INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Distributed DataBases

2do Examen

Alumno:  
-Fuentes Mora Roberto Javier

Pofesor

Luis Salinas

**Distributed Database.**

**2do Examen**

Contenido

[11. Introducción a cursores 2](#_Toc515441575)

[11.1. Manipulación de cursores 2](#_Toc515441576)

[11.1.1. Make Use of Record Types 7](#_Toc515441577)

[11.1.2. Procesar un cursor explicito 8](#_Toc515441578)

[11.1.3. Usando los atributos del cursor 11](#_Toc515441579)

[11.1.4. Ponerlo todo junto 12](#_Toc515441580)

[11.2. Uso del cursor FOR Loops y cursores anidados 14](#_Toc515441581)

[11.2.1. Uso del cursor FOR Loop 14](#_Toc515441582)

[11.2.2. Procesar cursores anidados 15](#_Toc515441583)

[12. Cursores avanzados 18](#_Toc515441584)

[12.1. Usar parámetros con cursores y cursores anidados complejos 18](#_Toc515441585)

[12.1.1. Uso de parámetros en un cursor 19](#_Toc515441586)

[12.1.2. Uso de cursores anidados complejos 20](#_Toc515441587)

[12.2. Cursores FOR UPDATE y WHERE CURRENT 24](#_Toc515441588)

[12.2.1. Cursores FOR UPDATE y WHERE CURRENT 24](#_Toc515441589)

[13. Triggers 29](#_Toc515441590)

[13.1. Que son los triggers 31](#_Toc515441591)

[13.1.1. Entendiendo que es un trigger. 36](#_Toc515441592)

[13.1.2. Uso de BEFORE y AFTER Triggers 37](#_Toc515441593)

[13.2. Tipos de triggers 38](#_Toc515441594)

[13.2.1. Uso de triggers de filas y de declaraciones 40](#_Toc515441595)

[13.2.2. Uso de triggers INSTEAD OF 41](#_Toc515441596)

[19. Procedimientos 43](#_Toc515441597)

[19.1. Creando procedimientos 44](#_Toc515441598)

[19.1.1. Creación de procedimientos 45](#_Toc515441599)

[19.1.2. Consultar el diccionario de datos para obtener información sobre los procedimientos 46](#_Toc515441600)

[19.2. Paso de parámetros dentro y fuera de los procedimientos 47](#_Toc515441601)

[19.2.1. Uso de los parámetros IN y OUT con los procedimientos 49](#_Toc515441602)

[20. Funciones 52](#_Toc515441603)

[20.1. Creación y uso de funciones 52](#_Toc515441604)

[20.1.1. Crear funciones almacenadas 53](#_Toc515441605)

[20.1.2. Haciendo uso de funciones 54](#_Toc515441606)

[20.1.3. Funciones de invocación en sentencias SQL 54](#_Toc515441607)

[20.1.4. Escribiendo funciones complejas 56](#_Toc515441608)

[21. Paquetes 58](#_Toc515441609)

# 11. Introducción a cursores

Los cursores son áreas de memoria donde Oracle ejecuta sentencias de SQL. En la base de datos, los cursores de programación son estructuras internas de datos que permiten el procesamiento de resultados de consultas SQL. Por ejemplo, utiliza un cursor para operar en todas las filas de la tabla STUDENT para los estudiantes que están tomando un curso en particular (que tiene entradas asociadas en la tabla de ENROLLMENT).

## 11.1. Manipulación de cursores

Para que Oracle procese una declaración SQL, necesita crear un área de memoria conocida como área de contexto; esto tendrá la información necesaria para procesar la declaración. Esta información incluye el número de filas procesadas por la declaración y un puntero a la representación analizada de la declaración. (El análisis de una declaración SQL es el proceso mediante el cual la información se transfiere al servidor, momento en el cual la declaración SQL se evalúa como válida). En una consulta, el conjunto activo se refiere a las filas que se devuelven.

Un cursor *es un asa,* o un puntero, en el área de contexto. A través del cursor, **un programa PL/SQL puede controlar el área de contexto y lo que le sucede a medida que se procesa la declaración**. Los cursores tienen dos características importantes:

* Los cursores le permiten buscar y procesar filas devueltas por una instrucción SELECT una fila a la vez.
* Un cursor se llama para que se pueda referenciar.

Hay **dos tipos de cursores:**

* Oracle declara automáticamente un **cursor implícito** cada vez que se ejecuta una instrucción SQL. El usuario desconoce esto y no puede controlar ni procesar la información en un cursor implícito.
* El programa define un **cursor explícito** para cualquier consulta que devuelva más de una fila de datos. Esto significa que el programador ha declarado el cursor dentro del bloque de código PL/SQL. Esta declaración permite que la aplicación procese secuencialmente cada fila de datos a medida que el cursor la devuelve.

**CURSOR IMPLICITO**

Para comprender mejor las capacidades de un cursor explícito, primero necesita comprender el proceso de un cursor implícito:

* Cualquier bloque PL/SQL dado emite un cursor implícito cada vez que se ejecuta una instrucción SQL, siempre que no exista un cursor explícito para esa instrucción SQL.
* Un cursor se asocia automáticamente con cada declaración DML (manipulación de datos) (UPDATE, DELETE, INSERT)
* Todas las instrucciones UPDATE y DELETE tienen cursores que identifican el conjunto de filas que se verán afectadas por la operación.
* Una instrucción INSERT necesita un lugar para recibir los datos que se insertarán en la base de datos; el cursor implícito cumple con esta necesidad.
* El cursor abierto más recientemente se llama cursor SQL.

El cursor implícito se utiliza para procesar instrucciones INSERT, UPDATE, DELETE y SELECT INTO. Durante el procesamiento de un cursor implícito, Oracle realiza automáticamente OPEN, FETCH y operaciones CLOSE.

Por cierto, un cursor implícito puede indicar cuántas filas se vieron afectadas por una actualización. Los cursores tienen atributos como ROWCOUNT. SQL% ROWCOUNT devuelve el número de filas actualizadas. Se puede usar de la siguiente manera:

SET SERVEROUTPUT ON

BEGIN

UPDATE student

SET first\_name = 'B'

WHERE first\_name LIKE 'B%';

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT);

END;

Consideremos el siguiente ejemplo de un cursor implícito:

1. SET SERVEROUTPUT ON;
2. DECLARE
3. v\_first\_name VARCHAR2(35);
4. v\_last\_name VARCHAR2(35);
5. BEGIN
6. SELECT first\_name, last\_name
7. INTO v\_first\_name, v\_last\_name
8. FROM student
9. WHERE student\_id = 123;
10. DBMS\_OUTPUT.PUT\_LINE ('Student name: '||
11. v\_first\_name||' '||v\_last\_name);
12. EXCEPTION
13. WHEN NO\_DATA\_FOUND THEN
14. DBMS\_OUTPUT.PUT\_LINE
15. ('There is no student with student ID 123');
16. END;

Oracle asocia automáticamente un cursor implícito con la instrucción SELECT INTO y obtiene los valores para las variables *v\_first\_name* y *v\_last\_name*. Después de que la instrucción SELECT INTO se complete, Oracle cierra el cursor implícito.

A diferencia de un cursor implícito, el programa define un cursor explícito para cualquier consulta que devuelva más de una fila de datos. Para procesar un cursor explícito, primero lo declara y luego lo abre. Luego lo buscas y finalmente lo cierras.

**CURSOR EXPLICITO**

La única forma de generar un cursor explícito es nombrar el cursor en la sección DECLARE del bloque PL / SQL.

La ventaja de declarar un cursor explícito sobre un cursor implícito indirecto es que el cursor explícito otorga al programador un control más programático. Además, los cursores implícitos son menos eficientes que los cursores explícitos, por lo que es más difícil atrapar los errores de datos.

El proceso de trabajar con un cursor explícito consta de los siguientes pasos:

1. **Declarando** el cursor. Esto inicializa el cursor en la memoria.
2. **Apertura** del cursor. El cursor declarado se abre y la memoria se asigna.
3. **Obtener** (Fetching) el cursor. El cursor declarado y abierto ahora puede recuperar datos.
4. **Cerrar** el cursor. El cursor declarado, abierto y captado debe estar cerrado para liberar la asignación de memoria.

**DECLARANDO UN CURSOR**

Declarar un cursor define el nombre del cursor y lo asocia con una instrucción SELECT. Usted declara un cursor usando la siguiente sintaxis:

CURSOR c\_cursor\_name IS select statement

Se recomienda que siempre comience el nombre del cursor con **c\_.** Cuando lo haga, siempre estará claro que el nombre se refiere a un cursor.

No puede usar un cursor a menos que se haya realizado el ciclo completo de declarar, abrir, recuperar y cerrar. Para explicar estos cuatro pasos, los siguientes ejemplos muestran fragmentos de código para cada paso. Después de eso, se le muestra el proceso completo.

DECLARE

CURSOR c\_MyCursor IS

SELECT \* FROM zipcode WHERE state = 'NY';

...

-- code would continue here with opening, fetching -- and closing of the cursor>

Este fragmento PL / SQL demuestra **el primer paso** para declarar un cursor. Un cursor llamado c\_MyCursor se declara como una instrucción de selección de todas las filas en la tabla de código postal que tienen el estado del elemento igual a NY.

Por cierto, los nombres del cursor siguen las mismas reglas de alcance y visibilidad que se aplican a los identificadores PL / SQL. Como el nombre del cursor es un identificador PL / SQL, debe declararse antes de que se haga referencia a él. Cualquier instrucción de selección válida se puede usar para definir un cursor, incluidas combinaciones y declaraciones con la UNION o Cláusula MINUS.

**TIPOS DE REGISTRO**

Un registro es una estructura de datos compuesta, lo que significa que está compuesto por uno o más elementos. Los registros son muy parecidos a una fila de una tabla de base de datos, pero cada elemento del registro no se sostiene por sí mismo.

PL / SQL admite tres tipos de registros: **basado en tabla, basado en cursor y definido por programador**. Un registro basado en tablas es aquel cuya estructura se extrae de la lista de columnas de la tabla. Un registro basado en cursor es aquel cuya estructura coincide con los elementos de un cursor predefinido. Para crear un registro basado en tabla o basado en cursor, use el atributo %ROWTYPE:

record\_name table\_name o cursor\_name% ROWTYPE

Por ejemplo:

-- ch11\_1a.sql

SET SERVEROUTPUT ON

DECLARE

vr\_student student%ROWTYPE;

BEGIN

SELECT \*

INTO vr\_student FROM student WHERE student\_id = 156;

DBMS\_OUTPUT.PUT\_LINE (vr\_student.first\_name||' ' ||vr\_student.last\_name||' has an ID of 156');

EXCEPTION

WHEN no\_data\_found THEN RAISE\_APPLICATION\_ERROR(-2001,'The Student '|| 'is not in the database');

END;

La variable vr\_student es un tipo de registro del alumno de tabla de base de datos existente. En otras palabras, tiene los mismos componentes que una fila en la tabla de estudiantes. Un registro basado en cursor es muy similar, excepto que se extrae de la lista de selección de un cursor explícitamente declarado. Al hacer referencia a elementos del registro, usa la misma sintaxis que usa con las tablas:

record\_name.item\_name

Para definir una variable que se basa en un registro de cursor, primero debe declarar el cursor. En la siguiente práctica de laboratorio, comenzará declarando un cursor y luego abra el cursor, extraiga del cursor y cierre el cursor. Un registro basado en tabla se extrae de una estructura de tabla particular. Considere el siguiente fragmento de código:

Por ejemplo:

DECLARE

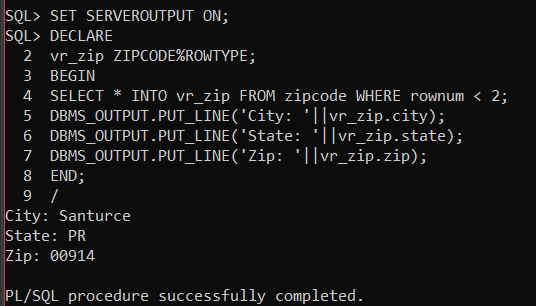
vr\_zip ZIPCODE%ROWTYPE;

vr\_instructor INSTRUCTOR%ROWTYPE;

El registro vr\_zip tiene una estructura similar a una fila de la tabla ZIPCODE. Sus elementos son CITY, STATE and ZIP. Es importante tener en cuenta que si la columna CITY de la tabla ZIPCODE se ha definido como VARCHAR2 (15), el atributo CITY del registro vr\_zip tiene la misma estructura de tipo de datos. Del mismo modo, el registro vr\_instructor se basa en la fila de la tabla INSTRUCTOR.

### 11.1.1. Make Use of Record Types

Se muestra un ejemplo de tipos de registro en un bloque anónimo PL/SQL:

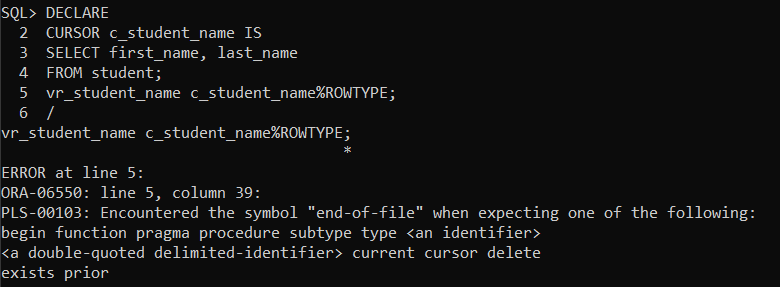


¿Qué está sucediendo?

En este ejemplo, selecciona una sola fila para la tabla ZIPCODE en el registro *vr\_zip*. A continuación, muestra cada elemento del registro en la pantalla. Se observa que para hacer referencia a cada atributo del registro, se utiliza la notación de puntos.

Un registro basado en cursor se basa en la lista de elementos de un cursor predefinido

Aquí hay otro ejemplo



¿Qué esta sucediendo?

El registro vr\_student\_name tiene una estructura similar a una fila devuelta por una declaración SELECT definida en el cursor. Contiene dos atributos: el nombre y apellido del student. Es importante tener en cuenta que un registro basado en cursor puede declararse solo después de su correspondiente cursor ha sido declarado; de lo contrario, se producirá un error de compilación como en este caso.

### 11.1.2. Procesar un cursor explicito

Para utilizar un cursor, debe hacer uso del ciclo completo de declarar, abrir, recuperar y cerrar el cursor (declaring, opening, fetching and closing the cursor)

Por ejemplo:

1. Escribe la sección de declaración de un bloque PL/SQL. Debe definir un cursor llamado c\_student basado en la tabla de estudiantes, con last\_name y first\_name concatenado en un elemento llamado name.También debe omitir las columnas created\_by y modified\_by. Entonces declara un registro basado en este cursor.

**Abriendo un cursor**

El siguiente paso para controlar un cursor explícito es abrirlo. Cuando se procesa la instrucción de cursor OPEN, las siguientes cuatro acciones tienen lugar automáticamente:

1. Se examinan las variables (incluidas las variables de vinculación) en la cláusula WHERE.
2. En función de los valores de las variables, se determina el conjunto activo y se ejecuta el motor PL/SQL la consulta para ese cursor. Las variables se examinan solo en el tiempo de apertura del cursor.
3. El motor PL/SQL identifica el conjunto activo de datos: las filas de todas las tablas involucradas que cumplir con los criterios de la cláusula WHERE.
4. El puntero del conjunto activo se establece en la primera fila.

La sintaxis para abrir un cursor es:

OPEN cursor\_name;

Entonces, el siguiente paso en el procesamiento de un cursor explicito es:

1. Agregue las líneas necesarias al bloque PL/SQL que acaba de escribir para abrir el cursor.

**Recuperando filas en cursor**

Después de que el cursor ha sido declarado y abierto, puede recuperar datos del cursor. El proceso de obtener datos del cursor se llama recuperar el cursor. Hay dos formas de obtener un cursor:

FETCH cursor\_name INTO PL/SQL variables;

O

FETCH cursor\_name INTO PL/SQL record;

Cuando se busca el cursor, ocurre lo siguiente:

1. El comando FETCH se usa para recuperar una fila a la vez del conjunto activo. Esto es generalmente hecho dentro de un bucle. Los valores de cada fila en el conjunto activo se pueden almacenar en el correspondiente las variables o PL / SQL graban una a la vez, realizando operaciones en cada una de ellas sucesivamente.
2. Después de cada FETCH, el puntero del conjunto activo se mueve hacia adelante a la siguiente fila. Por lo tanto, cada FETCH devuelve filas sucesivas del conjunto activo, hasta que se devuelva todo el conjunto. El último FETCH no asigna valores a las variables de salida; todavía contienen sus valores anteriores.
3. El siguiente paso es agregar un bloque al bloque PL/SQL. Dentro del ciclo, recuperar (fetch) el cursor dentro del al registro. Incluir una línea DBMS\_OUTPUT dentro del bucle de modo que cada vez que el bucle se repita, toda la información en el registro se muestre en una sesión SQL \* Plus.

**Cerrando un cursor**

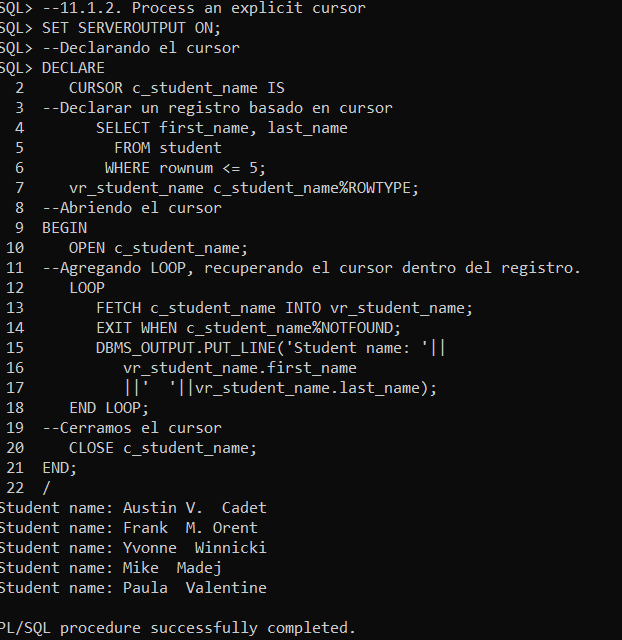
Tan pronto como todas las filas en el cursor hayan sido procesadas (recuperadas), el cursor debería estar cerrado. Esto le dice al motor PL / SQL que el programa ha terminado con el cursor, y los recursos asociados con él pueden liberarse. La sintaxis para cerrar el cursor es:

CLOSE cursor\_name;

Por cierto después de cerrar un cursor, ya no puede recuperarlo. Del mismo modo, no es posible cerrar un cursor ya cerrado. Intentar realizar cualquiera de estas acciones daría lugar a un error de Oracle.

1. Se agrega la instrucción CLOSE para cerrar el cursor.

A continuación se muestra un ejemplo donde se ejecutan todos los incisos previos



¿Qué esta sucediendo?

En este ejemplo, se declara un cursor que devuelve cinco nombres de estudiante. Luego, se declara un registro basado en cursor. En el cuerpo del programa, procesa cursores explícitos a través del bucle del cursor. En el cuerpo del ciclo, asigna cada registro devuelto por el cursor al registro basado en cursor, *vr\_student\_name.*

### 11.1.3. Usando los atributos del cursor

La siguiente tabla enumera los atributos de un cursor, que determinan el resultado de una operación del cursor cuando se busca o abre.

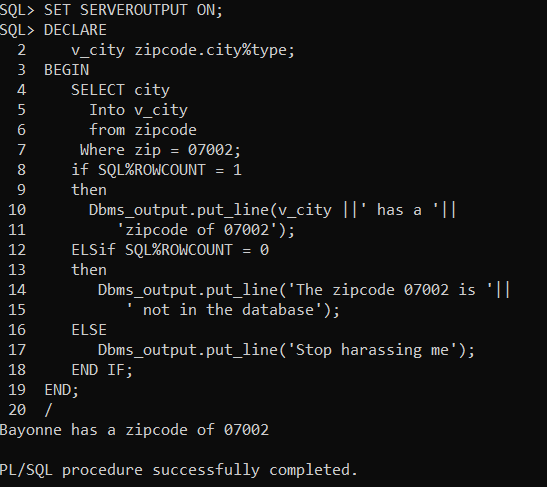
|  |  |  |
| --- | --- | --- |
| Atributos del cursor | Sintaxis | Descripción |
| %NOTFOUND | cursor\_name%NOTFOUND | Un atributo booleano que devuelve TRUE si el FETCH anterior no devolvió una fila y FALSE si lo hizo. |
| %FOUND | cursor\_name%FOUND | Un atributo booleano que devuelve TRUE si  el FETCH anterior devolvió una fila y  FALSO si no fuera |
| %ROWCOUNT | cursor\_name%ROWCOUNT | La cantidad de registros obtenidos de un  cursor en ese punto en el tiempo |
| %ISOPEN | cursor\_name%ISOPEN | Un atributo booleano que devuelve TRUE si  el cursor está abierto y FALSO si no es así. |

Ahora que conocemos estos atributos, podemos usar uno de ellos para salir del bucle dentro del código anterior.

Los atributos del cursor se pueden usar con cursores implícitos utilizando el prefijo SQL, como SQL% ROWCOUNT.

Si usamos la sintaxis SELECT INTO en su bloque PL / SQL, creará un cursor implícito. Se puede usar luego estos atributos en el cursor implícito.

Por ejemplo:

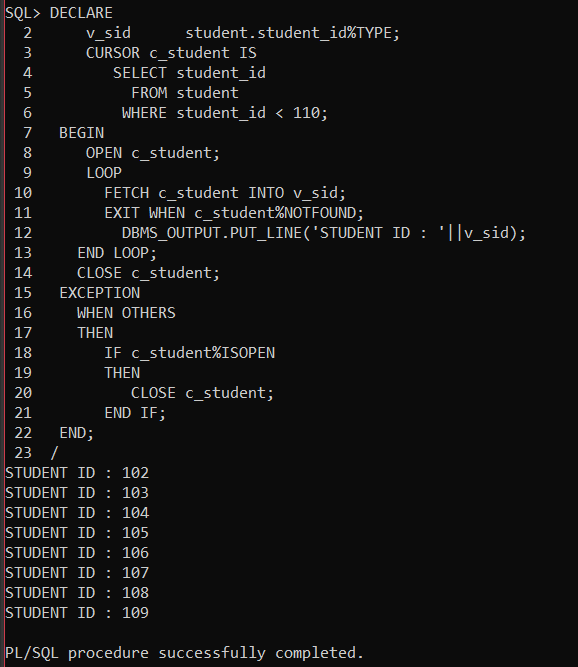


¿Qué esta sucediendo?

La sección de declaración declara una variable, v\_city, anclada al tipo de datos del elemento de ciudad en la tabla de código postal. La instrucción SELECT hace que un cursor implícito sea abierto, recuperado y luego cerrado. La cláusula IF utiliza el atributo% ROWCOUNT para determinar si el cursor implícito tiene un recuento de filas de 1. Si lo hace, se muestra la primera línea DBMS\_OUTPUT. Tenga en cuenta que este ejemplo no maneja una situación en la que el conteo de filas es mayor que 1. Debido a que la clave primaria de la tabla de códigos postales es el código postal, esto podría suceder.

### 11.1.4. Ponerlo todo junto

En el siguiente ejemplo se hace uso del ciclo completo: declarar, abrir, recuperar y cerrar el cursor, incluyendo el uso de los atributos del cursor.



¿Qué esta sucediendo?

Este ejemplo ilustra un bucle de búsqueda del cursor, en el que se devuelven varias filas de datos de la consulta. El cursor se declara en la sección de declaración del bloque (líneas 1 a 6), al igual que otros identificadores. En la sección ejecutable del bloque (líneas 7 a 15), se abre un cursor usando la instrucción OPEN (línea 8). Debido a que el cursor devuelve varias filas, se utiliza un bucle para asignar datos devueltos a las variables con una instrucción FETCH (línea 10). Como la instrucción de ciclo no tiene otro medio de terminación, se debe especificar una condición de salida. En este caso, uno de los atributos del cursor es% NOTFOUND (línea 11). El cursor se cierra para liberar la asignación de memoria (línea 14). Además, si se llama al manejador de excepciones, hay una comprobación para ver si el cursor está abierto (línea 18). Si es así, está cerrado (línea 20).

## 11.2. Uso del cursor FOR Loops y cursores anidados

### 11.2.1. Uso del cursor FOR Loop

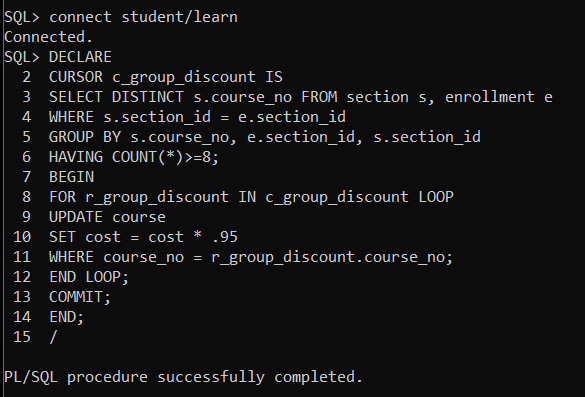
Hay una forma alternativa de manejar los cursores. Se denomina cursor FOR debido a la sintaxis simplificada que se utiliza. Con un cursor FOR, el proceso de abrir, recuperar y cerrar se maneja implícitamente. Esto hace que los bloques sean mucho más fáciles de codificar y mantener.

El cursor del bucle FOR especifica una secuencia de instrucciones que se repetirán una vez por cada fila devuelta por el cursor. Use el cursor FOR si necesita FETCH y PROCESE cada registro desde un cursor hasta que desee detener el procesamiento y salir del ciclo.

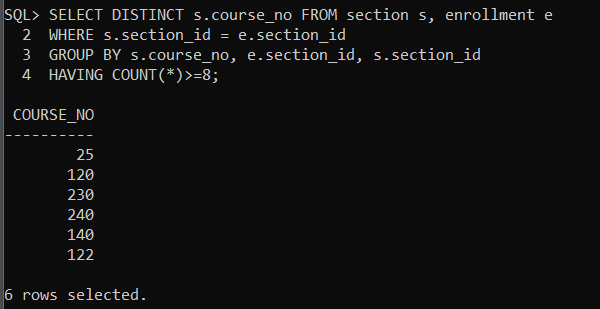
Para usar esta columna, necesita crear una nueva tabla llamada table\_log con la siguiente secuencia de comandos:

Por ejemplo:

El siguiente ejercicio consiste en escribir un bloque PL/SQL que reduzca el costo de todos los cursos en un 5 % para los cursos que tienen una inscripción de ocho estudiantes o mas. Se utiliza un cursor FOR LOOP que actualiza la tabla del curso.



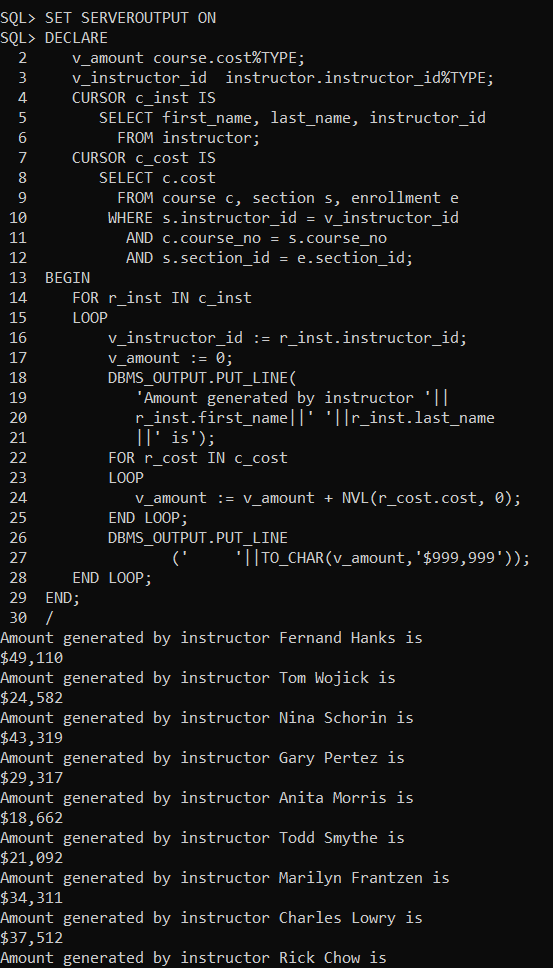
El cursor c\_group\_discount se declara en la sección de declaración. El SQL apropiado se usa para generar la instrucción SELECT para responder la pregunta dada. El cursor se procesa en un ciclo FOR. En cada iteración del ciclo, se ejecuta la instrucción de actualización SQL. Esto significa que no tiene que abrirse, buscarse y cerrarse. Esto también significa que no se debe usar un atributo de cursor para crear una condición de salida para el bucle que está procesando el cursor.

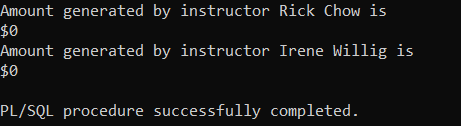


### 11.2.2. Procesar cursores anidados

Los cursores se pueden anidar uno dentro del otro. Aunque esto puede sonar complejo, en realidad es solo un bucle dentro de un bucle, muy parecido a los bucles anidados, que se trataron en los capítulos anteriores. Si tiene un cursor principal y dos cursores secundarios, cada vez que el cursor padre hace un solo bucle, recorre cada cursor secundario una vez y luego comienza una segunda ronda.

El siguiente ejemplo muestra un cursor anidado con un solo cursor secundario.





Este ejemplo tiene dos cursores. El primero es un cursor de los códigos postales, y el segundo es una lista de estudiantes. La variable v\_zip se inicializa en la línea 16 para ser el código postal del registro actual del cursor c\_zip. El cursor c\_ student enlaza el cursor c\_zip por medio de esta variable. Por lo tanto, cuando el cursor se procesa en las líneas 20 a 26, está recuperando estudiantes que tienen el código postal del registro actual para el cursor principal. El cursor padre se procesa desde las líneas 13 a 31. Cada iteración del cursor padre ejecutará DBMS\_OUTPUT en las líneas 16 y 17 solo una vez. El DBMS\_OUTPUT en la línea 22 se ejecutará una vez para cada iteración del ciclo secundario, produciendo una línea de salida para cada alumno. La instrucción DBMS\_OUTPUT.PUT\_LINE en la línea 29 solo se ejecutará si el ciclo interno no se ejecutó. Esto se logró estableciendo una variable v\_student\_flag. La variable se establece en N en el comienzo del ciclo principal. Si el ciclo secundario se ejecuta al menos una vez, la variable se establece en Y. Después de que el ciclo secundario se haya cerrado, se realiza una comprobación con una instrucción IF para determinar el valor de la variable. Si todavía es N, se puede concluir con seguridad que el lazo interno no se procesó. Esto permitirá que se ejecute la última instrucción DBMS\_OUTPUT.PUT\_LINE. Los cursores anidados se parametrizan con mayor frecuencia.

**Conclusion**

En el capitulo 11 pude comprender el uso y la implementación de los cursores, desde los diferentes tipos , la forma de declararlos tanto implicta como explícitamente , acceder y hacer usos de los atributos del mismo , la importancia de un LOOP , y de Fecth.

Ademas de declarar un cursor explícito que permite a un usuario procesar muchas filas devueltas por una consulta y le permite al usuario escribir código que procesa cada fila de a una por vez

Logre comprender las ventajas de usar  cursores

* Se pueden procesar los datos fila por fila
* El procesamiento se hace normalmente sobre las filas que cumplan con una condición dada
* Los cursors permiten el procesamiento fila por fila
* Se pueden modificar los datos fila por fila

# 12. Cursores avanzados

## 12.1. Usar parámetros con cursores y cursores anidados complejos

**CURSORES CON PARAMETROS**

Un cursor se puede declarar con parámetros. Esto permite que un cursor genere un conjunto de resultados específico que sea reducido pero también reutilizable.

Un cursor de todos los datos de la tabla de código postal puede ser muy útil, pero sería más útil para cierto procesamiento de datos si tuviera información para un solo estado. Hasta este punto sabemos cómo crear dicho cursor. Pero, ¿no sería más útil si pudieras crear un cursor que pudiera aceptar un parámetro de un estado y luego correr solo a través de la ciudad y zip para ese estado?

Por ejemplo

CURSOR c\_zip (p\_state IN zipcode.state%TYPE) IS

SELECT zip, city, state

FROM zipcode

WHERE state = p\_state;

Estos son los puntos principales a tener en cuenta para los parámetros en los cursores:

* Los parámetros del cursor hacen que el cursor sea más reutilizable.
* A los parámetros del cursor se les pueden asignar valores predeterminados.
* El alcance de los parámetros del cursor es local al cursor.
* El modo de los parámetros solo puede ser IN.

Cuando un cursor se ha declarado que toma un parámetro, se debe invocar con un valor para ese parámetro. El cursor c\_zip declarado en el ejemplo anterior se llama de la siguiente manera:

OPEN c\_zip (parameter\_value)

El mismo cursor puede ser abierto con un CURSOR FOR LOOP de la siguiente manera:

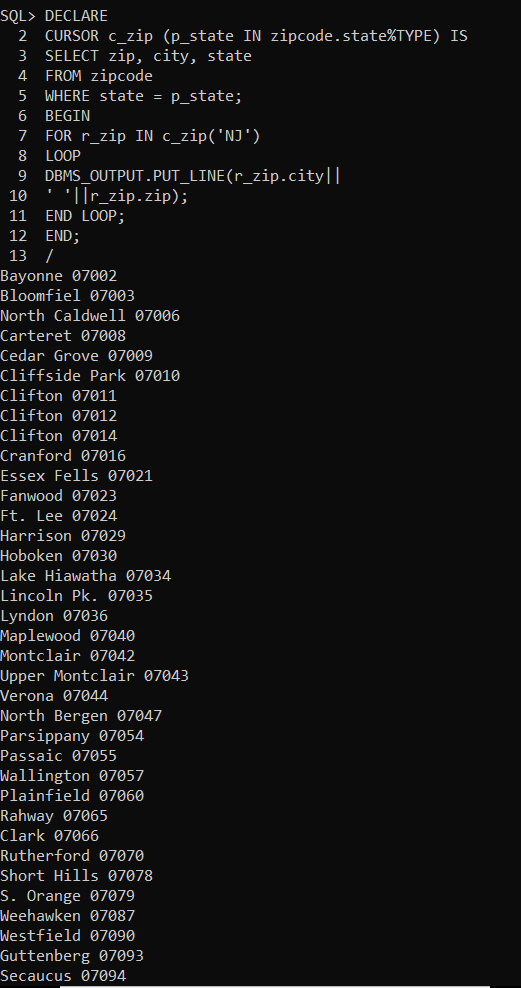
FOR r\_zip IN c\_zip('NY')

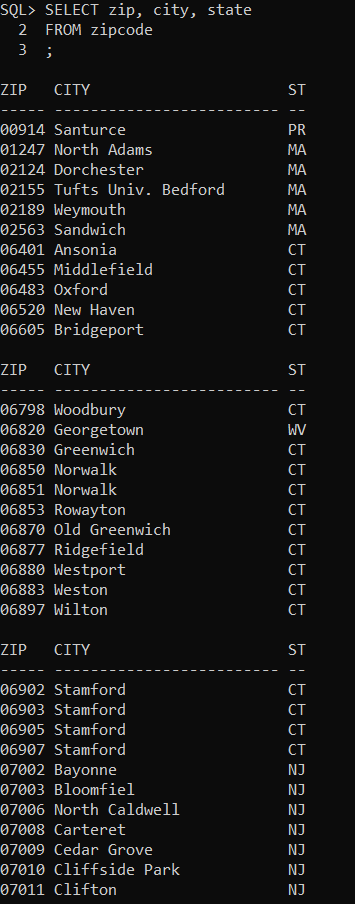
LOOP ...

### 12.1.1. Uso de parámetros en un cursor

En el siguiente ejemplo se incluye la línea DBMS\_OUTPUT que muestra el código postal, la ciudad y el estado. Esto es idéntico al proceso que ya ha usado en un CURSOR FOR LOOP, solo que ahora, cuando abre el cursor, pasa un parámetro

(Ejemplo de la pagina 255)



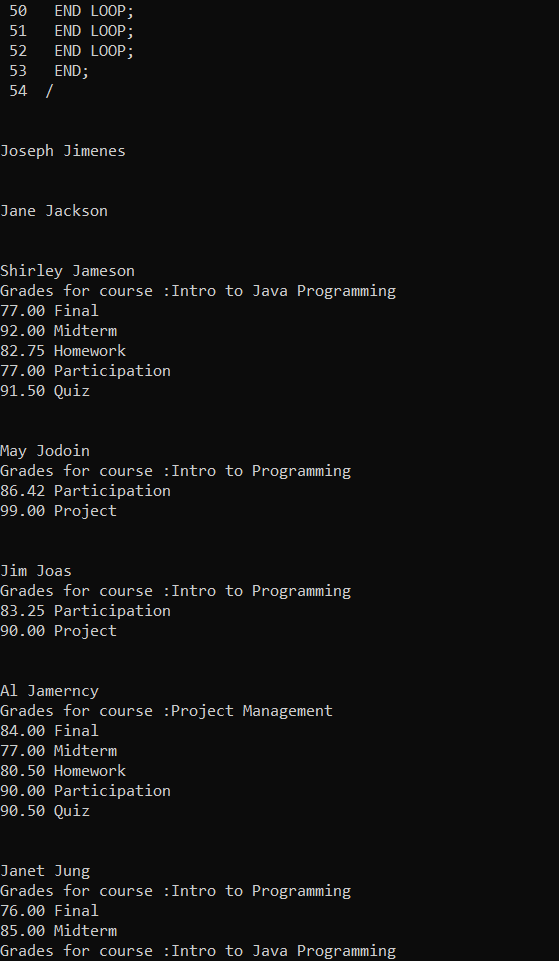
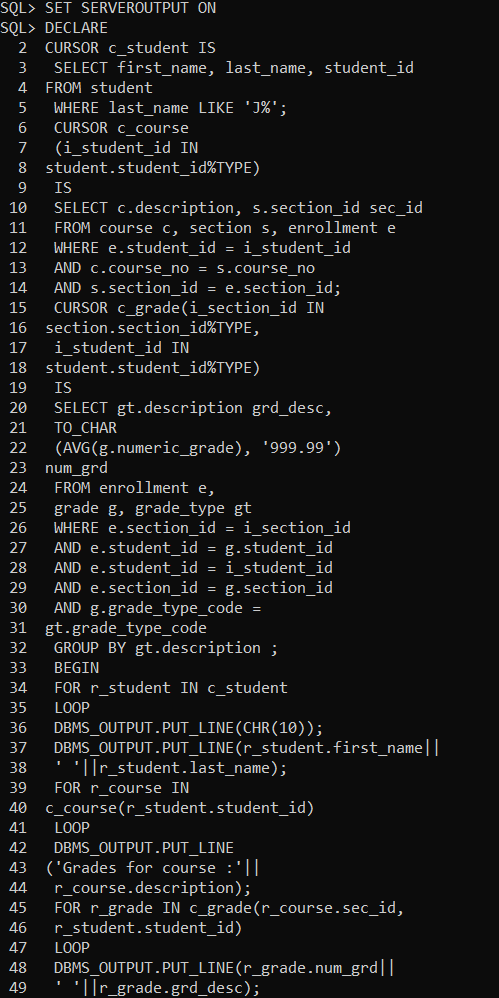


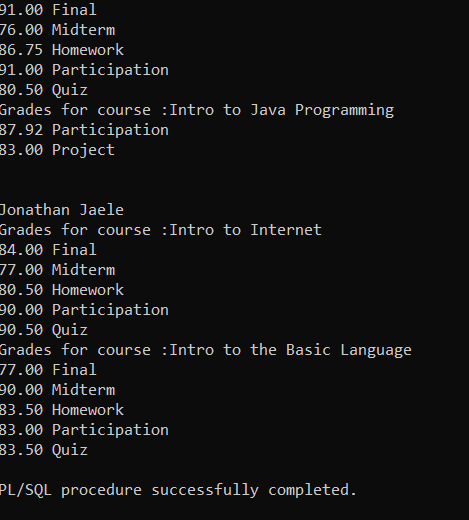
### 12.1.2. Uso de cursores anidados complejos

Los cursores de anidación permiten recorrer datos en varias etapas. Por ejemplo, un cursor podría recorrer códigos postales. Cuando llegue a un código postal, un segundo cursor anidado pasará por los estudiantes que viven en ese código postal.

Para entender mejor, veamos el siguiente ejemplo: (Ejemplo de la pagina 255 y 256)

1. El siguiente código PL / SQL es complejo. Incluye todos los temas tratados hasta ahora en este capítulo. Tiene un cursor anidado con tres niveles, lo que significa un cursor abuelo, un cursor principal y un cursor secundario. Antes de ejecutar este script, revise el código e identifique sus niveles de anidación.





¿Qué esta sucediendo?

1. El cursor abuelo, c\_student, se declara en las líneas 2 a 5. No requiere parámetros y es una colección de estudiantes con un apellido que comienza con J.
2. El cursor padre, c\_course, se declara en las líneas 6 a 13. Toma el parámetro de student\_ID para generar una lista de cursos que el estudiante está tomando.
3. El cursor secundario, c\_grade, se declara en las líneas 14 a 27. Incluye dos parámetros, section\_id y student\_id. De esta manera, puede generar un promedio de los diferentes tipos de calificación (cuestionarios, tareas, final, etc.) para ese alumno para ese curso.
4. El cursor loop abuelo comienza en la línea 29, y solo el nombre del alumno se muestra con DBMS\_OUTPUT.
5. El ciclo de cursor padre comienza en la línea 35. Toma el parámetro de student\_id del cursor abuelo. Solo se muestra la descripción del curso.
6. El ciclo del cursor del niño comienza en la línea 40. Toma el parámetro de section\_id del cursor principal y student\_id del cursor del abuelo. Las calificaciones se muestran.
7. El bucle del cursor abuelo termina en la línea 45, el bucle del cursor principal en la línea 44 y el bucle del cursor del niño en la línea 43.

## 12.2. Cursores FOR UPDATE y WHERE CURRENT

La cláusula cursor FOR UPDATE se usa solo con un cursor cuando desea actualizar tablas en la base de datos. En general, cuando ejecuta una instrucción SELECT, no está bloqueando ninguna fila.

El propósito de usar la cláusula FOR UPDATE es bloquear las filas de las tablas que desea actualizar para que otro usuario no pueda realizar una actualización hasta que realice la actualización y libere el bloqueo. La siguiente instrucción COMMIT o ROLLBACK libera el bloqueo.

La cláusula FOR UPDATE cambia la manera en que el cursor opera solo en unos pocos aspectos. Cuando abre un cursor, todas las filas que cumplen los criterios de restricción se identifican como parte del conjunto activo. El uso de la cláusula FOR UPDATE bloquea estas filas que se han identificado en el conjunto activo.

Si se utiliza la cláusula FOR UPDATE, las filas no se pueden recuperar desde el cursor hasta que se haya emitido un COMMIT. Es importante pensar dónde colocar el COMMIT.

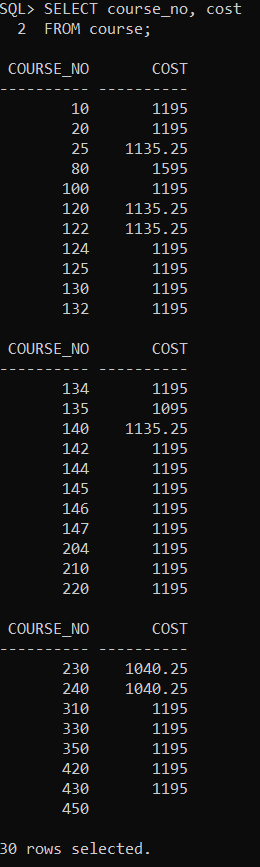
La sintaxis es simplemente agregar FOR UPDATE al final de la definición del cursor. Si se seleccionan varios elementos, pero desea bloquear solo uno de ellos, finalice la definición del cursor con la siguiente sintaxis:

FOR UPDATE OF <ítem\_name>

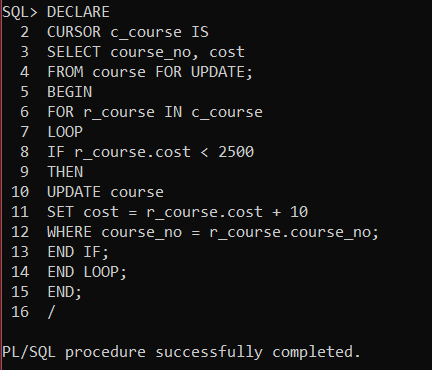
### 12.2.1. Cursores FOR UPDATE y WHERE CURRENT

El siguiente ejemplo muestra como actualizar el costo de todos los cursos que cuestan menos de $2,500. Los incrementa en 10. (Ejemplo de la página 259, en la siguientes paginas vienen mas ejemplos cortos y bonitos como este)

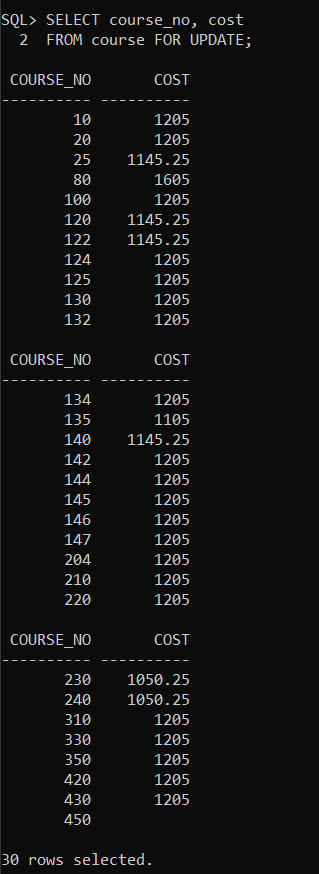
Inicialmente, estos son los valores:



Una vez ejecutado el siguiente código



El resultado es el siguiente



¿Qué esta sucediendo?

La pregunta inicial es: ¿dónde debe colocarse el COMMIT?, ¿Qué problemas están involucrados al decidir donde colocar un COMMIT en este ejemplo?

Colocar un COMMIT después de cada actualización puede ser costoso. Pero si hay muchas actualizaciones y el COMMIT viene después del ciclo de bloqueo, el segmento de reversión podría no ser lo suficientemente grande.

Normalmente, COMMIT iría después del bucle, excepto cuando el recuento de transacciones es alto. En ese caso, es posible que desee codificar algo que haga un COMPROMISO por cada 10.000 registros. Si esto fue parte de un procedimiento grande, es posible que desee colocar un SAVEPOINT después del ciclo. Entonces, si necesita deshacer esta actualización más tarde, esto sería fácil

**Conclusion**

En este capitilo comprendi las funciones avanzadas de los cursosres el recibir parametros y el usar sentencias como Update , comprendi el funcionamiento de el bloqueo de funciones y por ello la importancia del COMMIT para liberar las tablas.

Las ventajas de usar cursores avanzados

* Reutilizar cursoes cuando se reciben parametros
* Usar parámetros con cursores y cursores anidados complejos
* Utilizar cursores FOR UPDATE y WHERE CURRENT.

# 13. Triggers

En este capítulo, aprendimos sobre. qué son los tiggers (en español, desencadenantes) y cuales son los tipos de tiggers.

En el Capítulo 1, "Conceptos PL / SQL", encontró el concepto de bloques PL / SQL con nombre de procedimientos, funciones y paquetes que se pueden almacenar en la base de datos. En este capítulo, se aprendió sobre **otro tipo de bloque PL / SQL** llamado trigger (disparador) de base de datos.

Un "trigger" (disparador o desencadenador) es un bloque de código que se ejecuta automáticamente cuando ocurre algún evento (como inserción, actualización o borrado) sobre una determinada tabla (o vista); es decir, cuando se intenta modificar los datos de una tabla (o vista) asociada al disparador.

Se crean para conservar la integridad referencial y la coherencia entre los datos entre distintas tablas; para registrar los cambios que se efectúan sobre las tablas y la identidad de quien los realizó; para realizar cualquier acción cuando una tabla es modificada; etc.

Si se intenta modificar (agregar, actualizar o eliminar) datos de una tabla asociada a un disparador, el disparador se ejecuta (se dispara) en forma automática.

La diferencia con los procedimientos almacenados del sistema es que los triggers:

- no pueden ser invocados directamente; al intentar modificar los datos de una tabla asociada a un disparador, el disparador se ejecuta automáticamente.

- no reciben y retornan parámetros.

- son apropiados para mantener la integridad de los datos, no para obtener resultados de consultas.

Sintaxis general para crear un disparador:

create or replace trigger NOMBREDISPARADOR

MOMENTO-- before, after o instead of

EVENTO-- insert, update o delete

of CAMPOS-- solo para update

on NOMBRETABLA

NIVEL--puede ser a nivel de sentencia (statement) o de fila (for each row)

when CONDICION--opcional

begin

CUERPO DEL DISPARADOR--sentencias

end NOMBREDISPARADOR;

Los triggers se crean con la instrucción "create trigger" seguido del nombre del disparador. Si se agrega "or replace" al momento de crearlo y ya existe un trigger con el mismo nombre, tal disparador será borrado y vuelto a crear.

"MOMENTO" indica cuando se disparará el trigger en relación al evento, puede ser BEFORE (antes), AFTER (después) o INSTEAD OF (en lugar de). "before" significa que el disparador se activará antes que se ejecute la operación (insert, update o delete) sobre la tabla, que causó el disparo del mismo. "after" significa que el trigger se activará después que se ejecute la operación que causó el disparo. "instead of" sólo puede definirse sobre vistas, anula la sentencia disparadora, se ejecuta en lugar de tal sentencia (ni antes ni después).

"EVENTO" especifica la operación (acción, tipo de modificación) que causa que el trigger se dispare (se active), puede ser "insert", "update" o "delete"; DEBE colocarse al menos una acción, puede ser más de una, en tal caso se separan con "or". Si "update" lleva una lista de atributos, el trigger sólo se ejecuta si se actualiza algún atributo de la lista.

"on NOMBRETABLA" indica la tabla (o vista) asociada al disparador;

"NIVEL" puede ser a nivel de sentencia o de fila. "for each row" indica que el trigger es a nivel de fila, es decir, se activa una vez por cada registro afectado por la operación sobre la tabla, cuando una sola operación afecta a varios registros. Los triggers a nivel de sentencia, se activan una sola vez (antes o después de ejecutar la operación sobre la tabla). Si no se especifica, o se especifica "statement", es a nivel de sentencia.

"CUERPO DEL DISPARADOR" son las acciones que se ejecutan al dispararse el trigger, las condiciones que determinan cuando un intento de inserción, actualización o borrado provoca las acciones que el trigger realizará. El bloque se delimita con "begin... end".

Entonces, un disparador es un bloque de código asociado a una tabla que se dispara automáticamente antes o después de una sentencia "insert", "update" o "delete" sobre la tabla.

Se crean con la instrucción "create trigger" especificando el momento en que se disparará, qué evento lo desencadenará (inserción, actualización o borrado), sobre qué tabla (o vista) y las instrucciones que se ejecutarán.

Los disparadores pueden clasificarse según tres parámetros:

- el momento en que se dispara: si se ejecutan antes (before) o después (after) de la sentencia.

- el evento que los dispara: insert, update o delete, según se ejecute una de estas sentencias sobre la tabla.

- nivel: dependiendo si se ejecuta para cada fila afectada en la sentencia (por cada fila) o bien una única vez por sentencia independientemente de la filas afectadas (nivel de sentencia).

Consideraciones generales:

- Las siguientes instrucciones no están permitidas en un desencadenador: create database, alter database, drop database, load database, restore database, load log, reconfigure, restore log, disk init, disk resize.

- Se pueden crear varios triggers para cada evento, es decir, para cada tipo de modificación (inserción, actualización o borrado) para una misma tabla. Por ejemplo, se puede crear un "insert trigger" para una tabla que ya tiene otro "insert trigger".

A continuación veremos la creación de disparadores para los distintos sucesos (inserción, borrado, actualización) y todas las cláusulas adicionales.

## 13.1. Que son los triggers

Después de completar este laboratorio, podrás. Utilizar triggers BEFORE y AFTER

Un trigger de base de datos es un bloque PL / SQL almacenado en una base de datos y ejecutado implícitamente cuando ocurre un evento desencadenante

Un evento desencadenante puede ser uno de los siguientes:

* Una instrucción DML (como INSERT, UPDATE o DELETE) ejecutada en una tabla de base de datos. Tal trigger puede disparar antes o después de un evento desencadenante (triggering). Por ejemplo, si ha definido un trigger para disparar antes de una instrucción INSERT en la tabla STUDENT, este trigger se dispara cada vez antes de insertar una fila en la tabla STUDENT.
* Una instrucción DDL (como CREATE o ALTER) ejecutada por un usuario particular contra un esquema o por cualquier usuario. Tales triggers a menudo se utilizan con fines de auditoría y son especialmente útiles para los DBA de Oracle. Pueden registrar varios cambios de esquema, cuándo se realizaron y por qué usuario.
* Un evento del sistema como inicio o cierre de la base de datos.
* Un evento de usuario como inicio de sesión y cierre de sesión. Por ejemplo, puede definir un trigger que se activa después del inicio de sesión de la base de datos que registra el nombre de usuario y la hora de inicio de sesión

La sintaxis general para crear un trigger es la siguiente (las palabras reservadas dentro de los corchetes son opcionales:

CREATE [OR REPLACE] TRIGGER Ttrigger\_name

{BEFORE|AFTER} Triggering\_event ON table\_name

[FOR EACH ROW]

[FOLLOWS another\_trigger]

[ENABLE/DISABLE]

[WHEN condition]

DECLARE

declaration statements

BEGIN

executable statements

EXCEPTION

exception-handling statements

END;

La palabra reservada CREATE especifica que está creando un nuevo trigger. La palabra reservada REPLACE especifica que está modificando un trigger existente. REPLACE es opcional. Sin embargo, tenga en cuenta que tanto CREATE como REPLACE están presentes la mayor parte del tiempo. Considera la siguiente situación. Crea un trigger de la siguiente manera:

CREATE TRIGGER Trigger\_name ...

En unos días decide modificar este trigger. Si no incluye la palabra reservada REPLACE en la cláusula CREATE del trigger, se generará un mensaje de error cuando compile el trigger. El mensaje de error indica que el nombre de su trigger ya está siendo utilizado por otro objeto. Cuando REPLACE se incluye en la cláusula CREATE del disparador, hay menos posibilidades de que se produzca un error porque, si se trata de un nuevo trigger, se crea, y si se trata de un trigger anterior, se reemplaza.

El *trigger\_name* es el nombre del trigger. ANTES o DESPUÉS especifica cuándo se dispare el trigger (antes o después del evento desencadenante). El evento triggering\_event es un enunciado DML emitido contra la tabla. table\_name que es el nombre de la tabla asociada al trigger.

La cláusula FOR EACH ROW especifica que un trigger es un trigger de fila y dispara una vez por cada fila que se inserta, actualiza o elimina. Una cláusula WHEN especifica una condición que debe evaluarse como TRUE para que el trigger dispare. Por ejemplo, esta condición puede especificar una cierta restricción en la columna de una tabla. Esta parte del trigger a menudo se llama encabezado trigger (desencadenante). A continuación, se define el cuerpo del trigger.

Tenga en cuenta las tres cláusulas, FOLLOWS, ENABLE y DISABLE. Estos se agregaron a la cláusula CREATE O REPLACE TRIGGER en Oracle 11g. Antes de Oracle 11g, era necesario emitir el comando ALTER TRIGGER para habilitar o deshabilitar un trigger después de haber sido creado.

Las cláusulas ENABLE y DISABLE especifican si el trigger se crea en estado habilitado o deshabilitado. Cuando el trigger está habilitado, se dispara cuando ocurre un evento desencadenante. De manera similar, cuando un trigger está desactivado, no se dispara cuando ocurre un evento de disparo. Tenga en cuenta que cuando el trigger se crea por primera vez sin una cláusula ENABLE o DISABLE, está habilitado de forma predeterminada. Para desactivar el trigger, debe emitir el comando ALTER TRIGGER de la siguiente manera:

ALTER TRIGGER trigger\_name DISABLE;

De forma similar, para habilitar un trigger que se deshabilitó previamente, se emite el comando ALTER TRIGGER de la siguiente manera:

ALTER TRIGGER nombre\_encendido ENABLE;

La opción FOLLOWS le permite especificar el orden en el que los triggers deben disparar. Esto aplica a los triggers que están definidos en la misma tabla y que disparan en el mismo punto de tiempo. Por ejemplo, si definió dos triggers en la tabla STUDENT que disparan antes de que ocurra la inserción, Oracle no garantiza el orden en que se dispararán estos triggers a menos que lo especifique con la cláusula FOLLOWS. Tenga en cuenta que el trigger al que se hace referencia en la cláusula FOLLOWS ya debe existir y haberse compilado correctamente.

Es importante que se dé cuenta de que si suelta una tabla, los trigger de la base de datos de la tabla también se descartan.

Debe tener cuidado al usar la palabra reservada REPLACE por varias razones. Primero, si usa REPLACE y el nombre de una función, procedimiento o paquete almacenado existente, terminará con diferentes objetos de base de datos que tienen el mismo nombre. Esto ocurre porque los triggers tienen espacio de denominación separado en la base de datos.

Aunque un trigger y un procedimiento, función o paquete que comparten el mismo nombre no provocan errores, puede ser confuso. Como resultado, no se considera una buena práctica de programación. En segundo lugar, cuando utiliza la palabra reservada REPLACE y decide asociar una tabla diferente con su trigger, se genera un mensaje de error. Por ejemplo, suponga que creó un desencadenador STUDENT\_BI en la tabla STUDENT. Luego, decide modificar este trigger y asociarlo con la tabla de ENROLLMENT. Como resultado, se genera el siguiente mensaje de error:

ERROR en la línea 1:

ORA-04095: el disparador 'STUDENT\_BI' ya existe en otra tabla, no puede reemplazarlo.

Los triggers se usan para diferentes propósitos:.

* Aplicación de reglas comerciales complejas que no se pueden definir mediante el uso de restricciones de integridad.
* Mantenimiento de reglas de seguridad complejas.
* Generación automática de valores para columnas derivadas.
* Recopilación de información estadística sobre los accesos a la tabla.
* Previniendo transacciones inválidas.
* Proporcionar auditoría de valor

El cuerpo de un trigger es un bloque PL / SQL. Sin embargo, debe conocer varias restricciones antes de crear un trigger.

* Un trigger no puede emitir una instrucción de control transaccional como COMMIT, SAVEPOINT o ROLLBACK. Cuando el trigger se dispara, todas las operaciones realizadas se convierten en parte de una transacción. Cuando se compromete o retrotrae esta transacción, las operaciones realizadas por el trigger también se comprometen o retrotraen. Una excepción a esta regla es un trigger que contiene una transacción autónoma.
* Cualquier función o procedimiento llamado por un trigger no puede emitir una instrucción de control transaccional a menos que contenga una transacción autónoma.
* No está permitido declarar variables LONG o LONG RAW en el cuerpo de un disparador

**BEFORE TRIGGERS**

Considere el siguiente ejemplo de un trigger en la tabla de STUDENT. Este trigger se dispara antes de la instrucción INSERT en la tabla STUDENT y rellena las columnas STUDENT\_ID, CREATED\_DATE, MODIFIED\_DATE, CREATED\_BY y MODIFIED\_BY. La columna STUDENT\_ID se completa con el número generado por la secuencia STUDENT\_ID\_SEQ, y las columnas CREATED\_DATE, MODIFIED\_DATE, CREATED\_USER y MODIFIED\_USER se rellenan con la fecha actual y la información de nombre de usuario actual.

CREATE OR REPLACE TRIGGER student\_bi

BEFORE INSERT ON student

FOR EACH ROW

BEGIN

:NEW.student\_id := STUDENT\_ID\_SEQ.NEXTVAL;

:NEW.created\_by := USER;

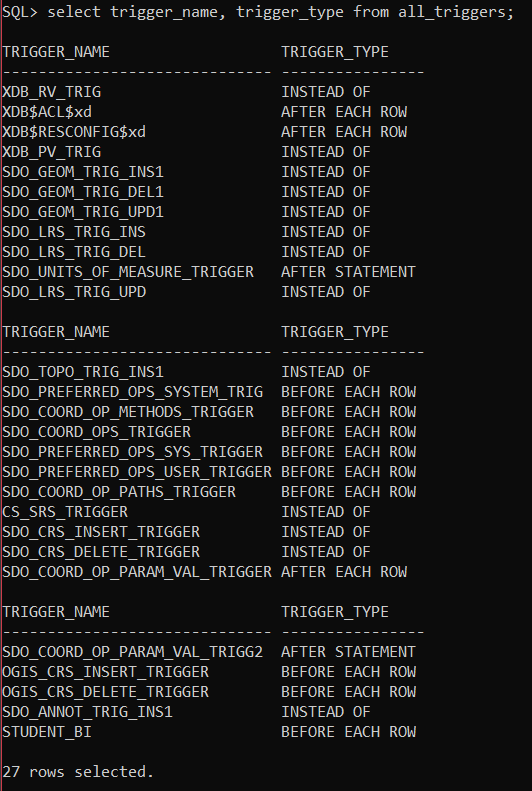
:NEW.created\_date := SYSDATE;

:NEW.modified\_by := USER;

:NEW.modified\_date := SYSDATE;

END;

Al ejecutar este código, podemos verificar su existencia y el tipo de trigger.



Este trigger se activa **para cada fila antes** de la instrucción INSERT en la tabla STUDENT. Observe que el nombre del trigger es STUDENT\_BI, donde STUDENT es el nombre de la tabla en la que se define el trigger, y BI significa BEFORE INSERT (ANTES DE INSERTAR). No hay un requisito específico para nombrar triggers; sin embargo, este enfoque para nombrar un trigger es descriptivo. El nombre del trigger contiene el nombre de la tabla afectada por el evento desencadenante, la hora del evento desencadenante (antes o después) y el evento desencadenante mismo.

En el cuerpo del trigger hay un pseudo registro, NEW, que le permite acceder a una fila que se está procesando actualmente. En otras palabras, se está insertando una fila en la tabla STUDENT. El:NEW pseudo registro es del tipo TRIGGERING\_TABLE% TYPE, por lo tanto, en este caso, es del tipo STUDENT% TYPE. Para acceder a miembros individuales del pseudo registro: NEW, se utiliza notación de punto. En otras palabras,: NEW.CREATED\_BY se refiere al miembro CREATED\_BY del NEW pseudo registro, y el nombre del registro está separado del nombre de su miembro por un punto. La capacidad de acceder a una secuencia mediante expresión PL / SQL es una nueva característica en Oracle 11g. Antes de Oracle 11g, solo se podía acceder a las secuencias mediante consultas.

Debe usar BEFORE triggers en las siguientes situaciones:

* Cuando un trigger proporciona valores para las columnas derivadas antes de que se complete una instrucción INSERT o UPDATE. Por ejemplo, la columna FINAL\_GRADE en la tabla de ENROLLMENT contiene el valor de la calificación final del alumno para un curso específico. Este valor se calcula en función del rendimiento del alumno durante el curso.
* Cuando un trigger determina si se debe permitir que se complete una instrucción INSERT, UPDATE o DELETE. Por ejemplo, cuando inserta un registro en la tabla INSTRUCTOR, un trigger puede verificar si el valor proporcionado para la columna ZIP es válido o, en otras palabras, si un registro en la tabla ZIPCODE corresponde al valor de zip que proporcionó .

**AFTER TRIGGERS**

Se deberíausar AFTER triggers en las siguientes situaciones:

* Cuando se dispara un trigger después de que se ejecuta una instrucción DML.
* Cuando un trigger realiza acciones no especificadas en un BEFORE TRIGGER.

**AUTONOMOUS TRANSACTION**

Una transacción autónoma es una transacción independiente iniciada por otra transacción que generalmente se denomina transacción principal. En otras palabras, la transacción autónoma puede emitir varias sentencias DML y comprometerlas o retrotraerlas, sin comprometer o revertir las declaraciones DML emitidas por la transacción principal.

### 13.1.1. Entendiendo que es un trigger.

En este ejercicio, debe determinar el evento de activación del disparador, su tipo, etc., de acuerdo con la cláusula CREATE del activador.

Consideramos el siguiente código

CREATE TRIGGER student\_au

AFTER UPDATE ON STUDENT

FOR EACH ROW

WHEN (NVL(NEW.ZIP, ' ') <> OLD.ZIP)

Trigger Body...

En la instrucción WHEN de la cláusula CREATE, el pseudo registro: OLD le permite acceder a una fila que se está procesando actualmente. Es importante tener en cuenta que ni NEW ni: OLD está precedido por dos puntos (:) cuando se utiliza en la condición de la instrucción WHEN. Ya estás familiarizado con el pseudo registro :NEW. El: pseudo registro antiguo le permite acceder a la información actual del registro que se está actualizando. En otras palabras, es información actualmente presente en la tabla STUDENT para un registro específico. El :NEW pseudo registro le permite acceder a la nueva información para el registro actual. En otras palabras,:NEW indica los valores actualizados. Por ejemplo, considere la siguiente instrucción UPDATE:

UPDATE student

SET zip = '01247'

WHERE zip = '02189';

El valor 01247 de la columna ZIP es un valor nuevo, y el desencadenador lo hace referencia como: NEW.ZIP. El valor 02189 en la columna ZIP es el valor anterior y se hace referencia como: OLD.ZIP

:OLD no está definido para las instrucciones INSERT, y :NEW no está definido para las instrucciones DELETE. Sin embargo, el compilador PL/SQL no genera errores de sintaxis cuando: OLD o: NEW se utiliza en activadores donde el evento desencadenante es una operación INSERT o DELETE, respectivamente. En este caso, los valores de campo se establecen en NULL para: OLD y :NEW pseudoregistros.

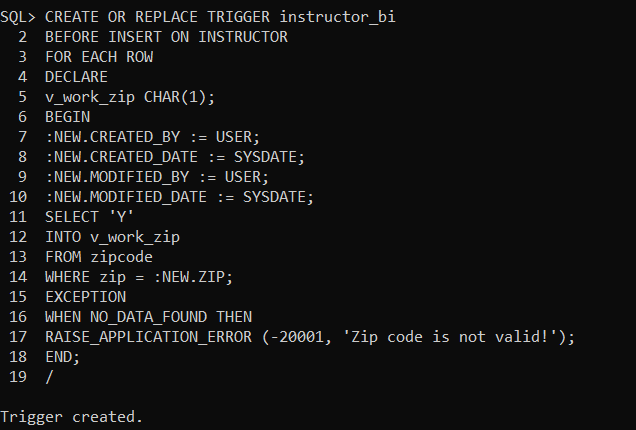
En el ejemplo anterior, suponga que ya existe un trigger llamado STUDENT\_AU en la base de datos. Si usa la cláusula CREATE para modificar el trigger existente, ¿qué mensaje de error se genera?

Veremos un mensaje de error que indica que el nombre STUDENT\_AU ya está siendo utilizado por otro objeto. La cláusula CREATE puede crear nuevos objetos en la base de datos, pero no puede manejar las modificaciones. Para modificar el trigger existente, debe agregar la instrucción REPLACE a la cláusula CREATE. En este caso, la versión anterior del trigger se descarta sin previo aviso y se crea la nueva versión del activador.

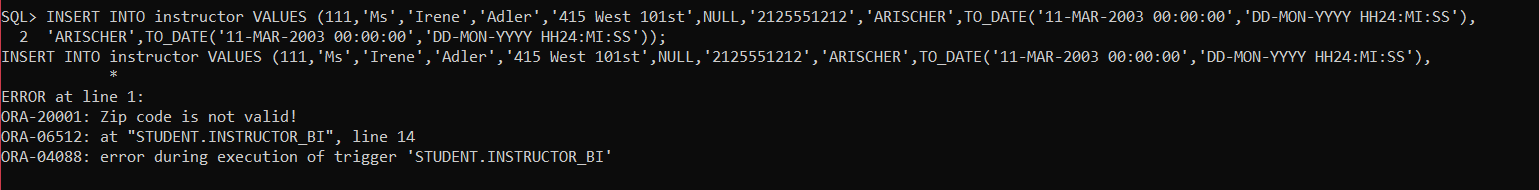
### 13.1.2. Uso de BEFORE y AFTER Triggers

En este ejercicio, crea un trigger en la tabla INSTRUCTOR que se activa antes de que se emita una instrucción INSERT contra la tabla. El trigger determina los valores para las columnas CREATED\_BY, MODIFIED\_BY, CREATED\_DATE y MODIFIED\_DATE. Además, determina si el valor de zip provisto por una instrucción INSERT es válido.

Crea el siguiente trigger:



Si a una declaración INSERT emitida contra la tabla INSTRUCTOR le falta un valor para la columna ZIP, ¿el disparador genera una excepción?



Sí, el desencadenante genera una excepción. Cuando una instrucción INSERT no proporciona un valor para la columna ZIP, el valor de: NEW.ZIP es NULL. Este valor se usa en la cláusula WHERE de la instrucción SELECT INTO. Como resultado, la instrucción SELECT INTO no puede devolver datos. Por lo tanto, el desencadenante genera una excepción NO\_DATA\_FOUND.

## 13.2. Tipos de triggers

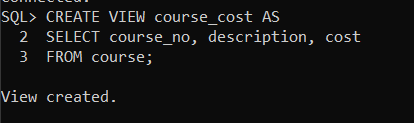
En este capitulo, vimos como

* Usar triggers de filas y declaraciones.
* Use los triggers INSTEAD OF

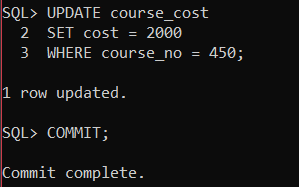
**INSTEAD OF TRIGGERS**

Hasta ahora, ha visto activadores que están definidos en las tablas de la base de datos. PL / SQL proporciona otro tipo de disparador que se define en las vistas de la base de datos. Una vista es una representación personalizada de datos y se puede llamar una consulta almacenada. Considere el siguiente ejemplo de la vista creada contra la tabla CURSO:

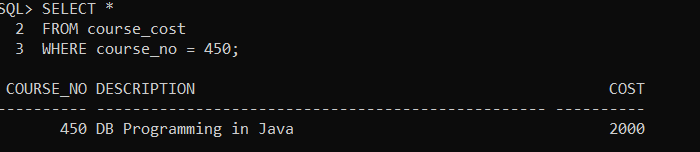
Se crea una vista para COURSE.

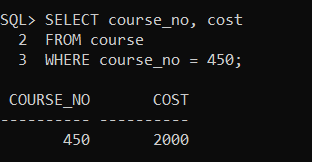


De forma similar a las tablas, las vistas se pueden manipular a través de instrucciones INSERT, UPDATE o DELETE, con algunas restricciones. Sin embargo, es importante tener en cuenta que cuando se emiten cualquiera de estas sentencias contra una vista, los datos correspondientes se modifican en las tablas subyacentes. Por ejemplo, considere una instrucción UPDATE contra la vista COURSE\_COST:



Después de que se ejecuta la instrucción UPDATE, ambas instrucciones SELECT contra la vista COURSE\_COST y la tabla CURSO devuelven el mismo valor del costo para el curso número 450:



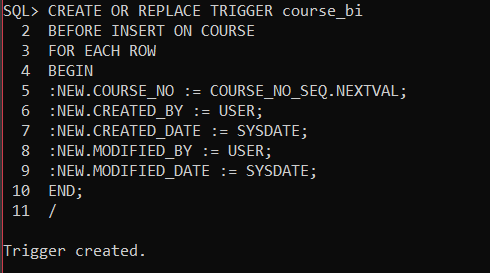


Como se mencionó anteriormente, algunas vistas están restringidas en cuanto a si se pueden modificar mediante instrucciones INSERT, UPDATE o DELETE. Específicamente, estas restricciones se aplican a la instrucción SELECT subyacente, que también se denomina **consulta de vista.** Por lo tanto, si una consulta de vista realiza cualquiera de las operaciones o contiene cualquiera de las siguientes construcciones, una vista no puede modificarse mediante una instrucción UPDATE, INSERT o DELETE:

* Establezca operaciones como UNION, UNION ALL, INTERSECT y MINUS.
* Funciones de grupo como AVG, COUNT, MAX, MIN y SUM.
* GROUP BY o HAVING cláusulas.
* Clausulas CONNECT BY o START WITH.
* El operador DISTINCT.
* La pseudocolumna ROWNUM

### 13.2.1. Uso de triggers de filas y de declaraciones

En este ejercicio, se crea un trigger que se dispara antes de que se emita una instrucción INSERT en la tabla curso.



¿Qué tipo de trigger se crea en la fila o declaración de la tabla CURSE?

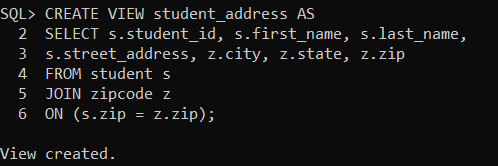
El trigger creado en la tabla CURSO es un trigger de fila porque la cláusula CREATE TRIGGER contiene la instrucción FOR EACH ROW. Esto significa que este trigger se dispara cada vez que se agrega un registro a la tabla COURSE.

¿Por qué se elige este tipo particular para el trigger para el ejemplo anterior?

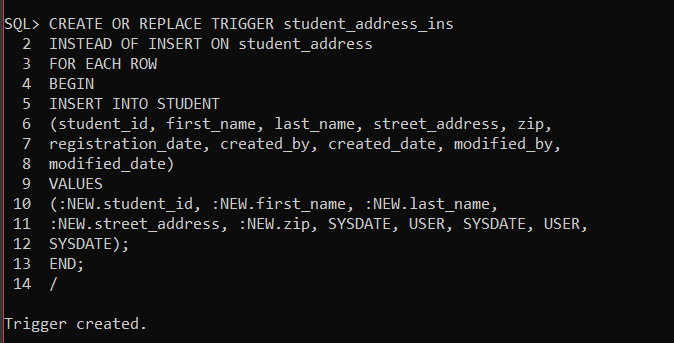
Este trigger es un trogger de fila porque sus operaciones dependen de los datos en el archivo individual. Por ejemplo, para cada registro insertado en la tabla COURSE, el trigger calcula el valor para la columna COURSE\_NO. Todos los valores en esta columna deben ser únicos, porque es definido como una llave primaria. Un trigger de fila garantiza que cada registro agregado a la tabla COURSE tiene un número único asignado a la columna COURSE\_NO.

### 13.2.2. Uso de triggers INSTEAD OF

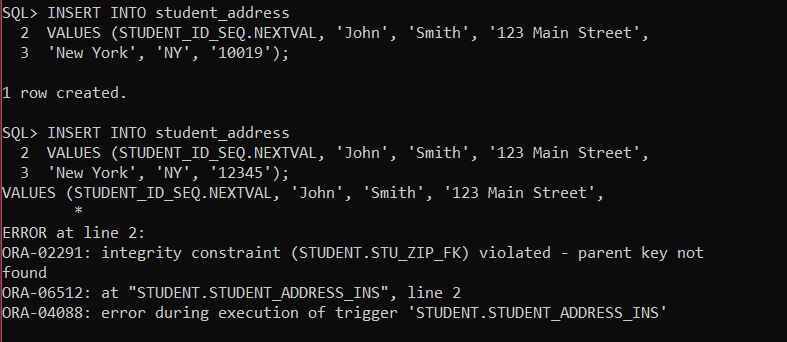
En este ejercicio, crea una vista STUDENT\_ADDRESS y un disparador INSTEAD OF que se dispara en lugar de una instrucción INSERT emitida en contra de la vista. Crea la siguiente vista:



Creamos un trigger:



Emita las siguientes instrucciones INSERT:



Explicamos por qué la segunda instrucción INSERT causa un error.

La segunda instrucción INSERT causa un error porque viola la restricción de clave externa en la tabla STUDENT. El valor del código postal proporcionado en el momento de una inserción no tiene un registro correspondiente en la tabla de ZIPCODE. La columna ZIP de la tabla STUDENT tiene una restricción de clave externa STU\_ZIP\_FK definida en ella. Esto significa que cada vez que se inserta un registro en la tabla STUDENT, el sistema verifica el valor entrante del código postal en la tabla ZIPCODE. Si hay un registro correspondiente, la instrucción INSERT contra la tabla ESTUDIANTE no causa errores. Por ejemplo, la primera instrucción INSERT es exitosa porque la tabla ZIPCODE contiene un registro correspondiente al valor del código postal 10019. La segunda instrucción Insert causa un error porque ningún registro en la tabla ZIPCODE corresponde al valor del código postal 12345

**Conclusion**

En este capitulo pude comprender la funcionalidad de los triggers , el uso del before y after asi como los diversos tipos de triggers que hay

La ventajas de usar triggers:

* Seguridad de los datos mejorada:
* Ofrecen chequeos de seguridad basada en valores.
* Integridad de los datos mejorada:
* Fuerzan restricciones dinámicas de integridad de datos yde integridad referencial.
* Aseguran que las operaciones relacionadas se realizan juntas de forma implícita

# 19. Procedimientos

Todo el PL/SQL que ha escrito hasta este punto ha sido bloques anónimos que se ejecutaron como scripts y compilados por el servidor de la base de datos en tiempo de ejecución. Ahora comenzará a usar código modular. El código modular es una forma de crear un programa a partir de distintas partes (módulos), cada una de las cuales realiza una función o tarea específica hacia el objetivo final del programa. Tan pronto como el código modular se almacena en el servidor de la base de datos, se convierte en un objeto de base de datos, o subprograma, que está disponible para otras unidades de programa para la ejecución repetida.

Para guardar el código en la base de datos, el código fuente debe enviarse al servidor para que pueda compilarse en código-p y almacenarse en la base de datos.

**BENEFICIOS DEL CODIGO MODULAR**

Un módulo PL / SQL es cualquier unidad lógica de trabajo completa. Los cinco tipos de módulos PL/SQL son **bloques anónimos** que se ejecutan con un script de texto (este es el tipo que ha utilizado hasta ahora), **procedimientos, funciones, paquetes** y **triggers**. El uso de un código modular ofrece dos beneficios principales: es más reutilizable y más manejable.

**ESTRUCTURA DEL BLOQUE**

La estructura del bloque es común para todos los tipos de módulos.

1. El bloque comienza con un encabezado (solo para bloques con nombre), que consiste en el nombre del módulo y una lista de parámetros (si se usa).
2. La sección de declaración consta de variables, cursores y subbloques que se necesitan en la siguiente sección.
3. La parte principal del módulo es la sección ejecutable, que es donde se realizan todos los cálculos y el procesamiento. Esta sección contiene código ejecutable como IF-THEN-ELSE, bucles, llamadas a otros módulos PL / SQL, etc.
4. La última sección del módulo es una sección opcional de manejo de excepciones, que es donde se ubica el código para manejar las excepciones.

**BLOQUE ANONIMO**

Hasta ahora, solo has escrito bloques anónimos. Los bloques anónimos son muy parecidos a los módulos, excepto que los bloques anónimos no tienen encabezados. Sin embargo, hay diferencias importantes. Como su nombre lo indica, los bloques anónimos no tienen nombre y, por lo tanto, no pueden ser llamados por otro bloque. **No se almacenan en la base de datos y deben compilarse y luego ejecutarse cada vez que se carga el script**.

El bloque PL / SQL en un subprograma es un bloque nombrado que puede aceptar parámetros y que puede invocarse desde una aplicación que puede comunicarse con el servidor de base de datos Oracle. Un subprograma se puede compilar y almacenar en la base de datos. Esto le permite al programador reutilizar el programa. También permite un mantenimiento de código más sencillo. Los subprogramas son procedimientos o funciones.

## 19.1. Creando procedimientos

Tras la ejecución de los ejemplos y ejercicios, ahora podemos

* Crear procedimientos
* Consulta el diccionario de datos para obtener información sobre los procedimientos

Un procedimiento es un módulo que realiza una o más acciones; no necesita devolver ningún valor. La sintaxis para crear un procedimiento es la siguiente:

CREATE OR REPLACE PROCEDURE name

[(parameter[, parameter, ...])]

AS

[local declarations]

BEGIN

executable statements

[EXCEPTION

exception handlers]

END [name];

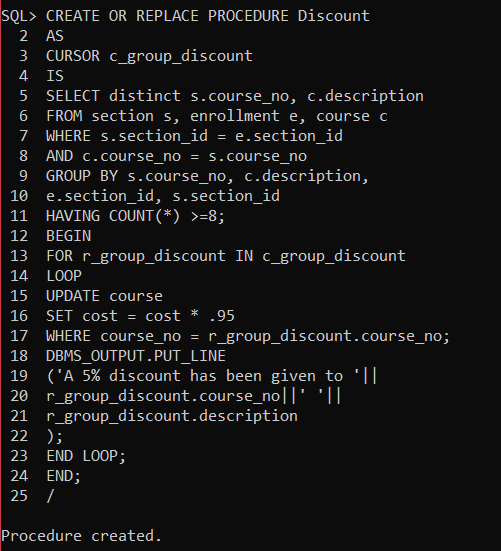
Un procedimiento puede tener de cero a muchos parámetros.

Cada procedimiento tiene tres partes: la parte del encabezado, que viene antes de AS (a veces se ve IS, son intercambiables);

* la palabra clave, que contiene el nombre del procedimiento y la lista de parámetros; y el cuerpo, que es todo después de la palabra clave AS.
* La palabra REPLACE es opcional. Cuando no se utiliza REPLACE en el encabezado del procedimiento, para cambiar el código en el procedimiento, debe soltar y luego volver a crear el procedimiento. Debido a que es muy común cambiar el código de un procedimiento, especialmente cuando está en desarrollo, se recomienda encarecidamente que utilice la opción OR REPLACE.

### 19.1.1. Creación de procedimientos

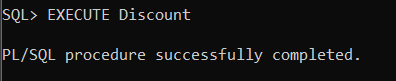
Por ejemplo (Ejemplo de la pagina 442, no hay otro en esta sección)



¿Qué esta sucediendo?

El procedimiento está creado. El procedimiento llamado Discount se compila en código p y se almacena en la base de datos para su posterior ejecución.

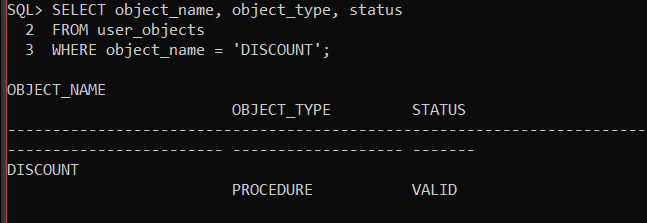
Ahora, se ejecuta el procedimiento Discount.



### 19.1.2. Consultar el diccionario de datos para obtener información sobre los procedimientos

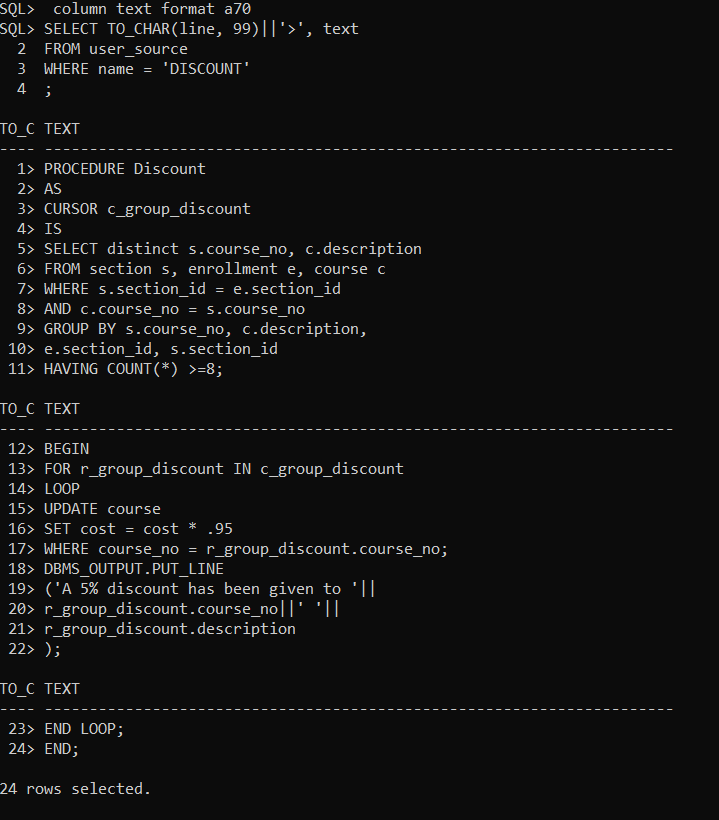
Dos vistas principales en el diccionario de datos proporcionan información sobre el código almacenado. USER\_OBJECTS muestra información sobre los objetos y USER\_SOURCE le muestra el texto del código fuente. El diccionario de datos también tiene las versiones ALL\_ y DBA\_ de estas vistas.

Como ejemplo, escribimos una instrucción SELECT para obtener información pertinente de la vista USER\_OBJECTS sobre el procedimiento de descuento que acabamos de escribir. (Página 443)



El estado indica que el procedimiento fue compilado con éxito. Un procedimiento inválido no puede ser ejecutado.

Ahora, escribimos una instrucción SELECT para mostrar el código fuente de la vista USER\_SOURCE para el procedimiento DISCOUNT.



## 19.2. Paso de parámetros dentro y fuera de los procedimientos

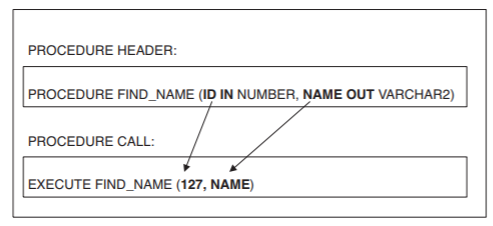
Con la ejecución de estos ejercicios, ahora podemos

* Use los parámetros IN y OUT con los procedimientos

Los parámetros son los medios para pasar valores hacia y desde el entorno de llamada al servidor. Estas son los valores que se procesan o devuelven al ejecutar el procedimiento. Los tres tipos de parámetros son IN, OUT y IN OUT

**MODOS**

Los modos especifican si el parámetro pasado se lee o un receptáculo para lo que sale (¿). La figura 19.1 ilustra la relación entre los parámetros cuando se encuentran en el encabezado del procedimiento y cuando se ejecuta el procedimiento.



**PARAMETROS FORMALES Y ACTUALES**

Los parámetros formales son los nombres especificados entre paréntesis como parte del encabezado de un módulo.

Los parámetros actuales son los valores o expresiones especificados entre paréntesis como una lista de parámetros cuando el módulo se llama. El parámetro formal y el parámetro real relacionado deben ser del mismo o tipos de datos compatibles.

La siguiente tabla explica los tres tipos de paámetros:

|  |  |  |
| --- | --- | --- |
| Modo | Descripción | Uso |
| IN | Pasa un valor en el programa  Constantes, literales, expresiones  No se puede cambiar dentro del modo predeterminado del programa | Solo lectura |
| OUT | Pasa un valor desde el programa  No se pueden asignar valores predeterminados  Debe ser una variable  Se asigna un valor solo si el programa tiene éxito | Solo escritura |
| IN OUT | Pasa valores en y también envía valores de vuelta | Es variable |

**RESTRICCIONES DE PASO (DATATYPE) CON VALORES PARÁMETROS**

Los parámetros formales no requieren restricciones en el tipo de datos. Por ejemplo, en lugar de especificar una restricción como VARCHAR2 (60), simplemente dices VARCHAR2 contra el nombre del parámetro en la lista de parámetros formales. La restricción se pasa con el valor cuando se realiza una llamada.

**COINCIDENCIA DE PARÁMETROS ACTUALES Y FORMALES**

Puede usar dos métodos para hacer coincidir los parámetros actuales y formales: la notación posicional y el nombre notación. La notación posicional es simplemente asociación por posición: el orden de los parámetros utilizados al ejecutar el procedimiento, coincide con el orden en el encabezado del procedimiento.

*Named notation is explicit association using the symbol =>:*

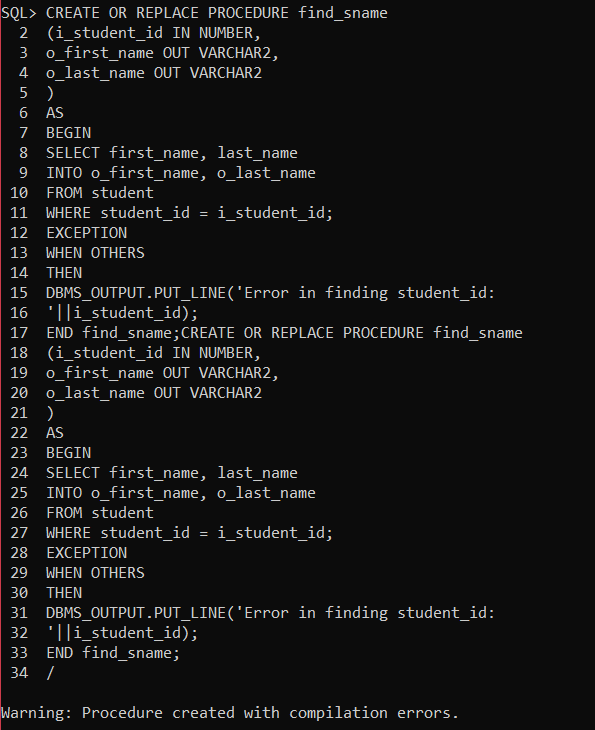
La notación con nombre es asociación explícita usando el símbolo =>:

formal\_parameter\_name => argument\_value

En notación con nombre, el orden no importa. Si mezcla notación, liste notación posicional antes notación nombrada.

### 19.2.1. Uso de los parámetros IN y OUT con los procedimientos

Se proporciona el siguiente ejemplo: (página 446)



¿Qué esta sucediendo?

¿Qué sucede en el procedimiento find\_sname.? ¿Qué parámetros se pasan dentro y fuera

del procedimiento? ¿Cómo llamarías al procedimiento?

Llamamos al script find\_sname con el siguiente bloque anónimo: (Página 446)

-- ch19\_03a.sql

DECLARE

v\_local\_first\_name student.first\_name%TYPE;

v\_local\_last\_name student.last\_name%TYPE;

BEGIN

find\_sname

(145, v\_local\_first\_name, v\_local\_last\_name);

DBMS\_OUTPUT.PUT\_LINE

('Student 145 is: '||v\_local\_first\_name||

' '|| v\_local\_last\_name||'.'

);

END;

**Conclusion**

Como conclusión: En este capítulo, he aprendido a como

* Crear procedimientos.
* Pasar parámetros dentro y fuera de los procedimientos

# 20. Funciones

## 20.1. Creación y uso de funciones

* Creación de funciones almacenadas.
* Haz uso de funciones.
* Invoca funciones en sentencias de SQL.
* Escribir funciones complejas

**FUNCIONES BÁSICAS**

Las funciones son otro tipo de código almacenado y son muy similares a los procedimientos. La diferencia significativa es **que una función es un bloque PL / SQL que devuelve un único valor.** Las funciones pueden aceptar uno, muchos o ningún parámetro, pero una función debe tener una cláusula de retorno en la sección ejecutable de la función. El tipo de datos del valor de retorno debe declararse en el encabezado de la función. Una función no es un ejecutable independiente de la misma manera que un procedimiento: debe usarse en algún contexto. Puedes pensarlo como un fragmento de oración. Una función tiene salida que debe asignarse a una variable, o puede usarse en una instrucción SELECT.

**SINTAXIS DE LA FUNCIÓN**

La sintaxis para crear una función es el siguiente:

CREATE [OR REPLACE] FUNCTION function\_name

(parameter list)

RETURN datatype

IS

BEGIN

<body>

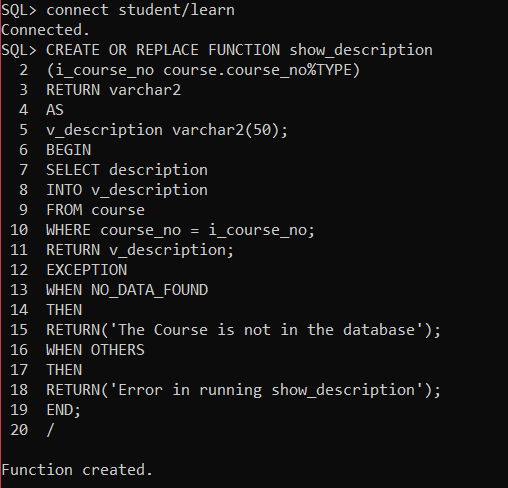
RETURN (return\_value);

END;

La función no tiene necesariamente ningún parámetro, pero debe tener un valor RETURN cuyo tipo de datos esté declarado en el encabezado, y debe devolver valores para todas las secuencias de ejecución posibles variables. La declaración RETURN no tiene que aparecer como la última línea de la sección de ejecución principal, y puede haber más de una instrucción RETURN (debe haber una declaración RETURN para cada excepción). Una función puede tener parámetros IN, OUT o IN OUT, pero rara vez se ve algo, excepto los parámetros IN, ya que es una mala práctica de programación hacer lo contrario.

### 20.1.1. Crear funciones almacenadas

Por ejemplo:



¿Qué esta sucediendo?

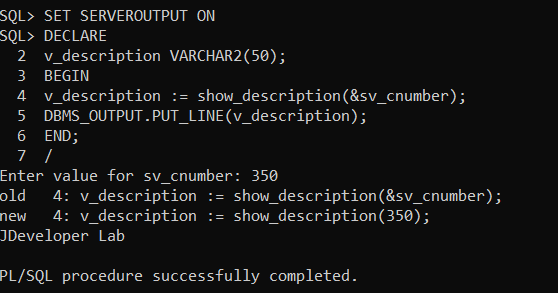
La sesión SQL\*PLUS devolvió “Function created”. Esto indica que la función se compiló con éxito.

El script crea la función show\_description. El encabezado de función indica que acepta un parámetro del tipo de datos NUMBER, i\_course\_no, y devuelve un valor de tipo de datos VARCHAR2.

La función declara una variable VARCHAR2 (5) llamada v\_description que se usa más adelante mediante la cláusula RETURN. Esta variable almacena el valor de la descripción del curso proporcionado en el tiempo de ejecución, y se inicializa a través de la instrucción SELECT INTO. Una vez inicializado, el valor de la variable v\_description se devuelve al entorno de llamada a través de la cláusula RETURN. Tenga en cuenta las dos excepciones empleadas por la función. La primera es la excepción NO\_DATA\_FOUND, la más probable de ocurrir. La segunda excepción es la excepción OTHERS, que es lo que se está utilizando como catchall para cualquier otro error que pueda ocurrir. Es importante tener en cuenta que tanto las secciones de excepción incluyen la cláusula RETURN, ya que la función siempre debe devolver un valor como el control de la ejecución se pasa al entorno de la llamada después de que se emita la cláusula RETURN

### 20.1.2. Haciendo uso de funciones

El siguiente ejercicio, muestra como usar la función creada previamente. Haciendo uso del siguiente bloque anónimo para ejecutar la función.



Debido a que el bloque PL/SQL tiene un parámetro léxico de &cnumber, se solicito ingresar un valor, en este caso ingresamos 350.

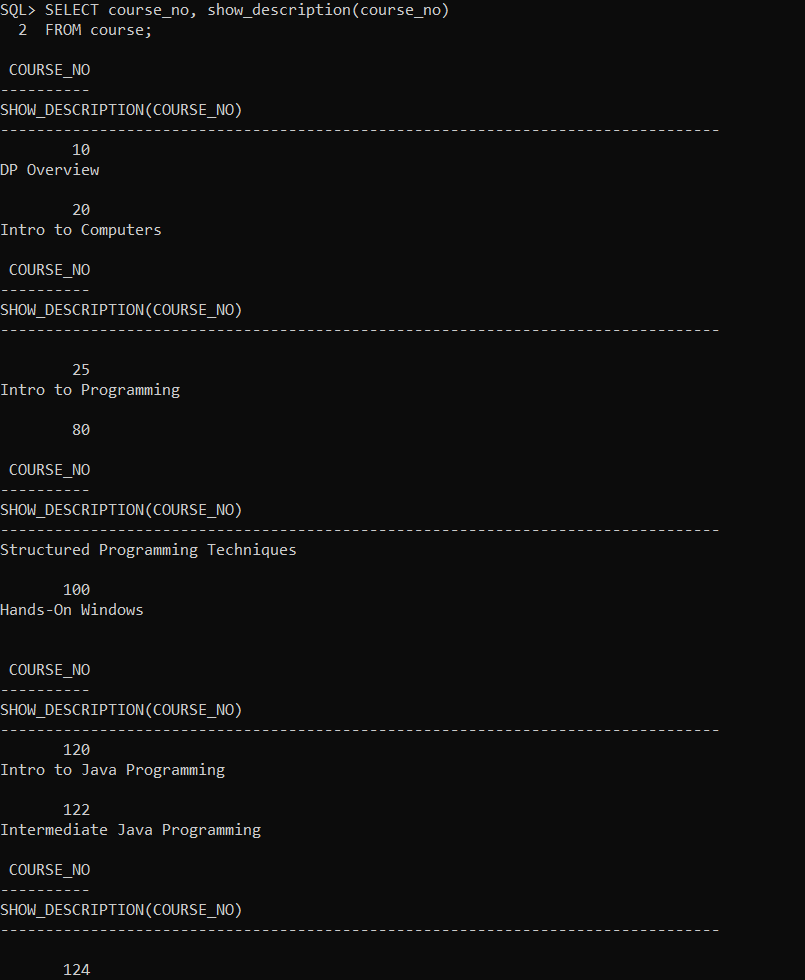
Esto significa que el valor de &sv\_cnumber se ha reemplazado por 350. La función show\_description devuelve un valor VARCHAR2, que es la descripción del curso para el número de curso que se pasa. El bloque PL / SQL inicializa el valor de v\_description con el retorno de la descripción\_de\_demostración función. Este valor se muestra con el paquete DBMS\_OUTPUT.

### 20.1.3. Funciones de invocación en sentencias SQL

Las funciones devuelven un único valor y pueden ser muy útiles en una instrucción SELECT.

En este ejemplo se muestra cómo usar funciones definidas por el usuario en las declaraciones SQL. El cual comienza creando una función que se puede usar en una instrucción SELECT.

Inicialmente probamos otro método para usar una función almacenada. Antes de escribir la siguiente instrucción SELECT, piense en lo que está haciendo la función show\_description. ¿Esta declaración producirá un error? Si no, ¿qué se mostrará?



Esta sentencia SELECT es idéntica a la siguiente sentencia SELECT:



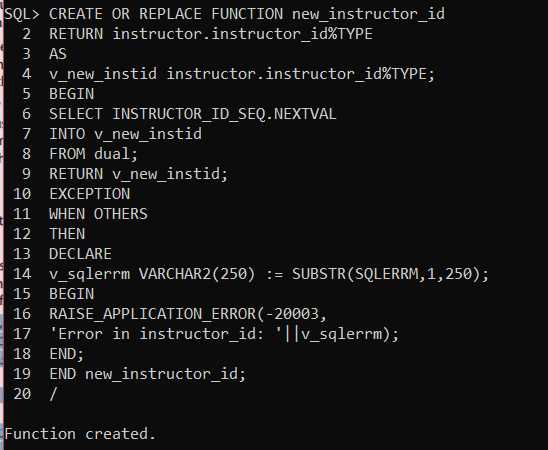
¿Qué está sucediendo?

Las funciones se pueden usar en una declaración de SQL. De hecho, lo hemos estado usando todo el tiempo.

### 20.1.4. Escribiendo funciones complejas

En este ejemplo se creo una función más compleja. En algunos casos, las funciones pueden volverse complejas y multifacéticas.

Inicialmente, creamos la función usando la siguiente secuencia de comandos. Y la pregunta entonces es: ¿Cuándo podríamos usar esta función? Sugerencia: lo usará en el paquete para el próximo capítulo.



Esta simple función se usa para generar una nueva ID de instructor. Si la secuencia no genera una nueva ID de instructor, la función genera una excepción.

Conclusion

En este capítulo, aprendendimos sobre:

* Creación y uso de funciones

Una función que se almacena en la base de datos es muy similar a un procedimiento en el sentido de que se trata de un bloque PL / SQL con nombre que puede tomar parámetros e invocarse.

Existen diferencias clave en la forma en que se crea y cómo se usa.

# 21. Paquetes

## 21.1. Los beneficios de usar paquetes

* Crea las especificaciones del paquete.
* Crear cuerpos de paquetes.
* Llamar a paquetes almacenados.
* Crea objetos privados
* Crear variables de paquete y cursores

Usar paquetes como un método para agrupar sus funciones y procedimientos ofrece numerosos beneficios.

El primero es que un paquete bien diseñado es una agrupación lógica de objetos como funciones, procedimientos, variables globales y cursores. Todo el código (árbol de análisis sintáctico y pseudocódigo [código-p]) se carga en la memoria (el área global compartida [SGA] del servidor de Oracle) en la primera llamada del paquete. Esto significa que la primera llamada al paquete es muy costosa (implica mucho procesamiento en el servidor), pero todas las llamadas posteriores resultan en un mejor rendimiento. Por lo tanto, los paquetes se usan a menudo en aplicaciones que usan procedimientos y funciones repetidamente.

Los paquetes le permiten incorporar algunos de los conceptos involucrados en la programación orientada a objetos, aunque PL / SQL no es un lenguaje de programación orientado a objetos "verdadero". El paquete PL / SQL le permite recopilar funciones y procedimientos similares y darles un contexto. También te permite encapsularlos. Debido a que todo el código del paquete se ha cargado en la memoria, también puede escribir el código para que se coloquen fragmentos de código similares en el paquete de manera que permita que múltiples procedimientos y funciones los llamen. Debería hacer esto si la lógica para el cálculo es bastante intensa y desea mantenerla en un solo lugar.

El uso de paquetes ofrece un nivel adicional de seguridad. Cuando un usuario ejecuta un procedimiento en un paquete (o procedimientos y funciones almacenados), el procedimiento opera con los mismos permisos que su propietario. Los paquetes le permiten crear funciones y procedimientos privados, que solo se pueden invocar desde otras funciones y procedimientos del paquete. Esto obliga a ocultar la información. La estructura del paquete fomenta así el diseño de arriba hacia abajo.

**LA ESPECIFICACIÓN DEL PAQUETE**

La especificación del paquete contiene información sobre el contenido del paquete, pero no el código para los procedimientos y funciones. También contiene declaraciones de variables globales / públicas. Cualquier cosa colocada en la sección de declaración de un bloque PL / SQL puede codificarse en una especificación de paquete. Todos los objetos colocados en la especificación del paquete se llaman objetos públicos.

Cualquier función o procedimiento que no esté en la especificación del paquete pero codificado en un cuerpo de paquete se denomina función o procedimiento privado. Cuando se llaman procedimientos y funciones públicas desde un paquete, el programador que escribe el proceso de "llamada" necesita solo la información en la especificación del paquete. Esto proporciona toda la información requerida necesaria para llamar a uno de los procedimientos o funciones dentro del paquete. La sintaxis para la especificación del paquete es la siguiente; tenga en cuenta que la información opcional se incluye entre corchetes:

PACKAGE package\_name

IS

[declarations of variables and types]

[specifications of cursors]

[specifications of modules]

END [package\_name];

**EL CUERPO DEL PAQUETE**

El cuerpo del paquete contiene el código ejecutable real para los objetos descritos en la especificación del paquete. El cuerpo del paquete contiene código para todos los procedimientos y funciones descritos en la especificación. También puede contener código para objetos no declarados en la especificación. El último tipo de objeto empaquetado es invisible fuera del paquete y se denomina oculto. Al crear paquetes almacenados, puede compilar la especificación y el cuerpo del paquete por separado. La sintaxis para el cuerpo del paquete es la siguiente; tenga en cuenta que la información opcional se incluye entre corchetes

PACKAGE BODY package\_name

IS

[declarations of variables and types]

[specification and SELECT statement of cursors]

[specification and body of modules]

[BEGIN

executable statements]

[EXCEPTION

exception handlers]

END [package\_name];

**REGLAS PARA EL CUERPO DEL PAQUETE**

Debe seguir varias reglas en el código del cuerpo del paquete:.

* Debe haber una coincidencia exacta entre el cursor y los encabezados de los módulos y sus definiciones en la especificación del paquete.
* No repita en el cuerpo la declaración de variables, excepciones, tipos o constantes en la especificación.
* Cualquier elemento declarado en la especificación puede ser referenciado en el cuerpo.

**REFERENCIA DE ELEMENTOS DE PAQUETE**

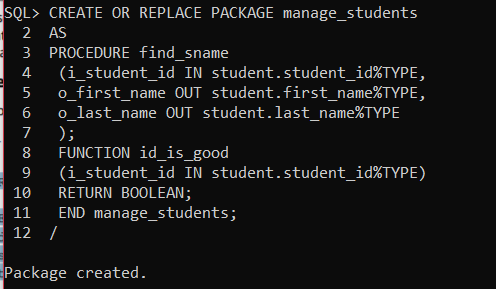
Utilice la siguiente sintaxis cuando llame a elementos empaquetados desde fuera del paquete:

package\_name.element

No necesita calificar elementos cuando se declaran y referencian dentro del cuerpo del paquete o cuando se declaran en una especificación y se hace referencia dentro del paquete cuerpo del mismo paquete.

### 21.1.1. Creación de especificaciones de paquetes

Con este ejercicio se visualiza el registro basado en tablas y basados en cursor. Creamos el siguiente script PL/SQL.

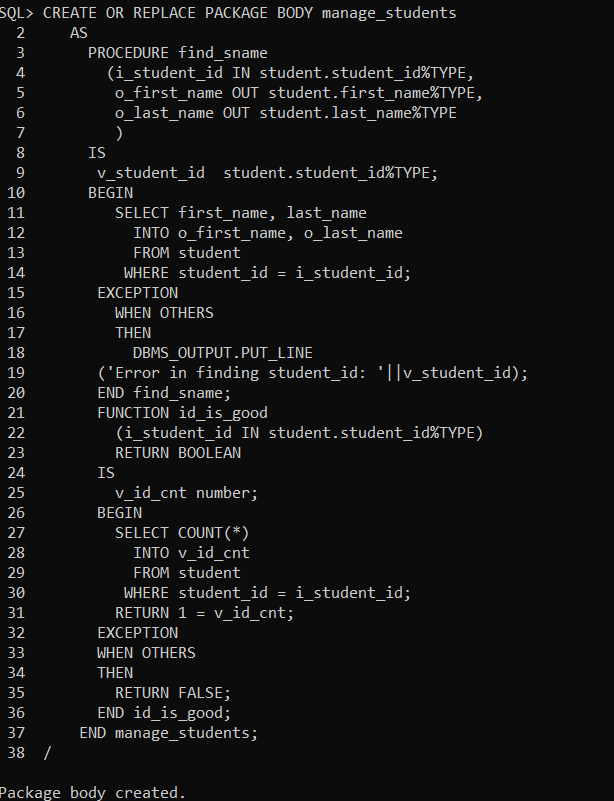


¿Qué esta sucediendo?  
La especificación para el paquete manage\_students se compila en la base de datos. La especificación para el paquete indica que hay un procedimiento y una función. El procedimiento, find\_sname, requiere un parámetro IN (el ID del estudiante) y devuelve dos parámetros OUT(los nombres y apellidos del alumno)

La función, id\_is\_good, toma un solo parámetro de un ID de estudiante y devuelve un booleano (verdadero o falso). Aunque el cuerpo aún no se ha ingresado en la base de datos, el paquete aún está disponible para otras aplicaciones. Por ejemplo, si incluyó una llamada a uno de estos procedimientos en otro procedimiento almacenado, ese procedimiento se compilaría (pero no se ejecutaría).

### 21.1.2. Creación de cuerpos de paquetes

En este ejercicio, creamos el cuerpo de los paquetes manage\_students, que se especificaron en la sección anterior.



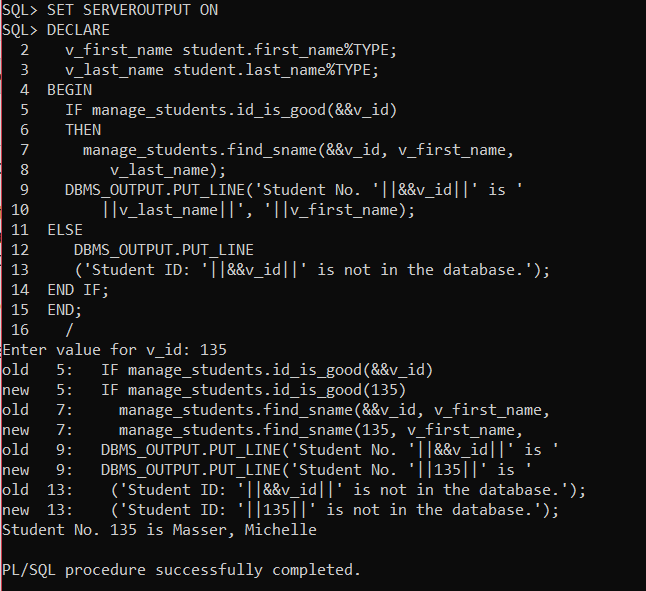
¿Qué esta sucediendo?

La especificación para el paquete manage\_students se compila en la base de datos. La especificación para el paquete indica que hay un procedimiento y una función.

El procedimiento, find\_sname, requiere un parámetro IN (el ID del estudiante) y devuelve dos parámetros OUT: los nombres y apellidos del alumno. La función, id\_is\_good, toma un parámetro único de una identificación de estudiante y devuelve un booleano (verdadero o falso). Aunque el cuerpo no tiene aún se ha ingresado en la base de datos, el paquete aún está disponible para otras aplicaciones. Por ejemplo, si incluyó una llamada a uno de estos procedimientos en otro procedimiento almacenado, ese el procedimiento compilaría (pero no se ejecutaría).

### 21.1.3. Llamando paquetes almacenados

En este ejercicio, hacemos uso de elementos del paquete manage\_student en otro bloque de código.



¿Qué esta sucediendo?

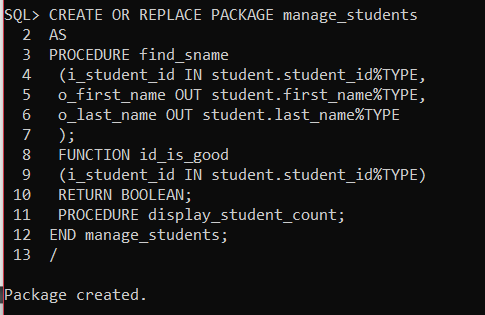
Este ejemplo muestra cómo se ejecuta un procedimiento dentro de un paquete.

Este es un bloque PL / SQL correcto para ejecutar la función y el procedimiento en el paquete manage\_students. Si se ingresa un student\_id existente, se muestra el nombre del estudiante. Si el ID no es válido, se muestra un mensaje de error.

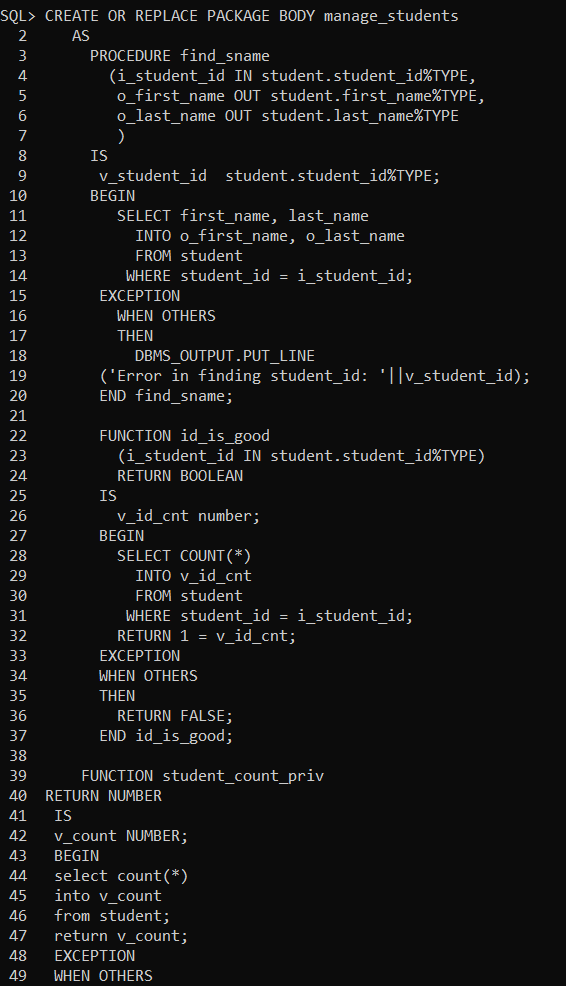
### 21.1.4 Creación de objetos privados

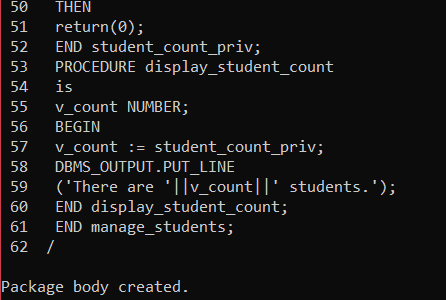
Los elementos públicos son elementos definidos en la especificación del paquete. Si un objeto está **definido solo en el cuerpo del paquete, es privado.** Los elementos privados no pueden ser accedidos directamente por ningún programa fuera del paquete. Puede pensar en la especificación del paquete como un "menú" de elementos empaquetados que están disponibles para los usuarios. Otros objetos pueden estar trabajando detrás de escena, pero son inaccesibles. No pueden ser llamados ni utilizados de ninguna manera. Están disponibles como parte del "menú" interno del paquete y solo pueden ser llamados por otros elementos del paquete.

Por ejemplo, se va a reemplazar las últimas líneas de la especificación del paquete manage\_students con lo siguiente y recompilar.



Ahora reemplazamos el extremo del cuerpo con lo siguiente:





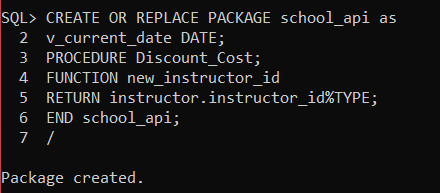
¿Qué hemos agregado al paquete manage\_student?

Hemos agregado una función privada (student\_count\_priv) y un procedimiento público (display\_student\_count), llamado a la función privada.

### 21.1.5. Creación de variables de paquete y cursores

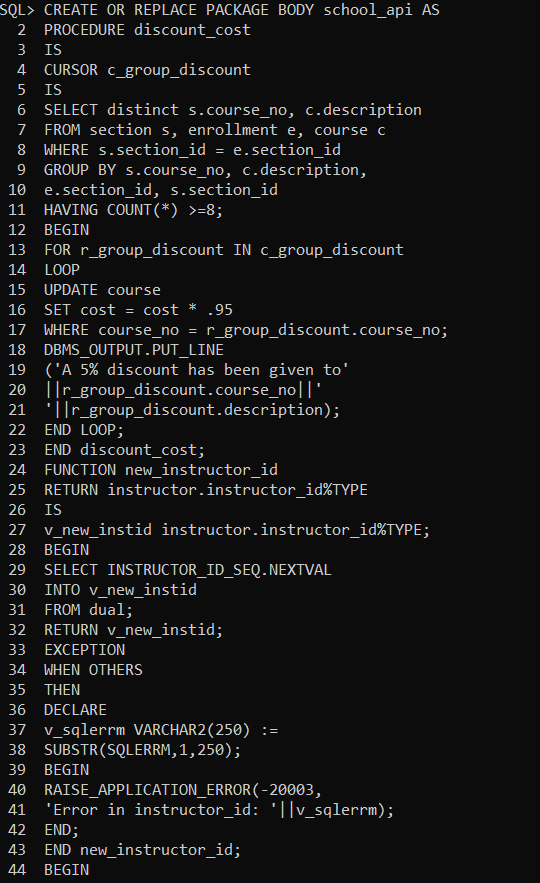
La primera vez que se llama un paquete dentro de una sesión de usuario, el código en la sección de inicialización del paquete se ejecuta si existe. Esto se hace solo una vez y no se repite si el usuario llama a otros procedimientos o funciones para ese paquete. Las variables, los cursores y los tipos de datos definidos por el usuario utilizados por numerosos procedimientos y funciones se pueden declarar una vez al comienzo de la especificación del paquete. Luego pueden ser utilizados por las funciones y procedimientos dentro del paquete sin tener que ser declarados nuevamente.

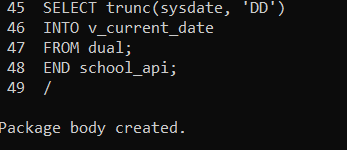
Por ejemplo, agregamos una variable global de paquete llamada v\_current\_date a student\_api.



Y agregamos una sección de inicialización que asigne el sysdate actual a la variable v\_current\_date.

Esta variable se puede usar en cualquier procedimiento del paquete que necesite utilizar la fecha actual.





## 21.2. Variables del cursor

* Uso de las variables de cursor

Hasta este punto de este libro, ha visto que los cursores se utilizan para recopilar datos específicos de una única instrucción SELECT. Al comienzo de este capítulo, aprendió cómo llevar una cantidad de procedimientos a un programa grande llamado paquete.

Un paquete puede tener un cursor utilizado por algunos procedimientos. En este caso, cada uno de los procedimientos que utiliza el mismo cursor debería declarar, abrir, recuperar y cerrar el cursor. En la versión actual de PL / SQL, los cursores se pueden declarar y manipular como cualquier otra variable PL / SQL. Este tipo de variable se denomina variable de cursor o REF CURSOR.

Una variable de cursor es solo una referencia o manejador de un cursor estático. Permite a un programador pasar esta referencia al mismo cursor entre todas las unidades del programa que necesitan acceso al cursor. Una variable de cursor vincula dinámicamente la instrucción SELECT del cursor en el tiempo de ejecución. Los cursores explícitos se utilizan para nombrar un área de trabajo que contiene la información de una consulta multirruta. Se puede usar una variable de cursor para señalar el área en la memoria donde se almacena el resultado de una consulta multirruta.

El cursor siempre se refiere a la misma información en un área de trabajo, mientras que una variable de cursor puede señalar diferentes áreas de trabajo. Los cursores son estáticos, pero las variables de cursor se pueden ver como dinámicas porque no están vinculadas a ninguna consulta. Las variables del cursor le brindan un fácil acceso a la recuperación centralizada de datos. Puede usar una variable de cursor para pasar el conjunto de resultados de una consulta entre procedimientos almacenados y varios clientes.

Un área de trabajo de consulta permanece accesible siempre que una variable de cursor lo señale. Por lo tanto, puede pasar libremente una variable de cursor de un alcance a otro. Los dos tipos de variables de cursor son fuertes y débiles. Para ejecutar una consulta multirruta, el servidor Oracle abre un área de trabajo llamada cursor para almacenar información de procesamiento. Para acceder a la información, puede nombrar el área de trabajo o utilizar una variable de cursor que lo señale. Un cursor siempre se refiere a la misma área de trabajo, pero una variable de cursor puede referirse a diferentes áreas de trabajo. Por lo tanto, los cursores y las variables del cursor no son intercambiables. Un cursor explícito es estático y está asociado a una declaración SQL. Una variable de cursor se puede asociar con diferentes declaraciones en tiempo de ejecución. En primer lugar, utiliza una variable de cursor para pasar un puntero a conjuntos de resultados de consulta entre subprogramas almacenados PL / SQL y varios clientes, como una aplicación cliente Oracle Developer Forms.

Ninguno de ellos posee el conjunto de resultados; simplemente comparten un puntero al área de trabajo de consulta que almacena un puntero al área de trabajo de consulta que almacena el conjunto de resultados. Puede declarar una variable de cursor en del lado del cliente, ábrelo y sácalo del lado del servidor, y luego sigue obteniéndolo de él el lado del cliente.

Las variables del cursor difieren de los cursores de la misma manera que las constantes difieren de las variables. UN el cursor es estático; una variable de cursor es dinámica. En PL / SQL una variable de cursor tiene un REF CURSOR tipo de datos, donde REF significa referencia y CURSOR representa la clase del objeto. Tú ahora aprenderá la sintaxis para declarar y usar una variable de cursor.

Para crear una variable de cursor, primero debe definir un tipo REF CURSOR y luego declarar una variable de ese tipo.

Antes de declarar REF CURSOR de tipo fuerte, debe declarar un registro que tenga el tipos de datos del conjunto de resultados de la instrucción SELECT que planea usar (tenga en cuenta que esto no es necesario para un REF CURSOR débil).

Por ejemplo

TYPE inst\_city\_type IS RECORD

(first\_name instructor.first\_name%TYPE;

last\_name instructor.last\_name%TYPE;

city zipcode.city%TYPE;

state zipcode.state%TYPE)

En segundo lugar, debe declarar un tipo de datos compuesto para la variable del cursor que sea del tipo REF CURSOR. La sintaxis es la siguiente:

TYPE ref\_type\_name es REF CURSOR [RETURN return\_type];

ref\_type\_name es un tipo especificado en declaraciones posteriores. return\_type es un tipo de registro para un cursor fuerte; un cursor débil no tiene un tipo de devolución específico, pero puede manejar cualquier combinación de elementos de datos en una instrucción SELECT. La palabra clave REF CURSOR indica que el nuevo tipo será un puntero al tipo definido. return\_type indica el tipo de lista SELECT que la variable de cursor finalmente devuelve. El tipo de devolución debe ser un tipo de registro.

Estos son los pasos clave para manejar una variable de cursor:

1. Defina y declare la variable del cursor.

Abra la variable del cursor. Asociar la variable de cursor con una instrucción SELECT multirruta, ejecutar la consulta e identificar el conjunto de resultados. Una instrucción OPEN FOR puede abrir la misma variable de cursor para diferentes consultas. No es necesario que cierre una variable de cursor antes de volver a abrirla. Tenga en cuenta que cuando vuelve a abrir una variable de cursor para una consulta diferente, la consulta anterior se pierde. Una buena técnica de programación sería cerrar las variables del cursor antes de volver a abrirlas más adelante en el programa.

1. Obtener filas del conjunto de resultados.

Recupere filas del conjunto de resultados de a una por vez. Tenga en cuenta que el tipo de retorno de la variable de cursor debe ser compatible con la variable nombrada en la cláusula INTO de la instrucción FETCH.

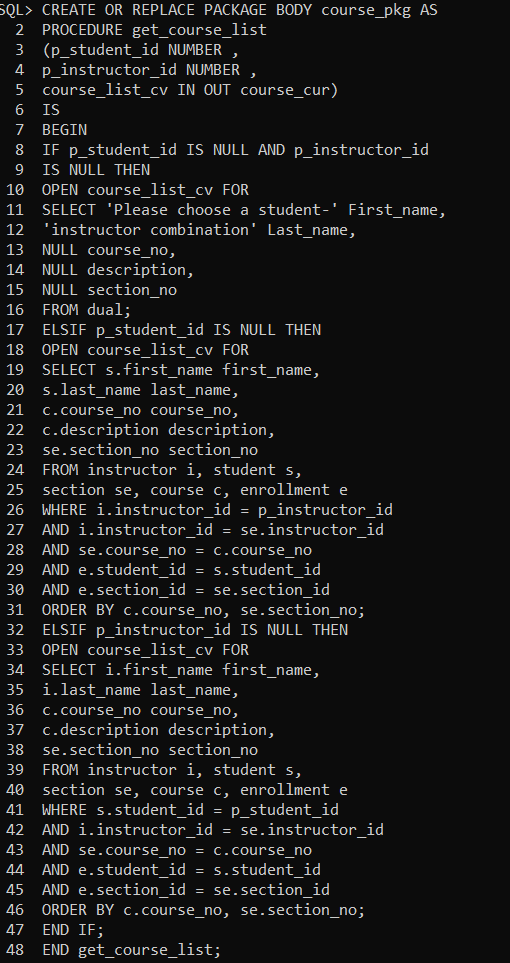
La instrucción FETCH recupera filas del conjunto de resultados de a una por vez. PL / SQL verifica que el tipo de retorno de la variable de cursor sea compatible con la cláusula INTO de la instrucción FETCH. Para cada valor de columna de consulta devuelto, la cláusula INTO debe tener una variable typecomparable. Además, el número de valores de columna de consulta debe ser igual al número de variables. En caso de una falta de coincidencia en número o tipo, el error ocurre en tiempo de compilación para variables de cursor fuertemente tipadas y en tiempo de ejecución para variables de cursor débilmente tipadas.

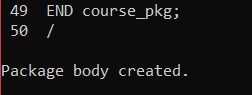
1. Cierre la variable del cursor.

El siguiente es un ejemplo completo que muestra el uso de una variable de cursor en un paquete. (siguiente sección).

### 21.2.1. Uso de variables de cursor

En este ejemplo se explica porqué el paquete tiene dos declaraciones TYPE diferentes. También explica cómo el procedimiento get\_course\_list usa la variable del cursor.





¿Qué esta sucediendo?

En este script la primera declaración TYPE es para el tipo de registro course\_rec\_ type. Este tipo de registro se declara para definir el conjunto de resultados de las instrucciones SELECT que se utilizarán para la variable del cursor.

Cuando los elementos de datos en un registro no coinciden con una sola tabla, es necesario crear un tipo de registro. La segunda declaración TYPE es para la variable de cursor, también conocida como REF CURSOR. La variable tiene el nombre course\_cur y se declara como un cursor fuerte, lo que significa que solo se puede usar para un solo tipo de registro. El tipo de registro es course\_rec\_ type.

El procedimiento get\_course\_list en el course\_pkg está hecho para que pueda devolver una variable de cursor que contenga tres conjuntos de resultados diferentes. Cada conjunto de resultados es del mismo tipo de registro. El primer tipo es para cuando los dos parámetros IN de ID de alumno e ID de instructor son nulos.

Esto produce un conjunto de resultados que es un mensaje. Elija una combinación estudiante-instructor. La siguiente manera en que se ejecuta el procedimiento es si el instructor\_id se transfiere pero el student\_id es nulo. (Tenga en cuenta que la lógica del procedimiento es inversamente negativa. Decir en la segunda cláusula de la instrucción IF p\_student\_id IS NULL significa cuando se pasa el instructor\_id.)

Esto ejecuta una instrucción SELECT para rellenar la variable de cursor que contiene una lista de todos los cursos que este instructor enseña y los estudiantes inscritos en estas clases. La última forma en que esto puede ejecutarse es para un student\_id y no instructor\_id. Esto produce un conjunto de resultados de todos los cursos en los que el alumno está inscrito y los instructores para cada sección. Además, tenga en cuenta que, después de abrir la variable de cursor, nunca se cierra hasta que la cierre específicamente.

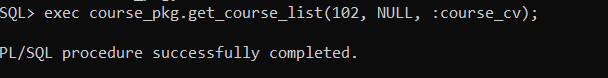
Se tiene que crear la variable SQL\*PLUS para que sea un tipo de variable de cursor. Con el siguiente sentencia:

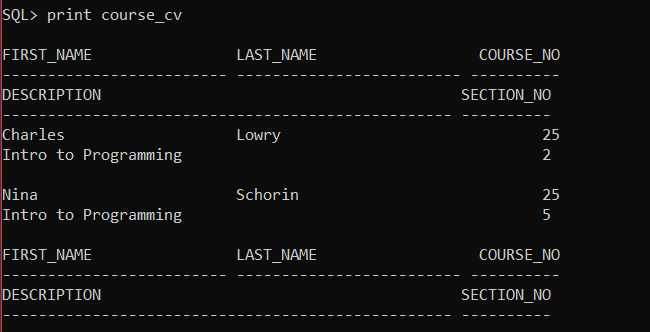
VARIABLE course\_cv REFCURSOR

Ahora, vamos a ejecutar el procedimiento course\_pkg.get\_course\_list, con tres tipos diferentes de combinaciones de variables para mostrar los tres posibles conjuntos de resultados.

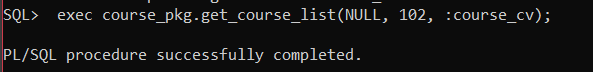
Después de ejecutar el procedimiento, mostramos los valores de la variable SQL \* Plus que declaró en la pregunta

Hay tres formas de ejecutar este procedimiento. La primera forma es pasar una identificación de estudiante pero no una ID de instructor:



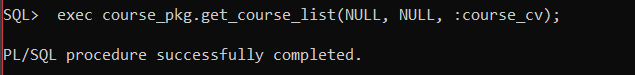


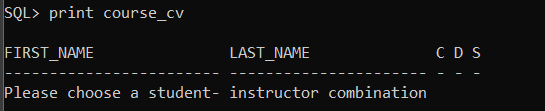
El siguiente método es pasar un ID instructor pero no un ID student:





El ultimo método es no pasar la ID del estudiante ni del instructor





**REGLAS PARA UTILIZAR LAS VARIABLES DEL CURSOR.**

* Las variables de cursor no se pueden definir en una especificación de paquete.
* No puede usar variables de cursor con subprogramas remotos en otro servidor, por lo que no puede pasar variables de cursor a un procedimiento que se llama a través de un enlace de base de datos. .
* No use FOR UPDATE con OPEN FOR en el procesamiento de una variable de cursor. .
* No puede usar operadores de comparación para probar variables de cursor para igualdad, desigualdad o nulidad. .
* A una variable de cursor no se le puede asignar un valor nulo. .
* Un tipo REF CURSOR no se puede usar en una instrucción CREATE TABLE o VIEW, porque no hay un tipo de datos equivalente para una columna de base de datos. .
* Un procedimiento almacenado que usa una variable de cursor puede usarse solo como fuente de datos de bloque de consulta; no se puede usar para una fuente de datos de bloques DML. El uso de REF CURSOR es ideal para consultas que dependen solo de las variaciones en las sentencias SQL, no PL / SQL. .
* No puede almacenar variables de cursor en una matriz asociativa, tabla anidada o varray.
* Si pasa una variable de cursor de host a PL / SQL, no puede recuperarla en el servidor a menos que también la abra allí en la misma llamada al servidor.

## 21.3. Extendiendo el paquete

Después de completar este laboratorio, podrás. Extienda el paquete En esta práctica de laboratorio, utiliza conceptos aprendidos previamente para ampliar los paquetes que ha creado y crear uno nuevo. Solo a través de ejercicios extensos te sentirás más cómodo con la programación en PL / SQL. Al escribir su código PL / SQL, es muy importante que considere cuidadosamente todos los aspectos de los requisitos del negocio. Una buena regla empírica es pensar en el futuro y escribir su código en componentes reutilizables para que sea fácil extender y mantener su código PL / SQL.

### 21.3.1. Extensión del paquete

Cree una nueva especificación de paquete llamada manage\_grades. Este paquete realizará una serie de cálculos en grados y necesitará dos cursores de nivel de paquete. El primero es para tipos de grado y se llamará c\_grade\_type. Tendrá un parámetro IN de una ID de sección. Enumerará todos los tipos de calificación (como cuestionarios o tareas) para una sección determinada que son necesarios para calcular la calificación de un alumno en esa sección. Los elementos devueltos del cursor serán el código de tipo de calificación, el número de ese tipo de calificación para esta sección, el porcentaje de la calificación final y el indicador de caída más baja. Primero, escriba una instrucción SELECT para asegurarse de tener los elementos correctos y luego escriba esto como un cursor en el paquete.

Conclusion

En este capítulo, aprendimos sobre.

* Los beneficios de usar paquetes.
* Variables del cursor
* Extendiendo el paquete

Un paquete es una colección de objetos PL / SQL agrupados bajo un nombre de paquete. **Los paquetes incluyen procedimientos, funciones, cursores, declaraciones, tipos y variables.** Recolectar objetos en un paquete ofrece numerosos beneficios

Unas de las ventajas de los paquetes son los siguientes

* Permite modularizar el diseño de nuestra aplicación
* Otorga flexibilidad al momento de diseñar la aplicación
* Agrega mayor funcionalidad a nuestro desarrollo
* Agrega mayor funcionalidad a nuestro desarrollo