1º CFGS Desarrollo De Aplicaciones Multiplataforma

UD 5. PL/SQL

Módulo: Bases de datos

Centro de Enseñanza
Gregorio Fernández

Macarena Cuenca Carbajo

1.Introducción

- Oracle incorpora el lenguaje **PL/SQL**.
- Lenguaje **procedimental** y **orientado a objetos**.
- Los programas creados con PL/SQL se pueden almacenar en la base de datos como cualquier otro objeto.
- Se ejecutan en el servidor.
- Estructura básica: el **bloque**

2. Bloques anónimos

- No tienen nombre.
- Se ejecutan en el servidor pero no se almacenan en él.
- Sintáxis:

```
[ DECLARE
      <declaraciones> ]
  BEGIN
      <órdenes>
  [ EXCEPTION
      <gestión de excepciones> ]
  END;
```

3. Tipos de datos

- PL/SQL dispone de los mismos tipos de datos que SQL.
- Además de otros propios como **boolean**.

4. Variables

- Deben declararse en la sección DECLARE antes de su uso.
- Sintáxis para **declarar** una variable:

  <nombre_de_variable> <tipo> [**NOT NULL**] [{ : = | **DEFAULT** } <valor>];
- Para cada variable se debe especificar el tipo.
- Si no se inicializan las variables en PL/SQL su valor es NULL.
- En lugar de indicar explícitamente el tipo y la longitud de una variable existe la posibilidad de utilizar los atributos **%TYPE** y **%ROWTYPE**.

4. Variables

- **%TYPE**: declara una variable del mismo tipo que otra, o que una columna de una tabla. Su formato es:

  nombre_variable nombre_objeto**%TYPE**;

- **%ROWTYPE:** declara una variable de tipo registro de una tabla. Su formato es:

  nombre_variable nombre_objeto**%ROWTYPE**;

- También se pueden declarar **constantes** de la siguiente forma:

  <nombre_constante> **CONSTANT** <tipo> := <valor>;

5. Operadores

- **Asignación (:=).** Asigna un valor a una variable.
- **Concatenación (||).** Une dos o más cadenas.
- **Comparación (=, !=, <, >, <=, >= , IN, IS NULL, LIKE, BETWEEN,…).**
- **Aritméticos (+, -, *, /).** Se emplean para realizar cálculos. Algunos de ellos se pueden utilizar también con fechas.
- **Lógicos (AND, OR y NOT).**

6. Funciones

- En PL/SQL se pueden utilizar todas las funciones de SQL.
- Algunas de ellas, como las de agrupamiento (AVG, MIN, MAX, COUNT, SUM, STDDEV, etc.), solamente se pueden usar en la SELECT.

7. Comentarios

- Estos comentarios pueden ser:
  - ❖ **De línea** con "--"
  - ❖ **De varias líneas** con /* ... */

8.1. Estructuras de control – CONDICIONES

```
IF    <condición> THEN
      instrucciones;
END IF;
```

```
IF    <condición> THEN
      instrucciones1;
ELSE
      instrucciones2;
END IF;
```

```
IF      <condición1> THEN
        instrucciones1;
ELSIF <condición2> THEN
        instrucciones2;
ELSIF <condición3> THEN
        instrucciones3;
. . .
[ELSE
        otras instrucciones;]
END IF;
```

```
CASE  <expresión>
WHEN <valor1> THEN
        instrucciones1;
WHEN <valor2> THEN
        instrucciones2;
WHEN <valor3> THEN
        instrucciones3;
. . .
[ELSE
      <otras instrucciones; ]
END CASE;
```

8.2. Estructuras de control - BUCLES

```
LOOP
    instrucciones;
    EXIT [WHEN <condición>];
END LOOP;
```

```
WHILE <condición>
LOOP
    instrucciones;
END LOOP;
```

```
FOR   <variable_control> IN [REVERSE]<valor_inicial>..<valor_final>
LOOP
    instrucciones;
END LOOP;
```

9. Sentencias SQL

- Desde PL/SQL se puede ejecutar cualquier orden INSERT, DELETE y UPDATE.
- También permite ejecutar SELECT. Pero en este caso el resultado no se muestra en el terminal del usuario, sino que queda en un área de memoria denominada **CURSOR** a la que podremos acceder utilizando variables.
- El formato básico para ejecutar una sentencia SELECT en PL/SQL es:

SELECT <columna/s> **INTO** <variable/s> **FROM** <tabla> **[WHERE …];**

Centro de Enseñanza
Gregorio Fernández

10. Excepciones

- Las **excepciones** sirven para tratar situaciones anómalas.
- En Oracle están disponibles excepciones predefinidas correspondientes a algunos de los errores más frecuentes que se producen al trabajar con la base de datos, como por ejemplo:
  - ✓ **NO_DATA_FOUND**. Cuando una SELECT INTO no ha devuelto ningún valor.
  - ✓ **TOO_MANY_ROWS**. Cuando una SELECT INTO ha devuelto más de una fila.
- Las excepciones se disparan automáticamente al producirse los errores asociados.
- La sección EXCEPTION es la encargada de gestionar las excepciones mediante los manejadores (WHEN).
- Cuando PL/SQL detecta una excepción, automáticamente pasa el control del programa a la sección EXCEPTION. Allí buscará un manejador (WHEN) para la excepción producida, o uno genérico (WHEN OTHERS).

11. Procedimientos y Funciones

- Son bloques PL/SQL que tienen un nombre, pueden recibir parámetros y devolver valores.
- Se almacenan en la base de datos.
- Podemos ejecutarlos invocándolos desde otros bloques.
- Un procedimiento se diferencia de una función, en que el primero no devuelve ningún valor por sí mismo, y la función si que puede hacerlo.

11. Procedimientos y Funciones

- Sintáxis para **crear** un procedimiento:

```
CREATE [OR REPLACE] PROCEDURE <nombre_procedimiento> [(param1 tipo1, param2 tipo2, …)]
IS|AS
    [<declaraciones>;]
BEGIN
    <instrucciones>;
[EXCEPTION
    <excepciones>;]
END [<nombre_procedimiento>];
```

11. Procedimientos y Funciones

- Sintáxis para **crear** una función:

```
CREATE [OR REPLACE] FUNCTION <nombre_función> [(param1 tipo1, param2 tipo2, …)]
RETURN <tipo_valor_devuelto>
IS|AS
      [<declaraciones>;]
BEGIN
      <instrucciones>;
      RETURN <expresión>;
[EXCEPTION
      <excepciones>;]
END [<nombre_función>];
```

11. Procedimientos y Funciones

- Llamada:

```
nombre_procedimiento[(lista_parámetros)];
variable :=  nombre_función [(lista_parámetros)];
```

11.1. Procedimientos y Funciones - Parámetros

- Podemos hacer el paso de parámetros de las siguiente formas:
  - ✓ **Notación posicional:** se pasan los valores en el mismo orden en el que se han definido.
  - ✓ **Notación nominal:** se pasan los valores en cualquier orden, indicando el nombre del parámetro, a continuación el símbolo **=>** y después el valor.
  - ✓ **Notación mixta:** se usan ambas notaciones con la restricción de que la notación posicional debe preceder a la nominal.

11.2. Procedimientos y Funciones

- Sintáxis para **compilar** un procedimiento/función:

  ALTER {PROCEDURE | FUNCTION} nombre_subprograma **COMPILE;**

- Sintáxis para **eliminar** un procedimiento/función:

  DROP {PROCEDURE | FUNCTION} nombre_subprograma**;**

11.3. Procedimientos y Funciones - Vistas

- **USER_OBJECTS:** objetos que son propiedad del usuario.
- **USER_PROCEDURES:** procedimientos que son propiedad del usuario.
- **USER_SOURCE:** código fuente almacenado en el esquema del usuario.

12. Cursores

- Un **cursor** es una zona de memoria utilizada por Oracle para analizar e interpretar cualquier sentencia SQL.
- Existen dos tipos de cursores: los **implícitos** y los **explícitos**.
- Hasta el momento hemos utilizado *cursores implícitos*, que son generados y gestionados por Oracle. Plantean diversas limitaciones.

  La más importante es que la consulta debe devolver una fila (y sólo una), de lo contrario, se produciría un error.
- Por ello, dado que normalmente una consulta devolverá varias filas, se suelen manejar *cursores explícitos*.

12.1. Cursores - Explícitos

- Son generados y gestionados por el usuario.
- Se utilizan para trabajar con consultas que pueden devolver más de una fila.
- Cuatro operaciones básicas para trabajar con un cursor explícito:

1. **Declaración del cursor** en la zona de declaraciones:

   CURSOR <nombre_cursor> **IS** <sentencia SELECT>;

2. **Apertura del cursor** en la zona de instrucciones**:**

   OPEN <nombre_cursor>;

3. **Recogida de información** almacenada en el cursor:

   FETCH <nombre_cursor> **INTO** {<variable>|<lista_variables>};

4. **Cierre del cursor**:

   CLOSE <nombre_cursor>;

12.1. Cursores - Explícitos

- La instrucción **OPEN** ejecuta automáticamente la sentencia SELECT asociada y sus resultados se almacenan en las estructuras internas de memoria manejadas por el cursor.
- Cada **FETCH** recupera una fila y el cursor avanza automáticamente a la fila siguiente.
- La fila recuperada por el FETCH puede:

  a) Cargarse en una variable que recogerá la información de todas las columnas. Puede declararse de esta forma:

  <variable> <nombre_cursor>**%ROWTYPE;**

  b) O una lista de variables. Cada una recogerá la columna correspondiente de la cláusula SELECT, por tanto, serán del mismo tipo que las columnas.

12.2. Cursores - Atributos

- Hay cuatro atributos para consultar detalles de la situación del cursor (implícito o explícito):
  - **%FOUND**. Devuelve verdadero si el último FETCH ha recuperado algún valor, en caso contrario, devuelve falso. Si el cursor no estaba abierto devuelve error, y si estaba abierto pero no se habla ejecutado aún ningún FETCH, devuelve NULL. Se suele utilizar como condición de continuación en bucles.
  - **%NOTFOUND.** Hace lo contrario que el atributo anterior. Se suele utilizar como condición de salida en bucles.
  - **%ROWCOUNT.** Devuelve el número de filas recuperadas hasta el momento por el cursor (número de FETCH realizados satisfactoriamente).
  - **%ISOPEN.** Devuelve verdadero si el cursor está abierto.

12.2. Cursores - Atributos

- El cursor implícito se llama SQL y dispone también de los cuatro atributos mencionados, que pueden facilitarnos información sobre la ejecución de la última instrucción SELECT INTO, INSERT, UPDATE y DELETE.
- En ese caso, el cursor se cierra inmediatamente después de procesar cada orden SQL, por lo que el atributo *SQL%ISOPEN* siempre devolverá FALSE.

12.2. Cursores – FOR ... LOOP

- Los cursores FOR ... LOOP simplifican el trabajo con cursores, ya que excepto la declaración del cursor, todas las tareas las realiza manera implícita.
- Para usar este tipo de cursores:

1. Se **declara** el cursor en la sección declarativa (como cualquier otro cursor):

```
CURSOR <nombre_cursor> IS <sentencia SELECT>;
```

2. Y se **procesa** el cursor utilizando el siguiente formato:

```
FOR <variable_registro> IN <nombre_cursor>
LOOP
...
END LOOP;
```

12.2. Cursores – FOR ... LOOP

- Al entrar en el bucle:
  - Se **abre** el cursor de manera automática.
  - Se **declara** implícitamente la variable *variable_registro* de tipo cursor%ROWTYPE y se ejecuta un FETCH implícito, cuyo resultado quedará en *variable_registro*.
  - A continuación, se realizarán las acciones que correspondan hasta llegar al END LOOP, que sube de nuevo al FOR ... LOOP ejecutando el siguiente FETCH implícito, y depositando otra vez el resultado en *variable_registro*, y así sucesivamente, hasta procesar la última fila de la consulta. En ese momento, se producirá la salida del bucle y se cerrará automáticamente el cursor.

12.3. Cursores – Alias en las columnas

- Cuando utilizamos variables de registro declaradas del mismo tipo que el cursor o que la tabla, los campos tienen el mismo nombre que las columnas correspondientes.
- Cuando esas consultas son expresiones, para referenciarlas debemos indicar un alias en la columna:

```
CURSOR c1 IS
SELECT dept_no, count( *) n_emp, sum(salario + NVL(comision,0)) suma
FROM emple
GROUP BY dept_no;
```

12.4. Cursores con parámetros

- La **declaración** de un cursor con parámetros se realiza indicando la lista de parámetros entre paréntesis a continuación del nombre del cursor.

  CURSOR <nombre_cursor> [(parámetro1, parámetro2, ..)]

  IS SELECT <sentencia con parámetros>;

- Los parámetros formales indicados después del nombre del cursor tienen la siguiente sintaxis:

  <nombre_parámetro> [**IN**] <tipo_dato> [{ := | **DEPAULT** } <valor>]

  Son parámetros de entrada y su ámbito es local al cursor, por eso solamente pueden ser referenciados dentro de la consulta.

- La **apertura** del cursor pasándole parámetros se hará:

  OPEN nombre_cursor [(parámetro1, parámetro2, ...)];

12.4. Cursores con parámetros

- Debemos recordar que:
  - ☆ Los parámetros formales de un cursor son siempre IN y no devuelven ningún valor ni pueden afectar a los parámetros actuales.
  - ☆ La recogida de datos se hará, igual que en otros cursores explícitos, con FETCH.
  - ☆ La cláusula WHERE asociada al cursor se evalúa solamente en el momento de abrir el cursor. En ese momento es cuando se sustituyen las variables por su valor.
  - ☆ En el caso de los cursores FOR ... LOOP, puesto que la instrucción OPEN va implícita, el paso de parámetros se hará a continuación del identificador del cursor en la instrucción FOR ... LOOP, tal como se muestra:

```
FOR reg_emple IN c1 (20, 'DIRECTOR' ) LOOP
```