Administración de Bases de Datos

Operación y Mantenimiento

Excepciones en PL/SQL

Todos los errores que ocurren en los bloques de código, procedimientos o funciones, son tratados por PL/SQL como excepciones. Las excepciones pueden tener muy diferentes causas, por lo que no es posible anticipar todas ellas, pero si es posible escribir un código que permita manejar las excepciones y continuar la ejecución del programa de forma normal.

En la sección de manejo de excepciones de un bloque de código se pueden definir una o varias condiciones que controlen las excepciones.

La sintaxis básica de esta sección es:

```
BEGIN
    ...
EXCEPTION
    WHEN e1 THEN ...
    WHEN e2 THEN ...
    WHEN OTHERS THEN ...
END;
```

Cuando ocurre una excepción en la sección de código ejecutable, se detiene la ejecución del programa y el control pasa a la sección de manejo de excepciones.

Después de la ejecución del manejador de excepciones correspondiente, el control regresa la siguiente instrucción del bloque de código.

Ejemplo 1

```
SQL> DECLARE
  2     nom alumnos.nombre%TYPE;
  3     cla alumnos.clave%TYPE := '23120356';
  4  BEGIN
  5     SELECT nombre INTO nom FROM alumnos WHERE clave=cla;
  6     dbms_output.put_line('Nombre del alumno: ' || nom);
  7  END;
  8  /
DECLARE
*
ERROR en línea 1:
ORA-01403: No se ha encontrado ningún dato
ORA-06512: en línea 5
```

Si se ejecuta el bloque anterior para una *clave* de un alumno inexistente, se provoca un mensaje de error:

```
ORA-01403: No se ha encontrado ningún dato
```

Pero si se define la excepción pertinente, la respuesta es distinta:

```
SQL> DECLARE
  2     nom alumnos.nombre%TYPE;
  3     cla alumnos.clave%TYPE := '23120356';
  4  BEGIN
  5     SELECT nombre INTO nom FROM alumnos WHERE clave=cla;
  6     dbms_output.put_line('Nombre del alumno: ' || nom);
  7  EXCEPTION
  8     WHEN NO_DATA_FOUND THEN
  9        dbms_output.put_line('El alumno ' || cla ||  ' no existe');
 10  END;
 11  /
El alumno 23120356 no existe

Procedimiento PL/SQL terminado correctamente.
```

Ejemplo 2

Se modifica el bloque para que devuelva un número elevado de resultados:

```
SQL> DECLARE
  2     nom alumnos.nombre%TYPE;
  3     ciu alumnos.ciudad%TYPE := 'Morelia';
  4  BEGIN
  5     SELECT nombre INTO nom FROM alumnos WHERE ciudad=ciu;
  6     dbms_output.put_line('Nombre del alumno: ' || nom);
  7  EXCEPTION
  8     WHEN NO_DATA_FOUND THEN
  9        dbms_output.put_line('En la ciudad ' || ciu ||  ' no hay alumnos');
 10  END;
 11  /
DECLARE
*
ERROR en línea 1:
ORA-01422: la recuperación exacta devuelve un número mayor de filas que el
solicitado
ORA-06512: en línea 5
```

Es necesario agregar un nuevo caso en la sección de excepciones y entonces el resultado de la ejecución es más preciso:

```
SQL> DECLARE
  2     nom alumnos.nombre%TYPE;
  3     ciu alumnos.ciudad%TYPE := 'Morelia';
  4  BEGIN
  5     SELECT nombre INTO nom FROM alumnos WHERE ciudad=ciu;
  6     dbms_output.put_line('Nombre del alumno: ' || nom);
  7  EXCEPTION
  8     WHEN NO_DATA_FOUND THEN
  9        dbms_output.put_line('En la ciudad ' || ciu ||  ' no hay alumnos');
 10     WHEN TOO_MANY_ROWS THEN
 11        dbms_output.put_line('La consulta devuelve más de un alumno');
 12  END;
 13  /
La consulta devuelve más de un alumno

Procedimiento PL/SQL terminado correctamente.
```

Categoría de las excepciones

En PL/SQL hay tres categorías principales de excepciones:

- Excepciones internas, provocadas por errores en el entorno de Oracle, surgen

de manera automática, por ejemplo: ORA-27102 (out of memory). Estas excepciones no tienen un nombre, pero si cuentan con un código de error.

- Excepciones Predefinidas, que son las que ocurren durante la ejecución de un programa y tienen un nombre asignado como: NO_DATA_FOUND y TOO_MANY_ROWS.
- Excepciones definidas por el usuario, las cuales debe ser provocadas explícitamente y se les debe asignar un nombre.

Provocar Excepciones con RAISE

Para provocar una excepción explícitamente, se emplea la cláusula RAISE y se definen de la siguiente manera:

- El primer paso consiste en declarar la excepción en la sección de declaraciones.

```
DECLARE
   nombre_excepción EXCEPTION;
```

- Se le asigna un código de error a la excepción, instruyendo al compilador mediante la directiva PRAGMA:

```
PRAGMA EXCEPTION_INIT (nombre_excepción, código_error);
```

  Donde el código de error es un entero en el rango desde -20,999 a -20,000.

- En la sección de código se define una condición específica y se provoca la excepción mediante la sentencia RAISE.

```
RAISE nombre_excepción;
```

- En la sección de excepciones, se indica la forma en que se va a manejar.

```
WHEN nombre_excepción THEN
```

Ejemplo

```
SQL> DECLARE
  2      calif_invalida EXCEPTION;
  3      PRAGMA exception_init(calif_invalida, -20001 );
  4      cal expediente.calif%TYPE;
  5      cmat expediente.clavemat%TYPE;
  6  BEGIN
  7      SELECT clavemat, calif INTO cmat, cal FROM expediente WHERE ncontrol= '21120463';
  8      IF cal < 70 THEN
  9          RAISE calif_invalida;
 10      END IF;
 11  EXCEPTION
 12      WHEN calif_invalida then
 13         dbms_output.put_line('Calificación Inválida');
 14  END;
 15  /
Calificación Inválida

Procedimiento PL/SQL terminado correctamente.
```

Provocar Excepciones Internas

Si no se desea declarar una excepción, es posible utilizar la sentencia RAISE para provocar excepciones definidas internamente, invocando su nombre.

Ejemplo

```
SQL> DECLARE
  2      cal expediente.calif%TYPE;
  3      cmat expediente.clavemat%TYPE;
  4  BEGIN
  5      SELECT clavemat, calif INTO cmat, cal FROM expediente WHERE ncontrol= '21120463';
  6      IF cal < 70 THEN
  7          RAISE invalid_number;
  8      END IF;
  9  EXCEPTION
 10      WHEN invalid_number then
 11         dbms_output.put_line('Calificación Inválida');
 12  END;
 13  /
Calificación Inválida

Procedimiento PL/SQL terminado correctamente.
```

Manejo de Excepciones no previstas

Para manejar excepciones que no estén previstas, es conveniente agregar la condición WHEN OTHERS y dentro de ella se recomienda el empleo de las funciones SQLCODE y SQLERRM, para obtener información concreta del error ocurrido.

Ejemplo

```
SQL> DECLARE
  2      r_exp expediente%rowtype;
  3  BEGIN
  4      SELECT * INTO r_exp FROM expediente WHERE ncontrol= '21120466';
  5  EXCEPTION
  6      WHEN OTHERS THEN
  7          dbms_output.put_line('Código del Error:' || SQLCODE);
  8          dbms_output.put_line(SQLERRM);
  9  END;
 10  /
Código del Error:100
ORA-01403: No se ha encontrado ningún dato

Procedimiento PL/SQL terminado correctamente.
```

Cursores PL/SQL

Un cursor es un puntero que apunta al resultado de una consulta. Existen dos tipos de cursores: implícitos y explícitos.

Cursores Implícitos

Cada vez que se ejecuta una sentencia como SELECT INTO, INSERT, UPDATE o DELETE; Oracle crea automáticamente un cursor implícito.

Oracle maneja internamente toda la ejecución del ciclo del cursor implícito y al final puede informar de su estado mediante: SQL%ROWCOUNT, SQL%ISOPEN, SQL%FOUND y SQL%NOTFOUND.

Pero su operación puede causar excepciones tales como: NO_DATA_FOUND o TOO_MANY_ROWS.

Cursores Explícitos

Un cursor explícito se declara explícitamente mediante una asociación con una sentencia SELECT.

```
CURSOR nombre_cursor IS sentencia_Select;
```

Se tiene control sobre el manejo de estos cursores mediante los comandos: OPEN, FETCH y CLOSE.

Para poder usar un cursor, se debe abrir con la sentencia OPEN:

```
OPEN nombre_cursor;
```

Una vez abierto el cursor, se ejecuta la sentencia asociada y se posiciona en el primer registro.

El comando FETCH pasa el contenido del registro actual a las variables correspondientes:

```
FETCH nombre_cursor INTO lista_variables;
```

Esta sentencia se debe repetir para cada uno de los registros restantes.

Una vez concluido el proceso, se debe cerrar el cursor con la sentencia CLOSE para liberar la memoria:

```
CLOSE nombre_cursor;
```

Atributos de los cursores

Un cursor tiene cuatro atributos:

- **nombre_cursor%ISOPEN**. Su valor es TRUE cuando el cursor está abierto, FALSE en caso contrario.
- **nombre_cursor%FOUND**. Su valor es NULL antes del primer fetch, TRUE cuando un registro ha sido cargado exitosamente, FALSE si no se ha obtenido ningún registro y INVALID_CURSOR si el cursor no está abierto.
- **nombre_cursor%NOTFOUND.** Su valor es NULL antes del primer fetch, FALSE cuando un registro ha sido cargado exitosamente, TRUE si no se ha obtenido ningún registro y INVALID_CURSOR si el cursor no está abierto.
- **nombre_cursor%ROWCOUNT**. Su valor corresponde a la cantidad de registros que cargó el cursor. Si el cursor no está abierto devuelve INVALID_CURSOR.

Ejemplo

```
SQL> DECLARE
  2     CURSOR cur_exp IS
  3     SELECT ncontrol,nombre,apellido,AVG(calif) AS promedio FROM estudiantes INNER JOIN expediente
USING(ncontrol) GROUP BY ncontrol,nombre,apellido ORDER BY promedio DESC;
  4     reg_exp cur_exp%ROWTYPE;
  5  BEGIN
  6     DBMS_OUTPUT.PUT_LINE( 'N.Control  Nombre  Apellido  Promedio');
  7     OPEN cur_exp;
  8     LOOP
  9        FETCH cur_exp INTO reg_exp;
 10        EXIT WHEN cur_exp%NOTFOUND;
 11        DBMS_OUTPUT.PUT_LINE( reg_exp.ncontrol || '   ' || substr(reg_exp.nombre||'      ',1,8) ||
substr(reg_exp.apellido||'      ',1,10) || reg_exp.promedio);
 12     END LOOP;
 13     CLOSE cur_exp;
 14  END;
 15  /
N.Control  Nombre  Apellido  Promedio
21120321   Juan    Avilés    90
21120179   Ana     Torres    80
21120463   Pedro   Vela      67

Procedimiento PL/SQL terminado correctamente.
```

Cursores FOR LOOP

Los cursores FOR LOOP ejecutan un bloque de instrucciones para cada uno de los registros devueltos en la consulta asociada al cursor.

Esto permite acceder a cada registro sin necesidad de manejar las operaciones sobre el cursor. El cursor FOR LOOP crea implícitamente un índice %ROWTYPE sobre los registros, al iniciar abre el cursor, recorre todos los registros y al terminar cierra al cursor automáticamente.

Su sintaxis es:

```
FOR registro IN nombre_cursor
LOOP
    sentencias;
END LOOP;
```

Ejemplo

```
SQL> DECLARE
  2    CURSOR cur_cli IS
  3    SELECT "Empresa", "Nombre" FROM "Cliente" WHERE "Sucursal"=108 ORDER BY 1;
  4  BEGIN
  5    FOR reg_cli IN cur_cli
  6    LOOP
  7      dbms_output.put_line( reg_cli."Empresa" || ' - ' ||  reg_cli."Nombre");
  8    END LOOP;
  9  END;
 10  /
Comercial Citlalli - Jessica Fox
Lácteos Artificiales - María Rojo
Lácteos del Centro - María Conesa
Vinos Importados - Julio Jaramillo

Procedimiento PL/SQL terminado correctamente.
```

También se puede reemplazar el cursor directamente por una sentencia SELECT.

La Sintaxis sería así:

```
FOR registro IN (sentencia_select)
LOOP
    sentencias;
END LOOP;
```

Ejemplo

```
SQL> BEGIN
  2    FOR reg_cli IN (SELECT "Empresa", "Nombre" FROM "Cliente" WHERE "Sucursal"=108 ORDER BY 1)
  3    LOOP
  4      dbms_output.put_line( reg_cli."Empresa" || ' - ' ||  reg_cli."Nombre");
  5    END LOOP;
  6  END;
  7  /
Comercial Citlalli - Jessica Fox
Lácteos Artificiales - María Rojo
Lácteos del Centro - María Conesa
Vinos Importados - Julio Jaramillo

Procedimiento PL/SQL terminado correctamente.
```

Cursores con Parámetros

Un cursor explícito puede aceptar una lista de parámetros, cada vez que se abra el cursor se le pueden pasar diferentes argumentos para obtener distintos resultados.

La sintaxis de la declaración del cursor con parámetros es:

```
CURSOR nombre_cursor (lista_parametros) IS Sentencia_select;
```

Al momento de abrir un cursor con parámetros, se emplea la siguiente sintaxis:

```
OPEN nombre_cursor (lista_valores)
```

Ejemplo

```
SQL> DECLARE
  2     CURSOR cur_emp (minS NUMBER, maxS NUMBER) IS
  3     SELECT * FROM "Empleado" WHERE "Sueldo" BETWEEN minS AND maxS;
  4     reg_emp "Empleado"%rowtype;
  5  BEGIN
  6      dbms_output.put_line('*** Sueldo Bajo: ');
  7      OPEN cur_emp(6000,14000);
  8      LOOP
  9          FETCH cur_emp INTO reg_emp;
 10          EXIT WHEN cur_emp%notfound;
 11          dbms_output.put_line(reg_emp."Nombre" || ' ' || reg_emp."Apellido");
 12      END LOOP;
 13      CLOSE cur_emp;
 14      dbms_output.put_line('*** Sueldo Alto: ');
 15      OPEN cur_emp(19000,26000);
 16      LOOP
 17          FETCH cur_emp INTO reg_emp;
 18          EXIT WHEN cur_emp%notfound;
 19          dbms_output.put_line(reg_emp."Nombre" || ' ' || reg_emp."Apellido");
 20      END LOOP;
 21      CLOSE cur_emp;
 22  END;
 23  /
*** Sueldo Bajo:
Luisa Lara
Juana Mena
Daniel Contreras
Miguel Ugalde
*** Sueldo Alto:
Laura Huerta
Juana Linares

Procedimiento PL/SQL terminado correctamente.
```

Cursores con Parámetros con valores por omisión

Un cursor parametrizado puede tener valores por omisión en sus parámetros:

```
CURSOR nombre_cursor (parámetro tipo := valor_omisión, ...) IS
Sentencia_select;
```

Esos valores por omisión se toman cuando se abre el cursor sin pasarle ningún argumento para esos parámetros.

Ejemplo

```
SQL> DECLARE
  2     CURSOR cur_emp (minS NUMBER :=14000, maxS NUMBER := 19000) IS
  3     SELECT * FROM "Empleado" WHERE "Sueldo" BETWEEN minS AND maxS;
  4     reg_emp "Empleado"%rowtype;
  5  BEGIN
  6      dbms_output.put_line('*** Sueldo Medio: ');
  7      OPEN cur_emp;
  8      LOOP
  9          FETCH cur_emp INTO reg_emp;
 10          EXIT WHEN cur_emp%notfound;
 11          dbms_output.put_line(reg_emp."Nombre" || ' ' || reg_emp."Apellido");
 12      END LOOP;
 13      CLOSE cur_emp;
 14  END;
 15  /
*** Sueldo Medio:
Maico Rico
Beatriz Álvarez
Luis Navarro

Procedimiento PL/SQL terminado correctamente.
```

Variables de Cursor con REF CURSOR

Una variable de cursor es una variable que hace referencia a un cursor. Su principal característica es que no está ligada a ninguna consulta SELECT, por lo que puede ser abierto para cualquier sentencia.

Esto permite pasar el resultado de una consulta entre diferentes programas, simplemente pasando la referencia a ese cursor. Para lo cual se emplea el tipo REF CURSOR para declarar la variable y se hace de tres formas:

- Un REF CURSOR fuerte es cuando la variable está asociada con un tipo específico de registro. Por ejemplo:

```
DECLARE
    TYPE tipo_cliente IS REF CURSOR RETURN cliente%ROWTYPE;
    cur_cliente tipo_cliente;
```

- Un REF CURSOR débil no está asociado con ninguna estructura específica:

```
DECLARE
    TYPE tipo_cliente IS REF CURSOR;
    cur_cliente tipo_cliente;
```

- Existe un REF CURSOR por omisión llamado SYS_REFCURSOR, que se declara

de la siguiente manera:

```
DECLARE
    cur_cliente SYS_REFCURSOR;
```

CURSOR FOR UPDATE

En ocasiones se desea bloquear un conjunto de registros antes de ejecutar una actualización dentro de un programa. Oracle ofrece la cláusula FOR UPDATE dentro de la sentencia SELECT, para declarar un cursor que actualice registros mediante un mecanismo de bloqueo. La sintaxis requerida es:

```
CURSOR nombre_cursor (lista_parametros) IS
    SELECT columnas FROM tabla WHERE condición FOR UPDATE;
```

Los registros seleccionados permanecerán bloqueados hasta que la transacción se complete con COMMIT o ROLLBACK, o se cierre el cursor, por lo que es importante definir la condición WHERE solamente con los registros necesarios.

Si lo que se desea es actualizar una columna en particular, se puede declarar el cursor de la siguiente manera:

```
CURSOR nombre_cursor (lista_parametros) IS
    SELECT columnas FROM tabla WHERE condición FOR UPDATE OF
    Nombre_columna;
```

Ejemplo

```
SQL> DECLARE
  2      CURSOR cur_clientes IS SELECT "Clave", "Nombre", Credito
  3          FROM "Cliente" WHERE Credito > 0 FOR UPDATE OF Credito;
  4      cre "Cliente".Credito%TYPE;
  5      inc NUMBER DEFAULT 0;
  6  BEGIN
  7      FOR reg_clientes IN cur_clientes
  8      LOOP
  9      cre := reg_clientes.Credito;
 10          IF cre >= 50000 THEN inc := 0.05;
 11              ELSIF cre < 50000 AND cre >= 20000 THEN inc := 0.1;
 12              ELSE inc := 0.15;
 13          END IF;
 14          UPDATE "Cliente" SET credito = credito * (1 + inc)
 15              WHERE "Clave" = reg_clientes."Clave";
 16          dbms_output.put_line('Nombre del cliente: '
 17              || reg_clientes."Nombre" || ' Crédito: '
 18              || reg_clientes.Credito );
 19      END LOOP;
 20  EXCEPTION
 21      WHEN OTHERS THEN
 22          dbms_output.put_line('Código de Error:' || SQLCODE);
 23          dbms_output.put_line('Mensaje:' || sqlerrm);
 24      RAISE;
 25  END;
 26  /
Nombre del cliente: Gilberto Huerta Crédito: 30000
Nombre del cliente: María Rojo Crédito: 15000
Nombre del cliente: Jessica Fox Crédito: 5000
Nombre del cliente: María Conesa Crédito: 40000
Nombre del cliente: Julio Jaramillo Crédito: 100000

Procedimiento PL/SQL terminado correctamente.
```