
- **En la función no se permiten llamadas a subprogramas que infrinjan la restricción anterior.**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Control de los Efectos Secundarios

Para ejecutar una sentencia SQL que llame a una función almacenada, Oracle Server debe saber si la función carece de efectos secundarios. Los efectos secundarios son cambios no aceptables en las tablas de base de datos. Por lo tanto, las restricciones se aplican a las funciones almacenadas que se llaman desde expresiones SQL.

Restricciones

- Si se llama desde una sentencia `SELECT` o una sentencia `UPDATE` o `DELETE` paralelizada, la función no puede modificar ninguna tabla de base de datos.
- Si se llama desde una sentencia `UPDATE` o `DELETE`, la función no puede consultar ni modificar ninguna tabla de base de datos modificada por esa sentencia.
- Si se llama desde una sentencia `SELECT`, `INSERT`, `UPDATE` o `DELETE`, la función no puede ejecutar ni las sentencias de control de transacciones SQL (como, por ejemplo, `COMMIT`), ni las sentencias de control de sesiones (como, por ejemplo, `SET ROLE`), ni las sentencias de control del sistema (como, por ejemplo, `ALTER SYSTEM`). Además, no puede ejecutar sentencias DDL (como, por ejemplo, `CREATE`) porque van seguidas de una validación automática.
- La función no puede llamar a otro subprograma que infrinja alguna de las restricciones anteriores.

Restricciones en las Llamadas desde SQL

```
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, 'employee 1', 'emp1@company.com',
                    SYSDATE, 'SA_MAN', 1000);
   RETURN (p_sal + 100);
END;
/
```

```
Function created.
```

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

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

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Restricciones en las Llamadas a Funciones desde SQL: Ejemplo

El código de la transparencia muestra un ejemplo en el que hay una sentencia DML en una función. La función DML_CALL_SQL contiene una sentencia DML que inserta un nuevo registro en la tabla EMPLOYEES. Esta función se llama desde la sentencia UPDATE que modifica el sueldo del empleado 170 con la cantidad que devuelve la función. La sentencia UPDATE devuelve un error que informa de que la tabla está mutando.

Observe el siguiente ejemplo, donde la función QUERY_CALL_SQL consulta la columna SALARY de la tabla EMPLOYEE:

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

Cuando se llama a la función anterior desde la siguiente sentencia UPDATE, devuelve un mensaje de error similar al que se muestra en la transparencia.

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

Eliminación de Funciones

Borre una función almacenada.

Sintaxis:

```
DROP FUNCTION function_name
```

Ejemplo:

```
DROP FUNCTION get_sal;
```

```
Function dropped.
```

- **Todos los privilegios otorgados sobre una función quedan revocados cuando se borra esa función.**
- **La sintaxis `CREATE OR REPLACE` equivale a borrar una función y volverla a crear. Los privilegios otorgados sobre la función siguen siendo los mismos cuando se utiliza esta sintaxis.**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Eliminación de Funciones

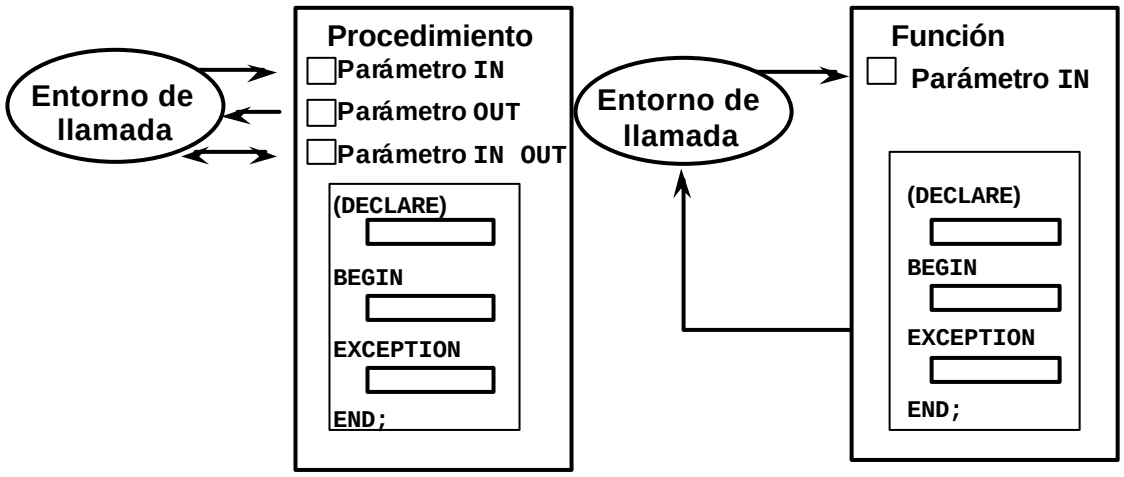
Cuando una función almacenada ya no es necesaria, puede utilizar una sentencia SQL en *i*SQL*Plus para borrarla.

Para eliminar una función almacenada utilizando *i*SQL*Plus, hay que ejecutar el comando SQL `DROP FUNCTION`.

`CREATE OR REPLACE` frente a `DROP` y `CREATE`

La cláusula `REPLACE` de la sintaxis `CREATE OR REPLACE` equivale a borrar una función y volverla a crear. Cuando se utiliza la sintaxis `CREATE OR REPLACE`, los privilegios otorgados sobre este objeto a otros usuarios siguen siendo los mismos. Cuando se borra (`DROP`) una función y luego se vuelve a crear, los privilegios otorgados sobre esta función quedan revocados automáticamente.

¿Procedimiento o Función?



ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

En Qué se Diferencian los Procedimientos y las Funciones

Los procedimientos se crean para almacenar una serie de acciones que se ejecutan posteriormente. Los procedimientos pueden contener cero o más valores transferibles hasta y desde el entorno de llamada, pero no tienen que devolver un valor.

Las funciones se crean cuando se quiere calcular un valor que se debe devolver al entorno de llamada. Las funciones pueden contener cero o más parámetros que se transfieren desde el entorno de llamada. Las funciones deberían devolver un único valor y éste se devuelve a través de la sentencia `RETURN`. Las funciones que se utilizan en las sentencias SQL no pueden tener parámetros de modo `OUT` ni `IN OUT`.

Comparación entre Procedimientos y Funciones

Procedimientos	Funciones
Se ejecutan como una sentencia PL/SQL	Se les llama como parte de una expresión
No contienen la cláusula RETURN en la cabecera	Deben contener una cláusula RETURN en la cabecera
Pueden devolver un valor, varios valores o ninguno	Deben devolver un solo valor
Pueden contener una sentencia RETURN	Deben contener al menos una sentencia RETURN

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

En Qué se Diferencian los Procedimientos y las Funciones (continuación)

Los procedimientos que contienen un parámetro OUT se pueden volver a escribir como una función que contiene la sentencia RETURN.

Ventajas de los Procedimientos y las Funciones Almacenadas

- **Mejor rendimiento**
- **Fácil mantenimiento**
- **Mayor integridad y seguridad de los datos**
- **Mayor claridad del código**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Ventajas

Además de permitir el desarrollo de aplicaciones modulares, las funciones y los procedimientos almacenados poseen las siguientes ventajas:

- Mejor rendimiento
  - Se evitan las repeticiones del análisis para múltiples usuarios utilizando el área de SQL compartido
  - Se evita el análisis de PL/SQL en tiempo de ejecución al realizarlo en tiempo de compilación
  - Se reduce el número de llamadas a la base de datos y se disminuye el tráfico de red agrupando comandos
- Fácil mantenimiento
  - Modificación de las rutinas en línea sin interferir con otros usuarios
  - Modificación de una rutina que afecta a varias aplicaciones
  - Modificación de una rutina para no duplicar las pruebas
- Mayor integridad y seguridad de los datos
  - Control del acceso indirecto a los objetos de base de datos por parte de los usuarios sin privilegios utilizando privilegios de seguridad.
  - Garantía de que las acciones relacionadas se realizan juntas o no se realizan, al canalizar la actividad de las tablas relacionadas a través de una sola ruta de acceso
- Mayor claridad del código: El uso de los nombres de identificador adecuados para describir las acciones de las rutinas, reduce la necesidad de incluir comentarios y mejora la claridad del código.

Resumen

En esta lección, ha aprendido que:

- **Una función es un bloque PL/SQL denominado que debe devolver un valor.**
- **Las funciones se crean utilizando la sintaxis `CREATE FUNCTION`.**
- **Las funciones se llaman como parte de una expresión.**
- **Se puede llamar a las funciones almacenadas en la base de datos en sentencias SQL.**
- **Las funciones se pueden eliminar de la base de datos utilizando la sintaxis `DROP FUNCTION`.**
- **Generalmente, los procedimientos se utilizan para realizar una acción y las funciones para calcular un valor.**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Resumen

Una función es un bloque PL/SQL denominado que debe devolver un valor. Generalmente, las funciones se crean para calcular y devolver un valor, y los procedimiento para realizar una acción.

Las funciones se pueden crear o borrar.

Las funciones se llaman como parte de una expresión.

Visión General de la Práctica 10

Esta práctica cubre los siguientes temas:

- **Creación de funciones almacenadas**
  - **Para consultar una tabla de base de datos y devolver valores específicos**
  - **Para ser utilizadas en una sentencia SQL**
  - **Para insertar una nueva fila con valores de parámetros específicos en una tabla de base de datos**
  - **Utilizando valores de parámetros por defecto**
- **Llamada a una función almacenada desde una sentencia SQL**
- **Llamada a una función almacenada desde un procedimiento almacenado**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Visión General de la Práctica 10

Si encuentra errores de compilación mientras está utilizando *i*SQL*Plus, utilice el comando `SHOW ERRORS`.

Si tiene que corregir algún error de compilación en *i*SQL*Plus, hágalo en el archivo de comandos original, no en el buffer, y luego vuelva a ejecutar la nueva versión del archivo. De esta manera, guardará una nueva versión de la unidad de programa en el diccionario de datos.

Práctica 10

1. Cree y llame a la función `Q_JOB` para devolver el nombre de un puesto de trabajo.
   a. Cree una función llamada `Q_JOB` para devolver el nombre de un puesto de trabajo a una variable del host.
   b. Compile el código; cree una variable del host `G_TITLE` y llame a la función con el identificador de puesto de trabajo `SA_REP`. Consulte la variable del host para ver los resultados.

G_TITLE
Sales Representative

2. Cree una función denominada `ANNUAL_COMP` para devolver el sueldo anual aceptando dos parámetros: el sueldo mensual del empleado y la comisión. La función debería tratar valores `NULL`.
   a. Cree y llame a la función `ANNUAL_COMP`, transfiriendo valores para la comisión y el sueldo mensual. Cualquiera de los dos valores que se han transferido, o los dos, puede ser `NULL`, pero la función debería seguir devolviendo el sueldo anual, que no es `NULL`. El sueldo anual se define por medio de la fórmula básica:

      ```
      (sueldo*12) + (porcentaje_comisión*sueldo*12)
      ```

   b. Utilice la función en una sentencia `SELECT` sobre la tabla `EMPLOYEES` del departamento 80.

EMPLOYEE_ID	LAST_NAME	Annual Compensation
145	Russell	235200
146	Partners	210600
147	Errazuriz	187200
148	Cambrault	171600
149	Zlotkey	151200
...		
176	Taylor	123840
177	Livingston	120960
179	Johnson	81840

34 rows selected.

Práctica 10 (continuación)

3. Cree un procedimiento, NEW_EMP, para insertar un nuevo empleado en la tabla EMPLOYEES. El procedimiento debería contener una llamada a la función VALID_DEPTID para comprobar si el identificador de departamento especificado para el nuevo empleado existe en la tabla DEPARTMENTS.

   a. Cree la función VALID_DEPTID para validar un identificador de departamento específico. La función debería devolver un valor BOOLEAN.

   b. Cree el procedimiento NEW_EMP para agregar un empleado a la tabla EMPLOYEES. Debería agregar una nueva fila a la tabla EMPLOYEES si la función devuelve TRUE. Si la función devuelve FALSE, el procedimiento debería alertar al usuario con el mensaje adecuado.

      Defina valores por defecto para la mayoría de los parámetros. La comisión por defecto es 0, el sueldo por defecto es 1000, el número de departamento por defecto es 30, el nombre del puesto de trabajo por defecto es SA_REP y el identificador de jefe por defecto es 145. Para el identificador del empleado, utilice la secuencia EMPLOYEES_SEQ. Proporcione el nombre, el apellido y la dirección de correo electrónico del empleado.

   c. Pruebe el procedimiento NEW_EMP agregando una nueva empleada llamada Jane Harris al departamento 15. Deje los valores por defecto en el resto de los parámetros. ¿Cuál es el resultado?

   d. Pruebe el procedimiento NEW_EMP agregando un nuevo empleado llamado Joe Harris al departamento 80. Deje los valores por defecto en el resto de los parámetros. ¿Cuál es el resultado?

11

Gestión de Subprogramas

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Objetivos

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

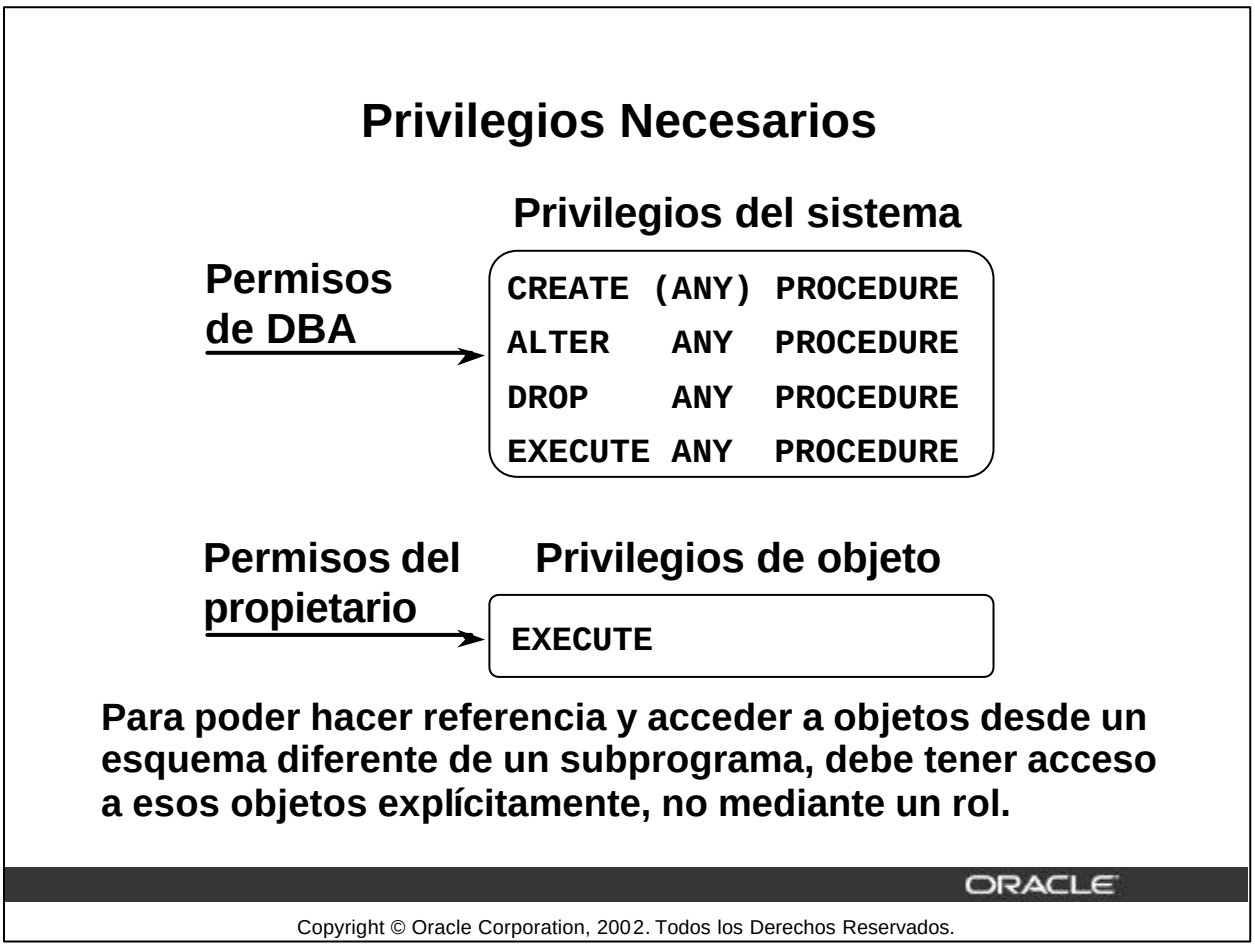
- **Comparar los privilegios del sistema con los privilegios de objeto**
- **Comparar los derechos de los invocadores con los derechos de los elementos de definición**
- **Identificar las vistas de diccionario de datos para gestionar objetos almacenados**
- **Describir cómo se depuran los subprogramas utilizando el paquete `DBMS_OUTPUT`**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Finalidad de la Lección

Esta lección presenta los requisitos de los privilegios del sistema y los privilegios de objeto. Aprenderá a utilizar el diccionario de datos para obtener información acerca de los objetos almacenados. También aprenderá a depurar subprogramas.



Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Privilegios del Sistema y de Objeto

Existen más de 80 privilegios del sistema. Los privilegios que utilizan la palabra CREATE o ANY son privilegios del sistema como, por ejemplo, `GRANT ALTER ANY TABLE TO green;`. El usuario SYSTEM o SYS es quien asigna los privilegios del sistema.

Los privilegios de objeto son derechos que se asignan a un objeto específico de un esquema e incluyen siempre el nombre del objeto. Por ejemplo, Scott puede asignar privilegios a Green para que pueda modificar su tabla EMPLOYEES de la siguiente manera:

```
GRANT ALTER ON employees TO green;
```

Para crear un subprograma PL/SQL, debe tener el privilegio del sistema CREATE PROCEDURE. Para modificar, borrar o ejecutar subprogramas PL/SQL no se necesita ningún otro privilegio.

Si un subprograma PL/SQL hace referencia a cualquier objeto que no se encuentra en el mismo esquema, deberá tener acceso a ese objeto explícitamente, no mediante un rol.

Si utiliza la palabra clave ANY, podrá crear, modificar, borrar o ejecutar sus propios subprogramas y los que se encuentran en otro esquema. Tenga en cuenta que la palabra clave ANY es opcional sólo para el privilegio CREATE PROCEDURE.

Para llamar al subprograma PL/SQL, debe tener el privilegio de objeto EXECUTE si no es el propietario y no tiene el privilegio del sistema EXECUTE ANY.

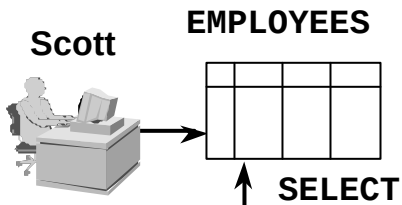
Por defecto, el subprograma PL/SQL se ejecuta en el dominio de seguridad del propietario.

Nota: La palabra clave PROCEDURE se utiliza en procedimientos almacenados, funciones y paquetes.

Otorgamiento de Acceso a los Datos

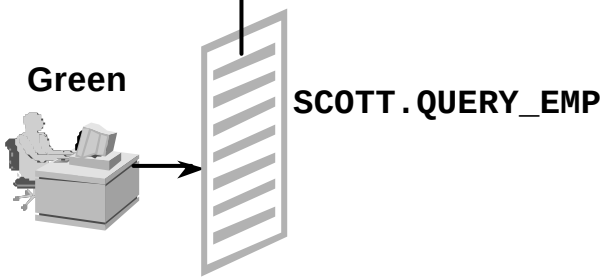
Acceso directo:

```
GRANT SELECT
ON   employees
TO   scott;
Otorgamiento realizado.
```



Acceso indirecto:

```
GRANT EXECUTE
ON   query_emp
TO   green;
Otorgamiento realizado.
```



El procedimiento se ejecuta con los privilegios del propietario (por defecto).

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Acceso Indirecto a los Datos

Suponga que la tabla EMPLOYEES está ubicada en el esquema PERSONNEL y que existe un desarrollador que se llama Scott y un usuario final que se llama Green. Asegúrese de que Green tiene acceso a la tabla EMPLOYEES únicamente utilizando el procedimiento QUERY_EMP que Scott ha creado y que sirve para consultar los registros de los empleados.

Acceso Directo

- Proporcione, desde el esquema PERSONNEL, privilegios de objeto a Scott sobre la tabla EMPLOYEES.
- Scott crea el procedimiento QUERY_EMP que consulta la tabla EMPLOYEES.

Acceso Indirecto

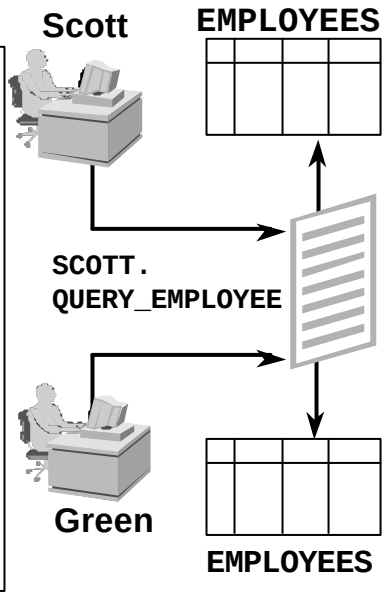
Scott proporciona el privilegio de objeto EXECUTE a Green en el procedimiento QUERY_EMP.

Por defecto, el subprograma PL/SQL se ejecuta en el dominio de seguridad del propietario. Esto se denomina derechos del elemento de definición. Dado que Scott tiene privilegios directos sobre EMPLOYEES y ha creado el procedimiento QUERY_EMP, Green puede recuperar información de la tabla EMPLOYEES utilizando el procedimiento QUERY_EMP.

Uso de los Derechos de los Invocadores

El procedimiento se ejecuta con los privilegios del usuario.

```
CREATE PROCEDURE query_employee
(p_id IN employees.employee_id%TYPE,
p_name OUT employees.last_name%TYPE,
p_salary OUT employees.salary%TYPE,
p_comm OUT
   employees.commission_pct%TYPE)
AUTHID CURRENT_USER
IS
BEGIN
  SELECT last_name, salary,
         commission_pct
   INTO  p_name, p_salary, p_comm
   FROM  employees
   WHERE employee_id=p_id;
END query_employee;
/
```



ORACLE

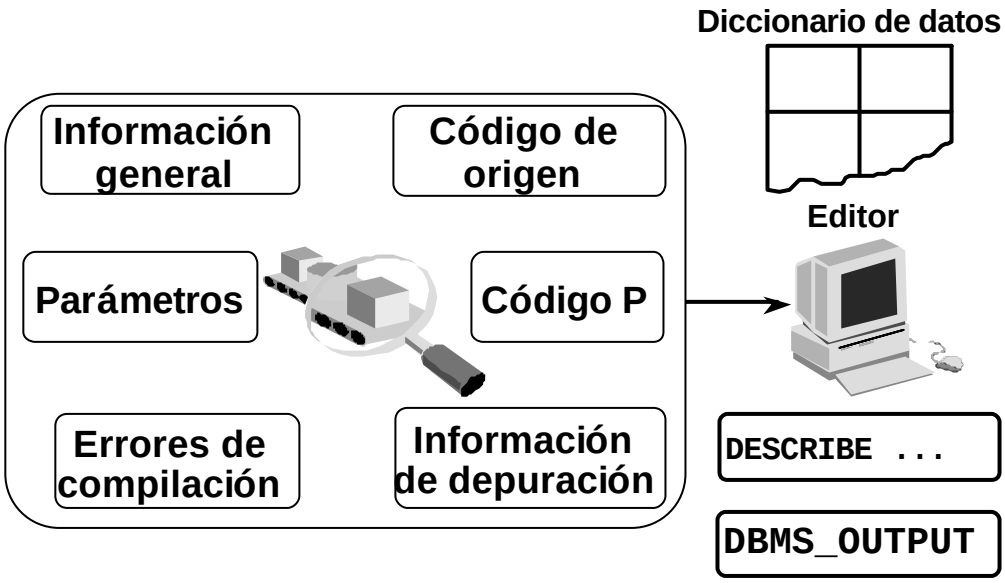
Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Derechos de los Invocadores

Para asegurarse de que el procedimiento se ejecuta utilizando la seguridad del usuario que lo ejecuta, y no del propietario, utilice `AUTHID CURRENT_USER`. Esto garantiza que el procedimiento se ejecuta con los privilegios y el contexto de esquema de su usuario actual.

El comportamiento por defecto, tal y como se muestra en la página anterior, es que el procedimiento se ejecute en el dominio de seguridad del propietario. Pero si quiere establecer explícitamente que el procedimiento se ejecute utilizando los privilegios del propietario, debe usar `AUTHID DEFINER`.

Gestión de Objetos PL/SQL Almacenados



ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Gestión de Objetos PL/SQL Almacenados

Información Almacenada	Descripción	Método de Acceso
General	Información del objeto	La vista del diccionario de datos `USER_OBJECTS`
Código de origen	Texto del procedimiento	La vista del diccionario de datos `USER_SOURCE`
Parámetros	Modo: tipo de dato `IN`/ `OUT`/ `IN OUT`	*i*SQL*Plus: comando `DESCRIBE`
Código P	Código de objeto compilado	No accesible
Errores de compilación	Errores de sintaxis PL/SQL.	La vista del diccionario de datos `USER_ERRORS` *i*SQL*Plus: comando `SHOW ERRORS`
Información de depuración en tiempo de ejecución	Variables y mensajes de depuración especificados por el usuario	Paquete de Oracle `DBMS_OUTPUT`

USER_OBJECTS

Columna	Descripción de la Columna
OBJECT_NAME	Nombre del objeto
OBJECT_ID	Identificador interno del objeto
OBJECT_TYPE	Tipo de objeto como, por ejemplo, TABLE, PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER
CREATED	Fecha de creación del objeto
LAST_DDL_TIME	Fecha de la última modificación del objeto
TIMESTAMP	Fecha y hora de la última recompilación del objeto
STATUS	VALID o INVALID

*Lista de columnas resumida

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Uso de USER_OBJECTS

Para obtener el nombre de todos los objetos PL/SQL almacenados en un esquema, consulte la vista de diccionario de datos USER_OBJECTS.

También puede examinar las vistas ALL_OBJECTS y DBA_OBJECTS, cada una de las cuales contiene la columna OWNER adicional para el propietario del objeto.

Lista de Todas las Funciones y Todos los Procedimientos

```
SELECT object_name, object_type
FROM  user_objects
WHERE object_type in ('PROCEDURE','FUNCTION')
ORDER BY object_name;
```

OBJECT_NAME	OBJECT_TYPE
ADD_DEPT	PROCEDURE
ADD_JOB	PROCEDURE
ADD_JOB_HISTORY	PROCEDURE
ANNUAL_COMP	FUNCTION
DEL_JOB	PROCEDURE
DML_CALL_SQL	FUNCTION
...	
TAX	FUNCTION
UPD_JOB	PROCEDURE
VALID_DEPTID	FUNCTION

24 rows selected.

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Ejemplo

El ejemplo de la transparencia muestra los nombres de todos los procedimientos y todas las funciones que se han creado.

Vista de Diccionario de Datos
USER_SOURCE

Columna	Descripción de la Columna
NAME	Nombre del objeto
TYPE	Tipo de objeto como, por ejemplo, PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY
LINE	Número de línea del código de origen
TEXT	Texto de la línea del código de origen

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Uso de USER_SOURCE

Para obtener el texto de un procedimiento o una función almacenada, utilice la vista de diccionario de datos USER_SOURCE.

Examine también las vistas ALL_SOURCE y DBA_SOURCE, cada una de las cuales contiene la columna OWNER adicional para el propietario del objeto.

Si el archivo de origen no está disponible, puede utilizar *i*SQL*Plus para regenerarlo desde USER_SOURCE.

Lista del Código de los Procedimientos y las Funciones

```
SELECT  text
FROM  user_source
WHERE  name = 'QUERY_EMPLOYEE'
ORDER BY  line;
```

TEXT
PROCEDURE query_employee
(p_id IN employees.employee_id%TYPE, p_name OUT employees.last_name%TYPE,
p_salary OUT employees.salary%TYPE, p_comm OUT employees.commission_pct%TYPE)
AUTHID CURRENT_USER
IS
BEGIN
SELECT last_name, salary, commission_pct
INTO p_name,p_salary,p_comm
FROM employees
WHERE employee_id=p_id;
END query_employee;

11 rows selected.

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Ejemplo

Utilice la vista de diccionario de datos USER_SOURCE para mostrar el texto completo del procedimiento QUERY_EMPLOYEE.

USER_ERRORS

Columna	Descripción de la Columna
NAME	Nombre del objeto
TYPE	Tipo de objeto como, por ejemplo, PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER
SEQUENCE	Número de secuencia, para realizar el ordenamiento
LINE	Número de la línea del código de origen en la que se produce el error
POSITION	Posición de la línea en la que se produce el error
TEXT	Texto del mensaje de error

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Obtención de Errores de Compilación

Para obtener el texto de los errores de compilación, utilice la vista de diccionario de datos USER_ERRORS o el comando SHOW ERRORS de *i*SQL*Plus.

Examine también las vistas ALL_ERRORS y DBA_ERRORS, cada una de las cuales contiene la columna OWNER adicional para el propietario del objeto.

Detección de Errores de Compilación: Ejemplo

```
CREATE OR REPLACE PROCEDURE log_execution
IS
BEGIN
INPUT INTO log_table (user_id, log_date)
                                    -- erróneo
VALUES (USER, SYSDATE);
END;
/
```

```
Warning: Procedure created with compilation errors.
```

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Ejemplo

En el código anterior de `LOG_EXECUTION` se producirá un error de compilación cuando ejecute el archivo de comandos para la compilación.

Lista de los Errores de Compilación Utilizando USER_ERRORS

```
SELECT line || '/' || position POS, text
FROM   user_errors
WHERE  name = 'LOG_EXECUTION'
ORDER BY line;
```

POS	TEXT
4/7	PLS-00103: Encountered the symbol "INTO" when expecting one of the following: := . (@ % ;
5/1	PLS-00103: Encountered the symbol "VALUES" when expecting one of the following: . (, % ; limit The symbol "VALUES" was ignored.
6/1	PLS-00103: Encountered the symbol "END"

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Lista de los Errores de Compilación Utilizando USER_ERRORS

La sentencia SQL anterior es una sentencia SELECT de la vista de diccionario de datos USER_ERRORS, que se utiliza para ver los errores de compilación.

Lista de los Errores de Compilación Utilizando SHOW ERRORS

```
SHOW ERRORS PROCEDURE log_execution
```

Errors for PROCEDURE LOG_EXECUTION:

LINE/COL	ERROR
4/7	PLS-00103: Encountered the symbol "INTO" when expecting one of th e following: := . (@ % ;
5/1	PLS-00103: Encountered the symbol "VALUES" when expecting one of the following: . (, % ; limit The symbol "VALUES" was ignore d.
6/1	PLS-00103: Encountered the symbol "END"

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

SHOW ERRORS

Utilice SHOW ERRORS sin ningún argumento en el prompt de SQL para obtener los errores de compilación del último objeto que haya compilado.

También puede utilizar este comando con una unidad de programa específica. La sintaxis es la siguiente:

```
SHOW ERRORS [{FUNCTION|PROCEDURE|PACKAGE|PACKAGE
             BODY|TRIGGER|VIEW} [esquema.]nombre]
```

Con el comando SHOW ERRORS, sólo puede ver los errores de compilación que genera la última sentencia que se ha utilizado para crear un subprograma. La vista de diccionario de datos USER_ERRORS almacena todos los errores de compilación que se han generado durante la creación de los subprogramas.

DESCRIBE en *i*SQL*Plus

```
DESCRIBE query_employee
DESCRIBE add_dept
DESCRIBE tax
```

PROCEDURE QUERY_EMPLOYEE

Argument Name	Type	In/Out	Default?
P_ID	NUMBER(6)	IN	
P_NAME	VARCHAR2(25)	OUT	
P_SALARY	NUMBER(8,2)	OUT	
P_COMM	NUMBER(2,2)	OUT	

PROCEDURE ADD_DEPT

Argument Name	Type	In/Out	Default?
P_NAME	VARCHAR2(30)	IN	DEFAULT
P_LOC	NUMBER(4)	IN	DEFAULT

FUNCTION TAX RETURNS NUMBER

Argument Name	Type	In/Out	Default?
P_VALUE	NUMBER	IN	

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Descripción de Procedimientos y Funciones

Para mostrar un procedimiento o una función y su lista de parámetros, utilice el comando `DESCRIBE` de *i*SQL*Plus.

Ejemplo

El código de la transparencia muestra la lista de parámetros de los procedimientos `QUERY_EMPLOYEE` y `ADD_DEPT` y de la función `TAX`.

Tenga en cuenta la lista de parámetros del procedimiento `ADD_DEPT`, que incluye valores por defecto. La columna `DEFAULT` sólo indica que hay un valor por defecto; no proporciona el valor real.

Depuración de Unidades de Programa PL/SQL

- **El paquete `DBMS_OUTPUT`:**
  - **Acumula información en un buffer**
  - **Permite recuperar información del buffer**
- **Llamadas de procedimiento autónomas (por ejemplo, escribir el resultado en un tabla de log)**
- **Software que utiliza `DBMS_DEBUG`**
  - **Procedure Builder**
  - **Software de depuración de terceros**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Depuración de Unidades de Programa PL/SQL

La transparencia muestra los diferentes paquetes que se pueden utilizar para depurar unidades de programa PL/SQL.

Puede utilizar procedimientos `DBMS_OUTPUT` empaquetados para extraer valores y mensajes desde un bloque PL/SQL. Para ello, se acumula información en un buffer y luego se permite la recuperación de la información desde ese buffer. `DBMS_OUTPUT` es un paquete de Oracle. Para cualificar todas las referencias a estos procedimientos, se utiliza el prefijo `DBMS_OUTPUT`.

Ventajas del Uso del Paquete `DBMS_OUTPUT`

Este paquete permite a los desarrolladores seguir de cerca la ejecución de una función o un procedimiento enviando mensajes y valores al buffer de salida. En *i*SQL*Plus, utilice `SET SERVEROUTPUT ON` u `OFF` en lugar del procedimiento `ENABLE` o `DISABLE`.

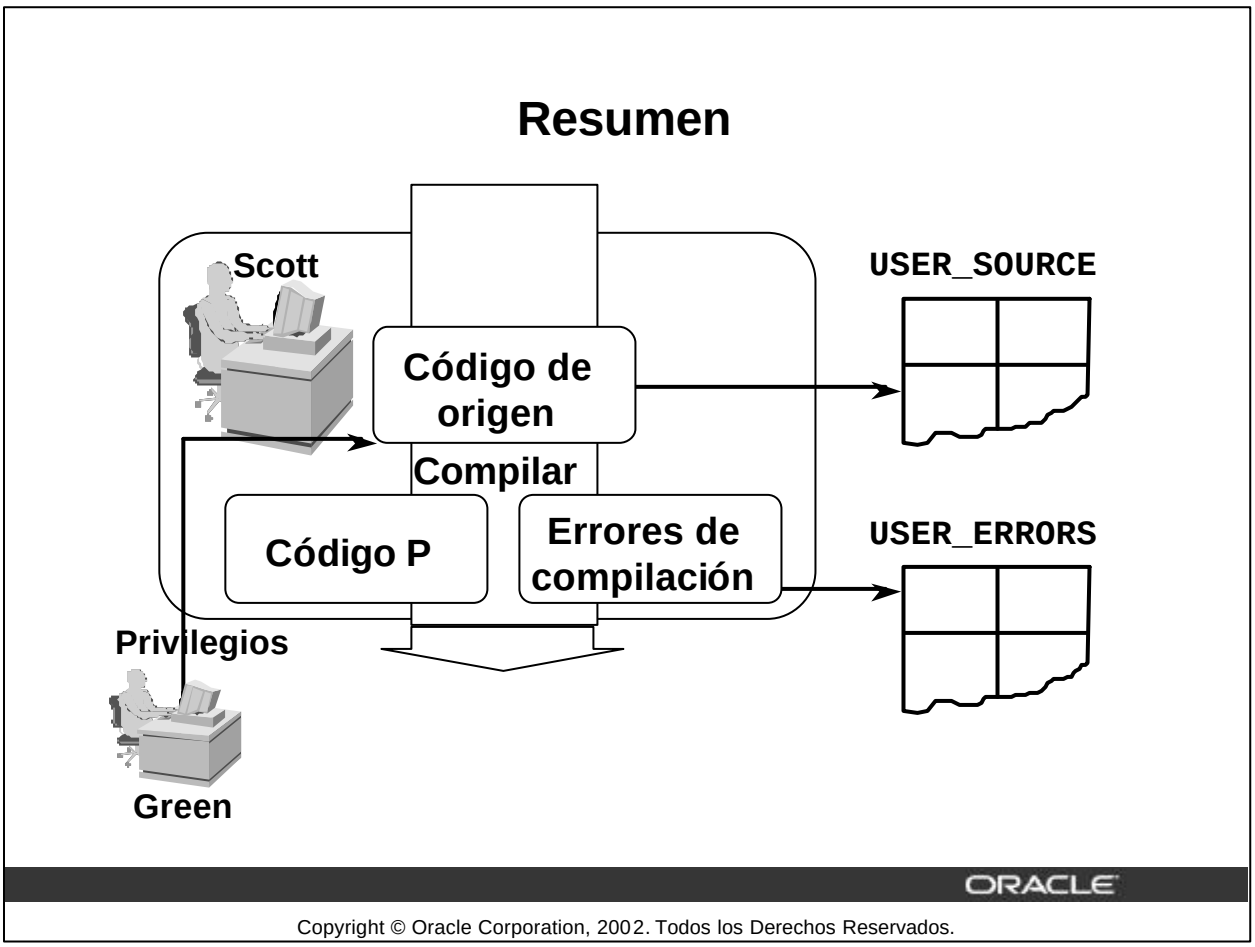
Información de Diagnóstico Sugerida

- Mensaje al entrar, al salir de un procedimiento o que indique que se ha realizado una operación
- Contador de un bucle
- Valor de una variable antes y después de una asignación

Nota: El buffer no se vacía hasta que el bloque no termina.

Puede depurar subprogramas especificando llamadas de procedimientos autónomos y almacenar el resultado como valores de columnas en una tabla de log.

En el Apéndice C se explica la depuración con Oracle Procedure Builder. Procedure Builder utiliza un paquete de depuración especificado por Oracle que se denomina `DBMS_DEBUG`.



Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

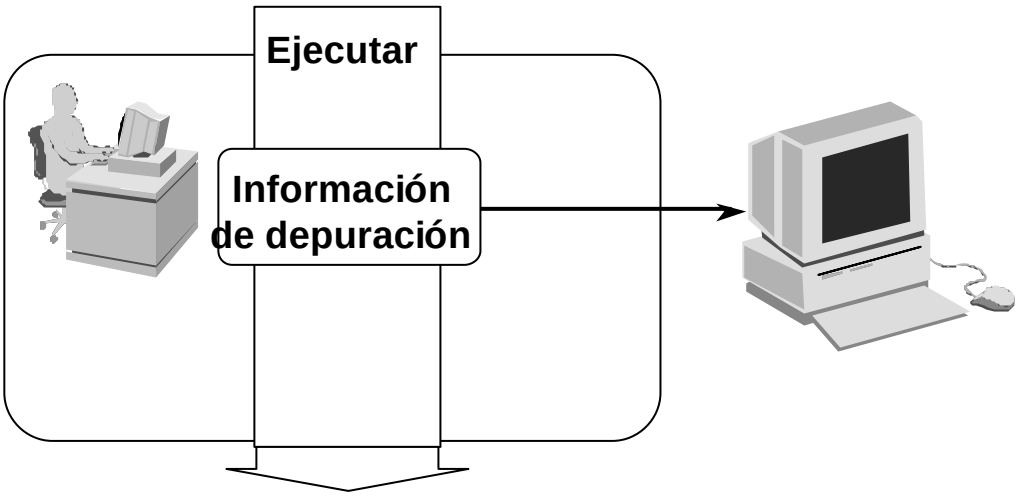
Resumen

Los usuarios deben tener los privilegios necesarios para acceder a los objetos de base de datos mediante un subprograma.

Aproveche las diversas vistas de diccionario de datos, los comandos SQL, los comandos de *i*SQL*Plus y los procedimientos de Oracle para gestionar una función o un procedimiento almacenado durante su ciclo de desarrollo.

Nombre	Comando o Vista de Diccionario de Datos	Descripción
USER_OBJECTS	Vista de diccionario de datos	Proporciona información general acerca del objeto
USER_SOURCE	Vista de diccionario de datos	Proporciona el texto del objeto (es decir, el bloque PL/SQL)
DESCRIBE	Comando de *i*SQL*Plus	Proporciona la declaración del objeto
USER_ERRORS	Vista de diccionario de datos	Muestra los errores de compilación
SHOW ERRORS	Comando de *i*SQL*Plus	Muestra los errores de compilación, por procedimiento o por función
DBMS_OUTPUT	Paquete de Oracle	Proporciona la depuración especificada por el usuario, con mensajes y valores de las variables
GRANT	Comando de *i*SQL	Proporciona privilegios de seguridad al propietario que crea el procedimiento y al usuario que lo ejecuta, permitiéndoles realizar sus respectivas operaciones

Resumen



ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Resumen (continuación)

- Consulte el diccionario de datos.
  - Obtenga una lista de los procedimientos y las funciones utilizando la vista `USER_OBJECTS`.
  - Obtenga una lista del texto de determinados procedimientos o funciones utilizando la vista `USER_SOURCE`.
- Prepare un procedimiento: Vuelva a crear un procedimiento y haga que los errores de compilación se muestren automáticamente.
- Pruebe un procedimiento: Pruebe un procedimiento suministrando valores de entrada; pruebe un procedimiento o una función mostrando el resultado o los valores de retorno.

Visión General de la Práctica 11

Esta práctica cubre los siguientes temas:

- **Volver a crear el archivo de origen de un procedimiento**
- **Volver a crear el archivo de origen de una función**

ORACLE

Copyright © Oracle Corporation, 2002. Todos los Derechos Reservados.

Visión General de la Práctica 11

En esta práctica tendrá que volver a crear el código de origen de un procedimiento y una función.

Práctica 11

Suponga que ha perdido el código del procedimiento `NEW_EMP` y de la función `VALID_DEPTID` que creó en la lección 10. (Si no ha realizado las prácticas de la lección 10, puede ejecutar los archivos de comandos de la solución para crear el procedimiento y la función.)

Cree un archivo de spool de *i*SQL*Plus para consultar la vista de diccionario de datos adecuada para regenerar el código.

Pista:

```
SET                 -- opciones ON|OFF
SELECT              -- sentencia(s) para extraer el código
SET                 -- restablecer opciones ON|OFF
```

Para enviar el resultado del archivo a un archivo `.sql` desde *i*SQL*Plus, seleccione la opción `Save` para `Output` y ejecute el código.