PUCP

1INF33 Base de Datos

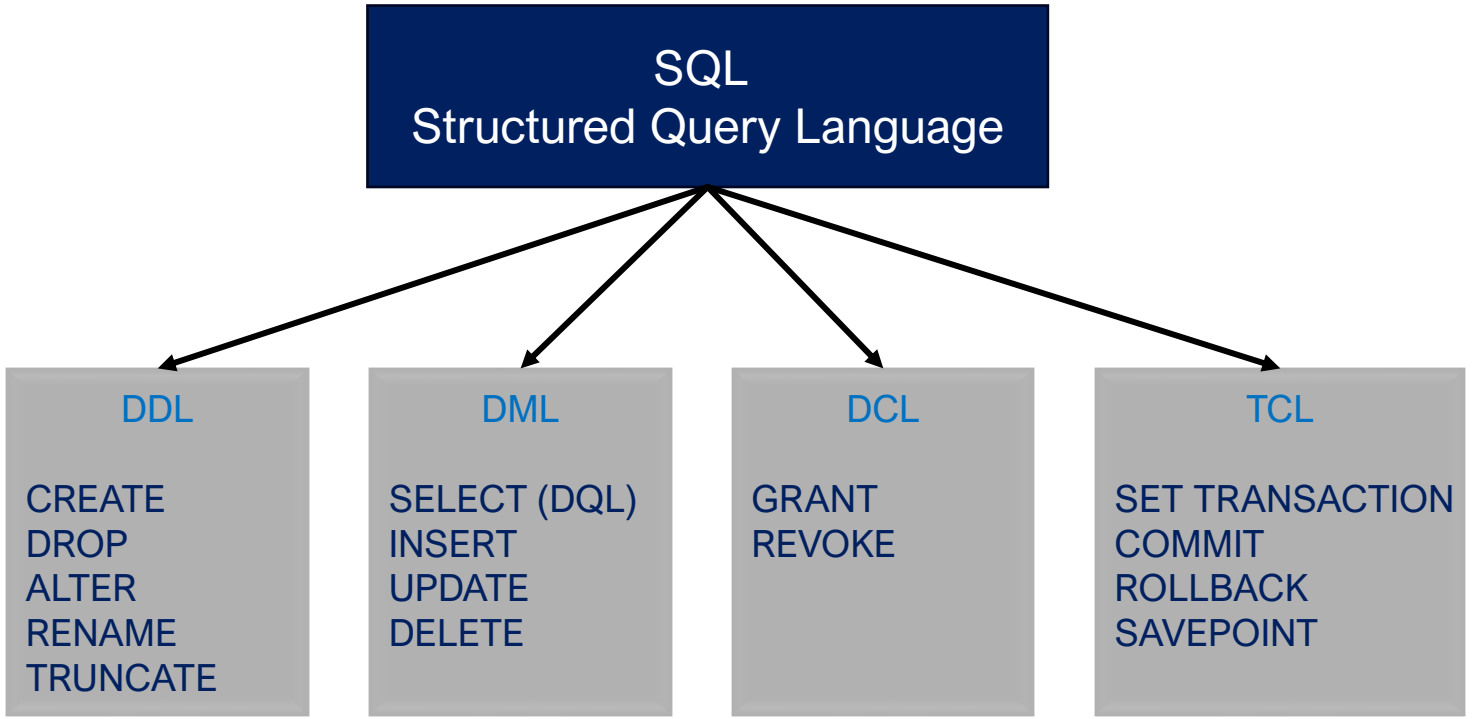
PL/SQL: Subprogramas y cursores

2024

Profesores del curso

ÍNDICE

Antecedentes



Procedure

Procedimiento

El procedimiento es un subprograma que ejecuta una acción específica y tiene tres partes:

- La especificación
- El cuerpo
- El manejo de excepciones

La especificación se inicia con la cláusula PROCEDURE y termina con el nombre del procedimiento o con una lista de parámetros. La declaración de parámetros es opcional.

El cuerpo del procedimiento sigue la misma estructura de un bloque PL/SQL.

El manejo de excepciones en una parte opcional, que se ocupa de las condiciones de error en tiempo de ejecución.

Procedimiento

Se puede escribir un procedimiento utilizando la siguiente sintaxis:

```
[CREATE [OR REPLACE]]
PROCEDURE procedure_name[(parameter[, parameter]...)]
  [AUTHID {DEFINER | CURRENT_USER}] {IS | AS}
  [PRAGMA AUTONOMOUS_TRANSACTION;]
  [local declarations]
BEGIN
  executable statements
[EXCEPTION
  exception handlers]
END [name];
```

Procedimiento

```
CREATE OR REPLACE PROCEDURE raise_salary(emp_id INTEGER, increase REAL) IS
/* se puede observar que, a diferencia de las variables, un parámetro no requiere
  especificar su magnitud, sólo su tipo */
current_salary NUMBER;
salary_missing EXCEPTION;
BEGIN
  SELECT sal INTO current_salary FROM empleado
  WHERE empno = emp_id;
  IF current_salary IS NULL THEN
      RAISE salary_missing;
  ELSE
      UPDATE empleado SET sal = sal + increase
      WHERE emp = emp_id;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
      INSERT INTO emp_audit VALUES (emp_id, 'Número no encontrado')
  WHEN salary_missing THEN
      INSERT INTO emp_audit  VALUES (emp_id, 'Salario es NULO')
END raise_salary;
```

Procedimiento

Un procedimiento es invocado como un bloque PL/SQL.

Por ejemplo, el procedimiento raise_salary es invocado de la siguiente manera:

```
raise_salary(1348, 550);
```

Procedimiento

```
CREATE OR REPLACE PROCEDURE award_bonus(emp_id NUMBER) IS
  bonus           REAL;
  comm_miss       EXCEPTION;
BEGIN
  SELECT comm*0.25 INTO bonus FROM employees
  WHERE empno = emp_id;
  IF bonus IS NULL
           RAISE comm_miss;
  ELSE
           UPDATE payroll SET ...;
  END IF;
EXCEPTION
  WHEN  comm_miss THEN
     ...
END;
```

Subprograma

Subprograma: uso de parámetros

Los subprogramas pasan información a través de parámetros. Las variables o expresiones referenciadas en la invocación de los subprogramas son llamadas los parámetros actuales.

Las variables declaradas en el subprograma como parte de su especificación son llamadas parámetros formales.

Modos

Se utilizan tres modos para definir el comportamiento de un parámetro: IN, OUT e IN OUT.

Subprograma: uso de parámetros

IN	OUT	IN OUT
Por defecto	Debe ser especificado	Debe ser especificado
Pasa el valor al subprograma	Devuelve el valor al subprograma	Pasa el valor inicial al subprograma; retorna el valor modificado
El parámetro formal actúa como constante	El parámetro formal actúa como una variable NO inicializada	El parámetro formal actúa como una variable inicializada
Al parámetro formal no se le puede asignar valores	El parámetro formal no puede ser utilizado en expresiones; debe asignársele un valor	Al parámetro formal puede asignársele un valor
El parámetro actual puede ser una constante, una variable inicializada, un literal o una expresión	El parámetro actual debe ser una variable	El parámetro actual debe ser una variable

Subprograma: uso de parámetros

Se muestra cómo es posible inicializar parámetros tipo IN :

```
CREATE OR REPLACE PROCEDURE create_dept
    (new_name              CHAR DEFAULT 'TEMP',
     new_localidad         CHAR DEFAULT 'TEMP') IS
BEGIN
    INSERT INTO dept
           VALUES (deptno_seq.NEXTVAL, new_dname, new_localidad);
END create_dept;
```

Subprograma: almacenamiento

Los subprogramas pueden ser almacenados permanentemente en la Base de datos y estar inmediatamente disponibles para su ejecución.

Para el almacenamiento de subprogramas se puede utilizar la sentencia CREATE PROCEDURE o CREATE FUNCTION.

```
CREATE PROCEDURE debit_account (acc_id  INTEGER, amount REAL) IS
    old_balance     REAL;
    new_balance     REAL;
    overdrawn       EXCEPTION;
BEGIN
    SELECT
    . . .
EXCEPTION
    WHEN overdraw THEN
    . . .
END debit_account;
```

Subprograma: almacenamiento

Para crear un procedimiento o reemplazarlo si existe se tiene las sentencias CREATE OR REPLACE seguido del PROCEDURE o FUNCTION según sea el objeto por crear o reemplazar.

```
CREATE  OR REPLACE PROCEDURE debit_account  . . . ;

CREATE OR REPLACE FUNCTION . . .;
```

Para eliminar un subprograma:

```
DROP PROCEDURE debit_account;

DROP FUNCTION debit_account;
```

Subprograma: invocación

- De otro subprograma

```
create_dept( name, location) ;
```

- De una aplicación: Precompilados en C y COBOL

```
EXEC SQL EXECUTE
BEGIN
        create_dept(:name, :location)
END
END-EXEC
```

- SQLWindows

```
Call SqlPLSQLCommand(hSql,'create_dept(name, location)')
```

- De un tool ORACLE (SQL *Plus, SQL*Forms and SQL*DBA)

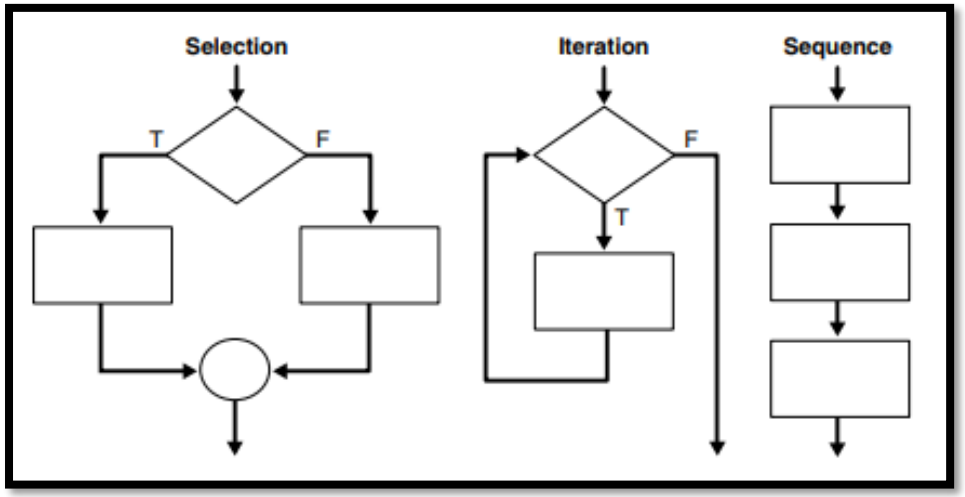
```
SQL> EXECUTE create_dept('MARKETING','NEW YORK ');
SQL> BEGIN create_dept('MARKETING','NEW YORK '); END;
```

Estructuras de control

Estructuras de Control de flujo

Las estructuras de control son la más importante extensión del PL/SQL sobre el SQL.

- Selectivas
  - IF-THEN-ELSE
  - CASE-WHEN
- Iterativas o bucles
  - LOOP
  - FOR
  - WHILE
  - EXIT WHEN
- Secuenciales
  - GOTO



Estructuras de Control de flujo

A continuación, se muestra cómo se pueden utilizar:

- Selectiva simple

```
IF <condicion> THEN
...<sentencias>
END IF;
```

- Selectiva doble

```
IF <condicion> THEN
...<sentencias>
ELSE
...<sentencias>
END IF;
```

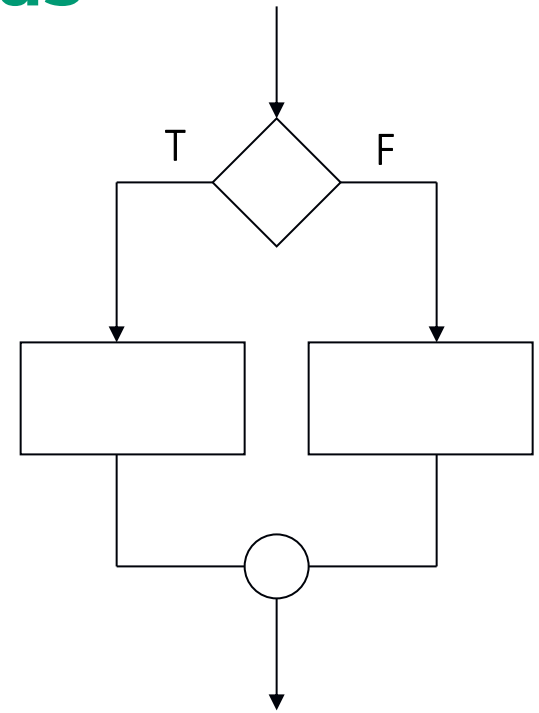
- Selectiva múltiple

```
IF <condicion> THEN
...<sentencias>
ELSIF <condicion> THEN
...<sentencias>
ELSIF <condicion> THEN
...<sentencias>
ELSE
...<sentencias>
END IF;
```

Estructuras de Control de flujo: Selectivas

- IF-THEN-ELSE
  Ejecuta una secuencia de sentencias de una manera condicional.

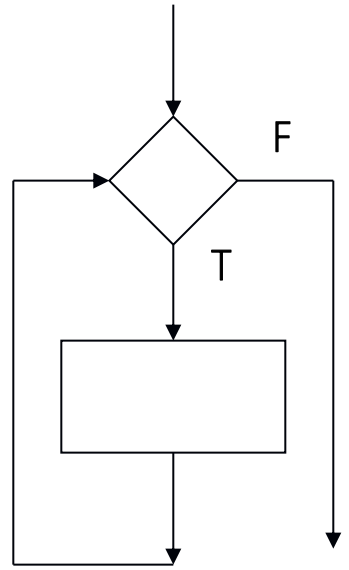
```
IF condición THEN
    secuencia_de_sentencias1
ELSE
    secuencia_de_sentencias2
END IF ;
```




```
IF condición THEN
    secuencia_de_sentencias1
ELSIF
    secuencia_de_sentencias2
ELSE
    secuencia_de_sentencias3
END IF ;
```

Estructuras de Control de flujo: iterativas

LOOP, ejecuta una secuencia de sentencias de una manera incondicional hasta que se fuerce a salir con EXIT o con EXIT WHEN.



- WHILE-LOOP, asocia una condición a la secuencia de sentencias entre las palabras reservadas LOOP y END LOOP.

- FOR-LOOP, permite iterar una secuencia de sentencias en función del valor de un entero.

Tipos de datos

Tipos de datos

Las variables y constantes de PL/SQL poseen operadores, los cuales permiten referenciar el tipo de dato y estructura de un objeto sin repetir su definición.

```
DECLARE
    rating  tabla.columna%type;
    activo cliente.estado%type;

BEGIN
```

Tenemos:

- %TYPE
- %ROWTYPE

Tipos de datos

- **%TYPE** provee el tipo de dato de una variable, constante o columna de la base de datos.

```
DECLARE
    Costo factura.monto%TYPE;
```

Ventajas:

1. No se necesita saber el tipo de dato
2. Si se altera la definición en la BD de la columna, el tipo de dato cambia en tiempo de ejecución

Tipos de datos

- **%ROWTYPE** representa el tipo de registro de una fila de una tabla. El registro puede almacenar una fila completa de datos seleccionados desde la tabla o recuperados (*fetched*) por un cursor.

```
DECLARE
    item_rec        item%ROWTYPE;
……

    nro_item := item_rec.item;
    cantidad_items := ítem_rec.cantidad;
    precio_item := item_rec.precio;
```

Tipos de datos

%ROWTYPE es un atributo que permite asignar un tipo de dato registro, que representa una fila (o parte de ella) de una tabla o vista de la base de datos.

La variable de este tipo (registro) puede contener parte o toda una fila recuperada con un SELECT (a tablas o vistas) o FETCH (a un cursor).

Las columnas de la fila tienen los mismos nombres y tipos de datos que los correspondientes campos del registro.

Tipos de datos

Una declaración usando **%ROWTYPE** no permite inicialización; sin embargo, puede asignarse valores a todos los campos mediante:

- Asignación de otra variable de tipo registro y que sea del mismo tipo de recuperación (SELECT o CURSOR).
- Usando INTO dentro de un SELECT

Importante: sólo sirve para recuperación. No se contempla su uso en INSERT

Tipos de datos y abstracción de datos(cursor)

- Para cursores:

```
DECLARE
CURSOR c1 IS SELECT item, precio, cantidad FROM producto;     -- abstracción
    item_rec   c1%ROWTYPE;                                     -- tipo de dato
    ……
BEGIN
    OPEN C1;
    ……
    LOOP
            FETCH c1 INTO item_rec;
            EXIT WHEN c1%NOTFOUND;
    -- procesa los datos leídos
    END LOOP;
END;
```

Alcance y visibilidad

- En bloques de código

```
DECLARE
        V OtraVariable%TYPE;
BEGIN
        : : :
        : : :
        DECLARE
                V OtraVariable%TYPE
        BEGIN
                : : :
                : : :
        END;
        : : :
        : : :
END;
```

Package

Paquete

Un paquete (*package*) es un objeto ORACLE que permite agrupar de manera lógica tipos, objetos y subprogramas PL/SQL que están relacionados entre sí.

Un paquete está formado de dos partes: la especificación (*especification*) y el cuerpo (*body*). Donde tenemos que:

- La especificación es la interface con las aplicaciones, donde se declaran los objetos disponibles para su uso.
- El cuerpo es la implementación de la especificación.

A diferencia de los subprogramas, un paquete no puede ser invocado.

Abstracción de datos

Definición

La abstracción de datos nos permite trabajar con las propiedades esenciales de los datos sin involucrarse demasiado en los detalles.

Después de diseñar una estructura de datos, puede concentrarse en diseñar algoritmos que manipulen la estructura de dato.

[En Oracle] se pueden usar:

- Cursores*
- Colecciones
- Registros
- Objetos

*sólo desarrollaremos este tipo de dato (estructura)

Cursores

Un cursor es un puntero a un área SQL privada que almacena información sobre el procesamiento de una sentencia SELECT o de un INSERT, UPDATE o DELETE. Un cursor PL/SQL permite referenciar una de estas áreas de tal manera que se pueda acceder a la información almacenada en ella.

Existen dos clases de cursores:

- Implícitos, son declarados para todas las instrucciones de manipulación de datos (DML), además de aquellas consultas que retornan una sola fila. Estos cursores no se pueden nombrar y, por lo tanto, no se pueden controlar ni hacer referencia a ellos desde otro lugar del código.

- Explícitos, son necesarios para aquellas consultas que retornan más de una fila.

Pasos para crear un cursor explícito

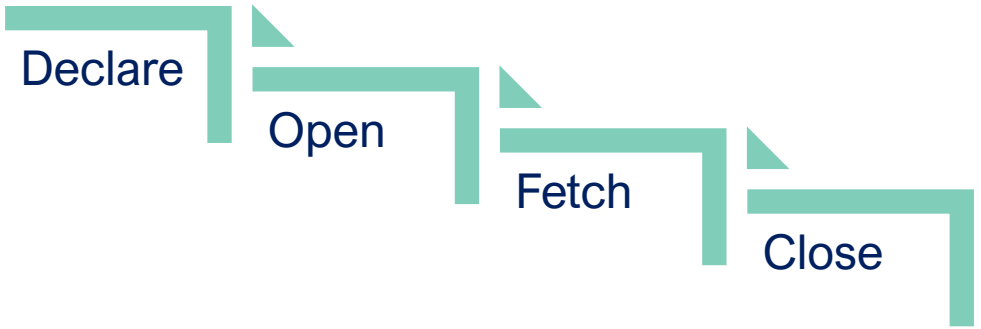
Puede declarar explícitamente un cursor y luego realizar las operaciones de apertura, recuperación y cierre usted mismo.

CURSOR **cursor_name** IS **select_statement;**

OPEN **cursor_name;**

FETCH **cursor_name** INTO PL/SQL **variable;**
o
FETCH **cursor_name** INTO PL/SQL **record;**

CLOSE **cursor_name;**



Cursor explícito

Cursor explícito

Son necesarios para aquellas consultas que retornan más de una fila.

```
DECLARE
CURSOR c1 IS   SELECT ename, deptno FROM emp  WHERE sal > 2000;

CURSOR c2 (mediana IN NUMBER) IS  SELECT job, ename FROM emp WHERE sal > mediana;
mi_registro   c1%ROWTYPE;

BEGIN
OPEN c1;
LOOP
        FETCH c1 INTO mi_registro;
        EXIT WHEN c1%NOTFOUND;
        ---
        --- Procesa los datos leídos
        ---
END LOOP;
```

Definición de parámetros

Puede crear un cursor explícito que tenga parámetros formales y luego pasar diferentes parámetros reales al cursor cada vez que se apertura.

```
DECLARE
CURSOR c1 (mediana IN NUMBER) IS
    SELECT job, ename FROM emp
    WHERE emp.sal > mediana;

-- Cursor con parámetros con valores por defecto
CURSOR c2 (inicio  INTEGER  DEFAULT 0, fin INTEGER DEFAULT 99)
                        IS SELECT . . .
```

Abriendo el cursor (OPEN)

Al abrir el cursor se ejecuta la consulta y se identifica el conjunto de resultados, que consta de todas las filas que cumplen los criterios de búsqueda de la consulta.

Ejemplos:

```
OPEN c1;  -- cursor c1, sin parámetros

OPEN c1('ATTLEY', 300);  -- forma 1 , con parámetros

OPEN c1(employee.name, 150);  -- forma 2, con parámetros

OPEN c2(10);  -- un solo parámetro
```

Barriendo las filas (FETCH)

Recupera las filas una a la vez del conjunto resultado de una consulta de varias filas. Cada búsqueda recupera la fila actual y hace avanzar el cursor a la siguiente fila del conjunto de resultados.

```
OPEN c1;  -- abrir el cursor
    LOOP
            FETCH c1 INTO my_record;
            EXIT WHEN c1%NOTFOUND;
            -- Procesa los datos leídos
    END LOOP;
CLOSE c1;
```

Barriendo las filas (FETCH)

Puede almacenar cada columna en una variable independiente (%TYPE) o almacenar la fila completa en un registro que tenga los campos apropiados, que normalmente se declaran mediante %ROWTYPE.

```
DECLARE
        my_sal    emp.sal%TYPE;
        my_job    emp.job%TYPE;
        factor    INTEGER : = 2;
        CURSOR c1 IS SELECT factor*sal FROM emp WHERE job = my_job;
BEGIN
        . . .
        OPEN c1;  --  aquí factor es igual a 2
        LOOP
                FETCH c1 INTO my_sal;
                EXIT WHEN c1%NOTFOUND;
                . . .
                factor := factor + 1;        -- no afecta el FETCH
        END LOOP;
END;
```

Cerrando el cursor (CLOSE)

La instrucción CLOSE desactiva el cursor y el conjunto de resultados se vuelve indefinido.

```
CLOSE c1;
```

Si se cierra el cursor, se puede volver a abrir, y puede ejecutar la consulta nuevamente con los últimos valores de los parámetros y variables del cursor a los que se hace referencia en la cláusula WHERE. Cualquier otra operación en un cursor cerrado genera la excepción predefinida INVALID_CURSOR.

Control del cursor

La sentencia OPEN ejecuta la consulta asociada con el cursor, identifica el área activa (active set) y coloca el cursor antes de la primera fila. La sentencia FETCH recupera la fila activa y avanza el cursor a la siguiente fila.

Cuando la última fila ha sido procesada, la sentencia CLOSE deshabilita al cursor.

Cursor explícito

Operadores

Operadores

Tanto el cursor implícito como el cursor explícito tienen ciertos atributos a los que se puede acceder. Estos atributos brindan más información sobre las operaciones del cursor.

A continuación, se muestran los diferentes atributos del cursor:

- %NOTFOUND
- %FOUND
- %ISOPEN
- %ROWCOUNT

%NOTFOUND

Si el último FETCH realizado no lee filas (porque el active set está vacío), %NOTFOUND evalúa a TRUE.

```
LOOP

        FETCH c1 INTO my_name, my_deptno;
        EXIT WHEN c1%NOTFOUND;
        . . . -- procesa los datos leídos

END LOOP;
```

%FOUND

Representa el opuesto lógico a %NOTFOUND.

Antes del primer FETCH y después del OPEN retorna NULL. Luego, retorna TRUE si el último FETCH encontró filas o FALSE en caso contrario.

```
LOOP
        FETCH c1 INTO my_name, my_deptno;
        IF c1%FOUND THEN
                INSERT INTO . . .
        ELSE
                EXIT;
        END IF;
END LOOP;
```

%ROWCOUNT

Retorna el número de FETCH realizados sobre un cursor.

Antes del primer FETCH y luego del OPEN %ROWCOUNT retorna cero.

```
LOOP
        FETCH c1 INTO my_name, my_deptno;
        IF c1%ROWCOUNT > 10  THEN
            . . .
        END IF;
        . . .
END LOOP;
```

%ISOPEN

Evalúa a TRUE si el cursor está abierto, FALSE en caso contrario.

```
IF c1%ISOPEN THEN          -- el cursor está abierto
                . . .
ELSE
                OPEN c1;
END IF;
```

Instrucción FOR LOOP

Uso

Control de cursores

Se puede simplificar el manejo de cursores explícitos mediante el uso del cursor con FOR LOOP en lugar de OPEN-FETCH-CLOSE.

```
DECLARE
        CURSOR c_item IS SELECT linea, precio, cantidad FROM pedido;
        ……
BEGIN
        ……
        FOR r_item IN c_item LOOP
                …….
                monto := monto + r_item.precio;
        END LOOP;
END;
```

Importante:

FOR LOOP implícitamente declara **r_item** como c_item%ROWTYPE

FOR LOOP para cursores

Usar FOR LOOP simplifica el código para el uso de cursores. FOR LOOP automáticamente abre el cursor, *fetchea* repetidamente las filas del active set y luego cierra el cursor.

```
DECLARE
        result   temp.col1%TYPE;
        CURSOR   c1 IS SELECT n1, n2, n3 FROM  data_table WHERE exper_num = 1;
BEGIN
        FOR r_c1 IN c1 LOOP
                result := r_c1.n2 / ( r_c1.n1 + r_c1.n3);
                INSERT INTO temp VALUES (result, NULL, NULL);
        END LOOP;
        COMMIT;
END;
```

Además, saber que r_c1 sólo tiene significado dentro del LOOP, por tanto, cualquier referencia fuera de él es ilegal. Otro aspecto importante es el hecho de que el cursor se cierra automáticamente al concluir el LOOP.

Pasando parámetros en FOR LOOP

El paso de parámetros en un cursor FOR LOOP, se realiza como en el siguiente ejemplo:

```
DECLARE
        resultado temp.col1%TYPE;
        my_num     NUMBER := 10;
        CURSOR    c1 (num NUMBER) IS
                SELECT n1, n2, n3 FROM  data_table WHERE exper_num =  num;
BEGIN
        FOR rec_c1 IN c1(my_num)  LOOP
                resultado := rec_c1.n2 / ( rec_c1.n1 + rec_c1.n3);
                INSERT INTO temp VALUES (resultado, NULL, NULL);
        END LOOP;
        COMMIT;
END;
```

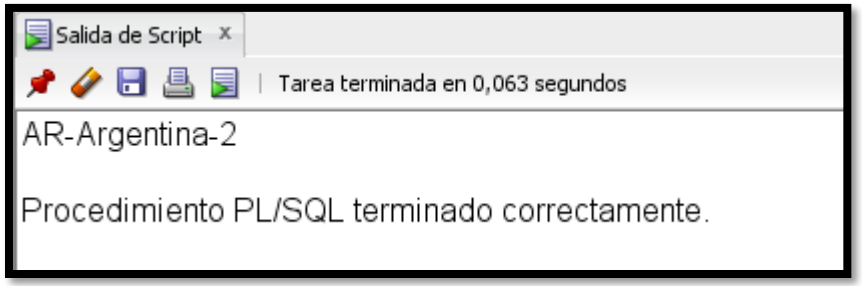
Ejemplo

```
DECLARE
CURSOR cpaises IS
SELECT COUNTRY_ID, COUNTRY_NAME, REGION_ID  FROM COUNTRIES;

co_pais       COUNTRIES.COUNTRY_ID%type;
descripcion   COUNTRIES.COUNTRY_NAME%type;
region        COUNTRIES.REGION_ID%type;

BEGIN
       OPEN cpaises;
       FETCH cpaises INTO co_pais, descripcion, region;
       dbms_output.put_line(co_pais||'-'||DESCRIPCION||'-'||region );
       CLOSE cpaises;
END;
```

Resultado: imprime un solo registro.



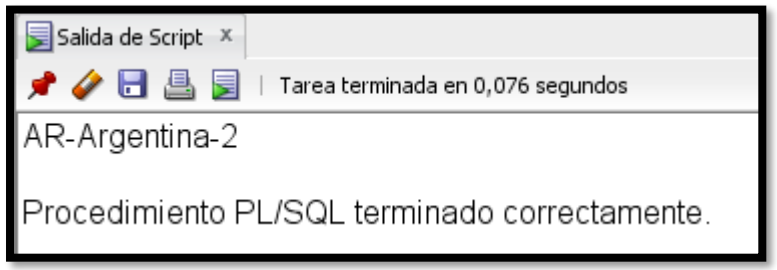
Ejemplo

```
DECLARE
CURSOR cpaises IS
       SELECT COUNTRY_ID, COUNTRY_NAME, REGION_ID FROM COUNTRIES;

registro      cpaises%ROWTYPE;    --uso de registro

BEGIN
       OPEN cpaises;
       FETCH cpaises INTO registro;
       dbms_output.put_line(registro.country_id||''||registro.country_name||'-
'||registro.region_id);
       CLOSE cpaises;
END;
```

Resultado: imprime un solo registro.



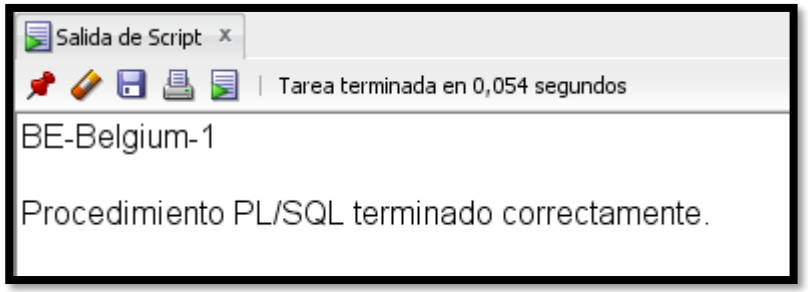
Ejemplo

```
DECLARE
CURSOR cpaises (p_region countries.region_id%type)
IS
SELECT COUNTRY_ID, COUNTRY_NAME, REGION_ID  FROM COUNTRIES
WHERE REGION_ID = p_region;

registro      cpaises%ROWTYPE; --uso de registro

BEGIN
   OPEN cpaises (1);          -- inicializando cursor
   FETCH cpaises INTO registro;
       dbms_output.put_line(registro.country_id||'-'||registro.country_name||'-
       '||registro.region_id);
   CLOSE cpaises;
END;
```

Resultado: imprime un solo registro.



Ejemplo

```
DECLARE
CURSOR cpaises
IS SELECT COUNTRY_ID, COUNTRY_NAME, REGION_ID  FROM COUNTRIES;
co_pais              COUNTRIES.COUNTRY_ID%type;
descripcion          COUNTRIES.COUNTRY_NAME%type;
region               COUNTRIES.REGION_ID%type;
BEGIN
OPEN cpaises;
     LOOP
        FETCH cpaises INTO co_pais, descripcion, region;
        EXIT WHEN cpaises%NOTFOUND;
        dbms_output.put_line(co_pais||'-'||DESCRIPCION||'-'||region );
      END LOOP;
CLOSE cpaises;
END;
```

Ejemplo

```
DECLARE
CURSOR  cpaises  IS  SELECT  COUNTRY_ID,  COUNTRY_NAME,  REGION_ID    FROM
COUNTRIES;
co_pais              COUNTRIES.COUNTRY_ID%type;
descripcion          COUNTRIES.COUNTRY_NAME%type;
region               COUNTRIES.REGION_ID%type;

BEGIN
OPEN cpaises;
FETCH cpaises INTO co_pais, descripcion, region;
      WHILE  cpaises%FOUND   LOOP
          dbms_output.put_line(co_pais||'-'||DESCRIPCION||'-'||region
);
          FETCH cpaises INTO co_pais, descripcion, region;
      END LOOP;
CLOSE cpaises;
END;
```

Ejemplo

```
DECLARE
CURSOR cpaises
IS
       SELECT COUNTRY_ID, COUNTRY_NAME, REGION_ID FROM COUNTRIES;

BEGIN
      FOR registro IN cpaises LOOP
       dbms_output.put_line(registro.country_id||'-'||registro.country_name||'-
'||registro.region_id);
      END LOOP;
END;
```

Recomendaciones

A continuación, se ofrecen algunas directrices que le ayudarán a decidir qué técnica utilizar:

- Cuando obtenga una única fila, utilice SELECT-INTO. No utilice un cursor explícito ni un bucle FOR de cursor.
- Cuando obtenga todas las filas de una consulta, utilice un bucle FOR de cursor a menos que el cuerpo del bucle ejecute una o más sentencias DML (INSERT, UPDATE o DELETE).
- Utilice un cursor explícito cuando obtenga varias filas, pero pueda salir condicionalmente antes de obtener todas las filas.

Ejemplo: Subprogramas

```
CREATE OR REPLACE PROCEDURE calcularComision(valor in number, salida out number)
IS
suma EMP.comm%type;

Cursor C1 (pcomision in NUMBER) IS
                    SELECT * FROM emp where comm > pcomision;

BEGIN
    suma := 0;
    FOR mi_registro IN C1(valor) LOOP
       dbms_output.put_line(mi_registro.empno||'-'||mi_registro.ename||'-
       '||mi_registro.comm);
       suma := suma + mi_registro.comm;
    END LOOP;
    salida:= suma;
END;
```

Ejemplo: Subprogramas

```
SET SERVEROUTPUT ON;
DECLARE
monto       EMP.COMM%type;
respuesta   EMP.COMM%type;

BEGIN
       monto := 100;
       respuesta := 0;
       calcularComision(monto, respuesta);
       dbms_output.put_line('el monto total por comisión es: '||respuesta);
END;
```

Manejo de excepciones

Excepciones

PL/SQL tiene mecanismos que permiten detectar y controlar fácilmente errores predefinidos y errores definidos por el usuario.

Definición

Cuando un error ocurre una excepción es activada. Esto significa que la ejecución del programa se detiene y el control es transferido a la sección de manejo de errores del bloque PL/SQL.

Existen excepciones predefinidas que son manejadas por el sistema como ZERO_DIVIDE.

Excepciones

Cuando se produce un error, este genera una excepción.

Por ejemplo:

```
BEGIN
  ...
  num := 1;
  den := 0;
  valor := num / den;
  ...
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    -- Instrucciones
    ...
END;
```

Las excepciones predefinidas más usuales son:

- Relacionadas con tipos de datos

  INVALID_NUMBER, VALUE_ERROR, ZERO_DIVIDE

- Relacionadas con operaciones de SQL

  DUP_VAL_ON_INDEX, NO_DATA_FOUND, TOO_MANY_ROWS

- Relacionadas con Cursores

  CURSOR_ALREADY_OPEN, INVALID_CURSOR

¿Cómo se activan?

```
DECLARE
        out_of_stock     EXCEPTION;
        number_on_hand   NUMBER(4);
BEGIN
        . . .
        IF number_on_hand  < 1  THEN
                RAISE out_of_stock;
        END IF;
        . . .
EXCEPTION
        WHEN out_of_stock THEN
        . . .
END;
```

Declaración explícita

```
DECLARE
        past_due  EXCEPTION;
        acc_num   NUMBER;
BEGIN
        . . .
```

Uso de la cláusula RAISE

Se pueden disparar excepciones definidas por el usuario con la sentencia:

```
RAISE nombre_excepcion;
```

La función SQLCODE y SQLERRM devuelven el código y el mensaje de error asociado a la última excepción predefinida que se ha producido, respectivamente.

El valor devuelto por SQLCODE es un NUMBER negativo y por SQLERRM un mensaje de 512 caracteres.

Uso de SQLCODE y SQLERRM

SQLCODE, siempre retornará el número del error en Oracle y un “0” en caso de éxito al ejecutarse una sentencia SQL.

SQLERRM, retornará el correspondiente mensaje de error para la situación ocurrida.

Estas funciones no pueden ser utilizadas directamente en una sentencia SQL, pero sí se pueden asignar su valor a alguna variable de programa y luego usar esta última en alguna sentencia.

Uso de SQLCODE y SQLERRM

```
DECLARE
  err_num NUMBER;
  err_msg VARCHAR2(255);

BEGIN

EXCEPTION
WHEN OTHERS THEN
        err_num := SQLCODE;
        err_msg := SQLERRM;

        DBMS_OUTPUT.put_line('Error:'||TO_CHAR(err_num));
        DBMS_OUTPUT.put_line(err_msg);
END;
        . . .
```

Excepciones

```
DECLARE
        salary          NUMBER(7,2);
        commision       NUMBER(7,2);
        comm_miss       EXCEPTION;
BEGIN
        SELECT sal, comm INTO salary, commision FROM emp
        WHERE empno = ‘03021’;
        IF commision IS NULL THEN
                RAISE comm_miss;  -- activa la excepción
        ELSE
                …..
        END IF;
EXCEPTION               -- inicio del área de manejadores de excepción
        WHEN comm_miss THEN
                         -- se procesa el error
END;
```

Excepciones: ejemplo

```
DECLARE
        pe_ratio NUMBER(3,1);
BEGIN
        . . .
        SELECT price / earnings INTO pe_ratio FROM stocks
                WHERE symbol = 'XYZ';
                -- esto puede causar una división por cero
        INSERT INTO stats( symbol,ratio) VALUES ('XYZ', pe_ratio);
        COMMIT;
EXCEPTION--aquí comienzan los manejadores de error
        WHEN ZERO_DIVIDE THEN
                INSERT INTO stats (symbol, ratio) VALUES ('XYZ', NULL);
                COMMIT;
        . . .
        WHEN OTHERS THEN
                ROLLBACK;
END;    -- el área de excepciones y el bloque finalizan aquí
```

Excepciones predefinidas más comunes

Excepciones predefinidas	Activación	sqlcode	error
CURSOR_ALREADY_OPEN		-6511	ORA-06511
DUP_VAL_ON_INDEX	Cuando un insert o un update produce un valor duplicado en un índice único	-1	ORA-00001
INVALID_CURSOR	Cuando se intenta abrir un cursor no declarado, o cerrar uno que fue cerrado, o leer de uno que no está abierto	-1001	ORA-01001
INVALID_NUMBER	Cuando se produce un error de conversión de una cadena a un número o viceversa	-1722	ORA-01722
NO_DATA_FOUND		+100	ORA-01403
TOO_MANY_ROWS		-1427	ORA-01427
ZERO_DIVIDE		-1476	ORA-01476
OTHERS	Resto de excepciones no mencionadas específicamente en el gestor de excepciones		

Excepciones

- ¿Es posible activar explícitamente **excepciones internas**?

```
DECLARE
        acct_type       INTEGER;
        . . .
BEGIN
        . . .
        IF acct_type NOT IN (1, 2, 3) THEN
                RAISE INVALID_NUMBER;
        END IF;
        . . .
EXCEPTION
        WHEN INVALID_NUMBER THEN
                ROLLBACK;
END;
```

Excepciones: No declaradas

```
EXCEPTION
        ...
        WHEN OTHERS THEN
                ROLLBACK;
END;
```

Excepciones

```
. . .
EXCEPTION
        WHEN error1  THEN
                -- maneja el error 1
        WHEN error2  OR error3  THEN
                -- maneja el error 2 y el 3
        . . .
        WHEN OTHERS THEN
                -- maneja todos los demás errores
END;
```

Uso de SQLCODE y SQLERRM

Estas funciones integradas son útiles cuando se usan en bloques de excepciones. Sirve para aclarar la situación del error ocurrido y obtener el mensaje de error asociado en OTHERS.

```
DECLARE
err_num          NUMBER;
err_msg          CHAR(100);
BEGIN
        . . .
EXCEPTION
        . . .
        WHEN OTHERS THEN
                err_num  := SQLCODE;
                err_msg  := SUBSTR(SQLERRM, 1, 100);
                INSERT INTO errors VALUES (err_num, err_msg);
END;
```

Usando EXCEPTION_INIT

Sirve para vincular una EXCEPTION definida por el usuario a un número de error determinado.

```
DECLARE
insufficient_privileges        EXCEPTION;
PRAGMA EXCEPTION_INIT(insufficient_privilegies, -1031);
--  En este caso el número de error -1031 se da cuando
--  uno realiza una operación sobre una tabla para la cual
--  no tiene el respectivo privilegio
BEGIN
            . . .
EXCEPTION
            WHEN insufficient_privilegies THEN
                        . . .
END;
```

Excepciones

- ¿Cómo continuar después que una excepción es activada?

  Las excepciones que no son manejadas se propagan hacia afuera desde la rutina que detecta la excepción hacia el programa que hace la llamada.

```
DECLARE
        pe_ratio NUMBER(3,1);
BEGIN
        DELETE FROM stats WHERE symbol = 'XYZ';
        -- Aquí se va a activar la excepción
        SELECT price / NVL(earnings, 0) INTO pe_ratio
        FROM stocks WHERE symbol = 'XYZ';
        -- Se desearía continuar acá
        INSERT INTO stats (symbol, ratio) VALUES ( 'XYZ', pe_ratio);
EXCEPTION
        WHEN ZERO_DIVIDE THEN
                . . .
END;
```

Excepciones

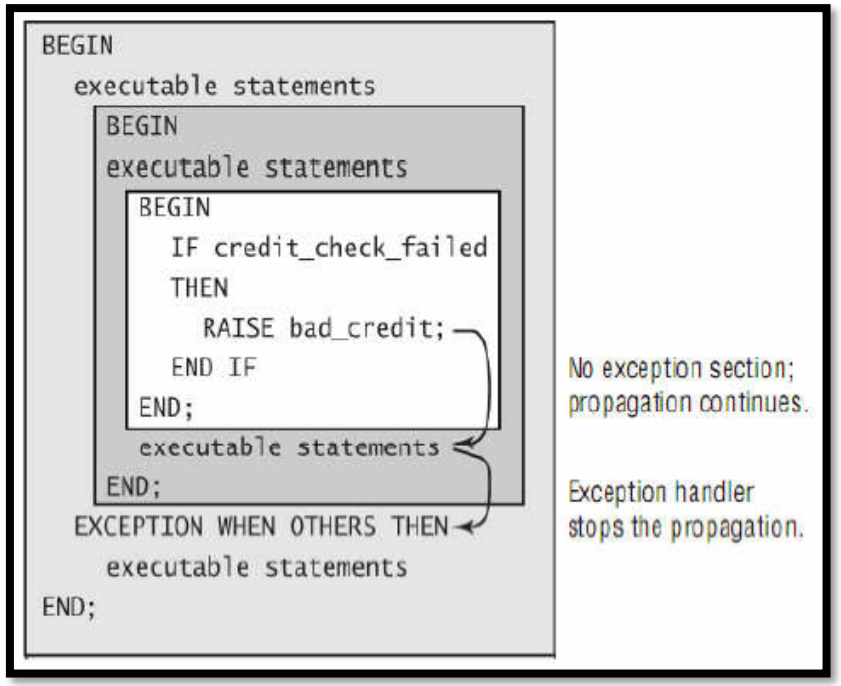
- ¿Cómo continuar después que una excepción es activada?
  La solución es la siguiente.

```
DECLARE
          pe_ratio   NUMBER(3,1);
BEGIN
          DELETE FROM stats WHERE symbol = 'XYZ';
          -- Inicio del sub-bloque
          BEGIN
                    SELECT price / NVL(earnings, 0) INTO pe_ratio
                    FROM stocks WHERE symbol = 'XYZ';
          EXCEPTION
                    WHEN ZERO_DIVIDE THEN
                              . . .
          END;
          -- fin del sub-bloque
          INSERT INTO stats (symbol, ratio) VALUES ( 'XYZ', pe_ratio);
EXCEPTION
                    . . .
END;
```

Excepciones

- ¿Cómo continuar después que una excepción es activada?

Las excepciones producidas en la sección de declaraciones de un bloque PL/SQL pasarán el control a la sección de excepciones del bloque que lo engloba.



Conclusiones

Existen dos tipos de cursores:

- Implícitos y
- Explícitos

Además, hemos revisado las principales características de un cursor.

Finalmente, se han presentado algunas excepciones predeterminadas usadas en PL/SQL

Referencias

- AR. Elmasri y S.B. Navathe. (2007). Fundamentos de Sistema de Base de Datos, 5ta edición
- Oracle Help Center. (26 de octubre de 2024). *Database PL/SQL User's Guide and Reference*. https://docs.oracle.com/cd/B19306_01/appdev.102/b14261/overview.htm#CJACCHHA

¡Gracias!

PUCP