PL/SQL

Francisco Moreno

Universidad Nacional

CURSORES

- Si una sentencia SELECT devuelve varias filas*, se debe usar un cursor para procesarlas en PL/SQL.
- Para declarar y usar un cursor se siguen estos pasos:

1. **Declarar** el cursor: Se le asigna un nombre al cursor y se le asocia una consulta
2. **Abrir** el cursor: Se ejecuta la consulta; en ese momento se está "apuntando" a la primera de las filas seleccionadas por la consulta
3. **Recuperar, tomar** las filas del cursor una a una → Proceso iterativo
4. **Cerrar** el cursor: Liberar espacio (disco y memoria)

* Los cursores funcionan para cualquier consulta ya sea que esta devuelva 0, 1 o varias filas

Sintaxis:

- `CURSOR nombre_cursor -- Se declara`
  `IS consulta SQL;`
- `OPEN nombre_cursor; -- Se abre`
- `FETCH nombre_cursor`
  `INTO` $var_1, var_2, \ldots, var_n$`; /*Se recupera la fila actual */`
- `CLOSE nombre_cursor; -- Se cierra`

$var_1, var_2, \ldots, var_n$ son las variables en las que se reciben los valores de las n columnas de la fila

Atributos (funciones) de los cursores:

%NOTFOUND: Es verdadero (TRUE) cuando el último FETCH ejecutado **no** recupera datos (no trajo fila)

%FOUND: Es verdadero cuando el último FETCH ejecutado recuperó datos (trajo una fila)

%ROWCOUNT: Devuelve el número de filas leídas hasta el momento

%ISOPEN: Es verdadero cuando el cursor está abierto

Ej: Sea la tabla:

```
DROP TABLE emp;
CREATE TABLE emp(
 cod NUMBER(8) PRIMARY KEY,
 nombre VARCHAR2(15),
 dep NUMBER(3),
 sueldo NUMBER(8) NOT NULL
);
INSERT INTO emp VALUES(12, 'Jessie J', 1,100);
INSERT INTO emp VALUES(15, 'Rihanna', 2, 300);
INSERT INTO emp VALUES(76, 'Aaliyah', 2, 400);
INSERT INTO emp VALUES(73, 'Emilia Clarke', 5, 500);
INSERT INTO emp VALUES(56, 'Jessy', 3, 100);
```

```
DECLARE
 CURSOR ord_c IS
 SELECT cod, dep FROM emp ORDER BY dep;
 codi emp.cod%TYPE;
 dpti emp.dep%TYPE;
BEGIN
 OPEN ord_c;
 LOOP
  FETCH ord_c INTO codi, dpti;
  EXIT WHEN ord_c%NOTFOUND;
  DBMS_OUTPUT.PUT_LINE(codi || ' ' || dpti);
 END LOOP;
DBMS_OUTPUT.PUT_LINE('Total: ' || ord_c%ROWCOUNT);
 CLOSE ord_c;
END;
/
```

Nota: si FETCH no recupera datos, las variables correspondientes quedan con el valor que tenían

Los cursores también se pueden procesar sin necesidad de abrirlos, ni hacerles FETCH, **ni** cerrarlos **explícitamente**. Estas acciones las puede hacer **automáticamente** un ciclo FOR así:

```
DECLARE
 CURSOR ord_c IS --Se declara el cursor
 SELECT cod, dep FROM emp ORDER BY dep;
BEGIN
 FOR mi_e IN ord_c LOOP
  DBMS_OUTPUT.PUT_LINE(mi_e.cod || ' ' || mi_e.dep);
 END LOOP;
--DBMS_OUTPUT.PUT_LINE('Total: ' || ord_c%ROWCOUNT);
END;
/
```

mi_e es una variable **local** del ciclo que automáticamente se declara del tipo del cursor

Se puede también evitar la declaración **explícita** de un cursor así:

```
BEGIN
 FOR mi_e IN (SELECT cod, dep FROM emp ORDER BY dep)
 LOOP
  DBMS_OUTPUT.PUT_LINE(mi_e.cod || ' ' || mi_e.dep);
 END LOOP;
END;
/
```

```
DECLARE
 total NUMBER(5); grantotal NUMBER(6) := 0;
 CURSOR emp_cur(v_dept NUMBER) IS
  SELECT nombre FROM emp WHERE dep = v_dept;
BEGIN
 FOR k IN 1..5 LOOP
   DBMS_OUTPUT.PUT_LINE('Dpto ' || k);
   total := 0;
   FOR mi_e IN emp_cur(k) LOOP
     DBMS_OUTPUT.PUT_LINE('Empleado: '|| mi_e.nombre);
     total := total + 1;
   END LOOP;
   DBMS_OUTPUT.PUT_LINE('Total emps en el dpto: ' || total);
   grantotal := grantotal + total;
 END LOOP;
DBMS_OUTPUT.PUT_LINE('Total emps en la empresa: '|| grantotal);
END;
/
```

Cursor con parámetro

Se puede declarar una variable del tipo de una tabla (mediante ROWTYPE) y se puede usar así:

```
DECLARE
 CURSOR emp_cur IS SELECT * FROM emp;
 mi_e emp%ROWTYPE;
BEGIN
 OPEN emp_cur;
 LOOP
  FETCH emp_cur INTO mi_e;
  EXIT WHEN emp_cur%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE(mi_e.cod || ' ' || mi_e.dep);
 END LOOP;
 CLOSE emp_cur;
END;
/
```

Todos los datos de **un** empleado quedan almacenados en `mi_e`

También se pueden declarar variables del tipo de un cursor:

```
DECLARE
 CURSOR ord_c IS
 SELECT cod, nombre, UTL_MATCH.EDIT_DISTANCE(nombre, 'Jessy') AS
   distancia
 FROM emp;
 mi_e ord_c%ROWTYPE;
BEGIN
 OPEN ord_c;
 LOOP
  FETCH ord_c INTO mi_e;
  EXIT WHEN ord_c%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(mi_e.cod || ' ' || mi_e.nombre || ' ' ||
   mi_e.distancia);
 END LOOP;
 CLOSE ord_c;
END;
/
```

Para similitud de cadenas ver también:
UTL_MATCH.JARO_WINKLER, LIKE y SOUNDEX

Un ejemplo con un doc. XML

```
DROP TABLE bodega;
CREATE TABLE bodega(
  id NUMBER(4) PRIMARY KEY,
  d XMLTYPE);

INSERT INTO bodega VALUES
(100, XMLTYPE('<Warehouse whNo="5">
               <Building>Owned</Building>
               </Warehouse>'));
INSERT INTO bodega VALUES
(200, XMLTYPE('<Warehouse whNo="8">
               <Building>Rented</Building>
               <Tel>21287</Tel>
               </Warehouse>'));
```

```
DECLARE
suma NUMBER(8) := 0;
BEGIN
 FOR mi_w IN (SELECT b.*,
                EXTRACTVALUE(d,'/Warehouse/@whNo') AS wh,
                EXTRACTVALUE(d,'/Warehouse/Building') AS bu
                FROM bodega b) LOOP
  DBMS_OUTPUT.PUT_LINE(mi_w.id || CHR(10) ||
                mi_w.d.EXTRACT('/*').getStringVal() ||
                mi_w.wh || ' ' || mi_w.bu);
      suma := suma + mi_w.wh;
 END LOOP;
DBMS_OUTPUT.PUT_LINE('Total: ' || suma);
END;
/
```

Como se trae todo el doc., EXTRACT('/*') se puede omitir.

Lo que está en rojo son expresiones XPath

- Nota: Es posible generar la salida de una consulta en formato XML. Ejemplo:

```
SELECT DBMS_XMLGEN.GETXML('SELECT *
FROM emp') docxml
FROM dual;
```

→ ¿Qué es dual?

El proceso contrario también es posible: tomar los datos de un documento XML y pasarlos a las columnas de una tabla, para ello lo mejor es usar el SQL*Loader

Un ejemplo con un doc. JSON

```
DROP TABLE depjson;
CREATE TABLE depjson (
   cod  NUMBER(8) PRIMARY KEY,
   dep_data JSON NOT NULL
);
```

```
INSERT INTO depjson VALUES (110,
'{
  "nombredep": "Música",
  "empleados": [
    {
      "nombre": "Aina Roxx",
      "trabajo": "Corista"
    },
    {
      "nombre": "Cath Coffey",
      "trabajo": "Cantante principal"
    }
  ]
}'
);
```

```
SELECT d.dep_data.empleados
FROM  depjson d;

SELECT d.dep_data.empleados[1]
FROM  depjson d;

SELECT d.dep_data.empleados[1].nombre
FROM  depjson d;
```

```
DECLARE
mijson JSON;
BEGIN
SELECT dep_data INTO mijson
FROM depjson;
DBMS_OUTPUT.PUT_LINE(JSON_SERIALIZE
(mijson PRETTY));
END;
/
--Si se quita el PRETTY sale ugly! ☺
```

Arrays (arreglos):

- Llamados *associative arrays* o *index-by tables* en PL/SQL
- Para crear un *array* se debe declarar primero su tipo y luego una variable de dicho tipo. Sintaxis:

```
TYPE tipo_array IS TABLE OF tipo_de_datos
INDEX BY BINARY_INTEGER;
mi_array tipo_array;
```

Declaración de variable de tipo tipo_array

También se permiten *arrays* indexados por cadenas de caracteres.

Ver *Associative arrays* en el menú Varios de la página del curso.

Arrays

- Los *arrays* en PL/SQL no están limitados por un número de posiciones dado
- Cualquier posición del *array* se puede usar para guardar valores, no tienen que guardar secuencialidad
- Se pueden usar índices ¡negativos!
- Si se intenta leer el dato de una posición no inicializada se genera error (excepción NO_DATA_FOUND → ver luego)

Atributos (funciones) de los *arrays* en PL/SQL

- COUNT: Devuelve el número de posiciones "activas" en el *array*
- DELETE: Borra elementos del *array*
- EXISTS: Dice si una posición está activa
- FIRST: Devuelve la menor posición activa
- LAST: Devuelve la mayor posición activa
- NEXT / PRIOR: Devuelve la próxima/anterior posición activa a la posición especificada

Ejercicio: analizar el comportamiento de estas funciones para *arrays* indexados por cadenas de caracteres

Array donde cada posición es un número de tres dígitos:

```
DECLARE
 TYPE t_num IS TABLE OF NUMBER(3) INDEX BY BINARY_INTEGER;
 mis_num t_num;
 i NUMBER;
BEGIN
 mis_num(9) := MOD(DBMS_RANDOM.RANDOM,1000);
 mis_num(4) := MOD(DBMS_RANDOM.RANDOM,1000);
 mis_num(2) := MOD(DBMS_RANDOM.RANDOM,1000);
 DBMS_OUTPUT.PUT_LINE(mis_num.COUNT);
 mis_num.DELETE(4);
 i := mis_num.FIRST;
 WHILE i IS NOT NULL LOOP
  DBMS_OUTPUT.PUT_LINE('Pos:'|| i || ' Val:' || mis_num(i));
  i := mis_num.NEXT(i);
 END LOOP;
END;
/
```

Array donde cada posición es un *array* (matriz):

```
DECLARE
TYPE t_num IS TABLE OF NUMBER(3) INDEX BY BINARY_INTEGER;
TYPE t_mat IS TABLE OF t_num INDEX BY BINARY_INTEGER;
mat t_mat;
BEGIN
FOR i IN 1..10 LOOP
 FOR j IN 1..20 LOOP
  mat(i)(j) := MOD(DBMS_RANDOM.RANDOM,1000);
 END LOOP;
END LOOP;
FOR i IN 1..mat.COUNT LOOP
 FOR j IN 1..mat(i).COUNT LOOP
  DBMS_OUTPUT.PUT_LINE('Fila ' || i || ' Col ' || j || ': ' ||
   mat(i)(j));
 END LOOP;
END LOOP;
END;
/
```

Array donde cada posición es un doc. XML

```
DECLARE
TYPE t_XML IS TABLE OF XMLTYPE INDEX BY BINARY_INTEGER;
mis_docs t_XML; -- Array de docs XML
k NUMBER(3) := 1;
BEGIN
FOR mi_e IN (SELECT d FROM bodega ORDER BY id) LOOP
  mis_docs(k) := mi_e.d;
  k := k+1;
END LOOP;
FOR i IN 1..mis_docs.COUNT LOOP
  DBMS_OUTPUT.PUT_LINE(mis_docs(i).getStringVal());
END LOOP;
END;
/
```

Array donde cada posición es una fila de una tabla:

```
DECLARE
 TYPE t_empleado IS TABLE OF emp%ROWTYPE
  INDEX BY BINARY_INTEGER;
 mis_emp t_empleado; -- Array de empleados
 k NUMBER(8) := 1;
BEGIN
 FOR mi_e IN (SELECT * FROM emp ORDER BY sueldo, cod) LOOP
  mis_emp(k) := mi_e;
  k := k+1;
 END LOOP;
 IF k > 1 THEN --Hay al menos un empleado
 DBMS_OUTPUT.PUT_LINE(mis_emp(1).cod || ' ' || mis_emp(1).sueldo);
 FOR i IN 2 .. mis_emp.COUNT LOOP
  DBMS_OUTPUT.PUT_LINE(mis_emp(i).cod || ' ' || mis_emp(i).sueldo || ' ' ||
                                                 mis_emp(i-1).sueldo);
 END LOOP;
 END IF;
END;
/
```

¿Qué hace este código? ¿Solución en SQL para este problema?

SQL: función analítica LAG*

Tarea: Comparar el resultado del programa anterior con la siguiente consulta SQL:

Lo que se desea traer de la fila anterior (ya que el desplazamiento es 1, la fila anterior según el ORDER BY sueldo, cod) con respecto a la fila actual (a la traída por el SELECT)

```
SELECT cod, sueldo,
    LAG(sueldo, 1) OVER
    (ORDER BY sueldo, cod) AS ant
FROM emp
ORDER BY sueldo, cod;
```

Desplazamiento (si es 1 se puede omitir)

*También existe LEAD

¿Otra solución?

- Tarea: Comparar también con esta:

```
SELECT e1.cod, e1.sueldo,
        (SELECT MAX(sueldo)
         FROM emp e2
         WHERE e2.sueldo < e1.sueldo OR
               (e2.sueldo = e1.sueldo AND
                e2.cod < e1.cod)
        ) AS ant
FROM emp e1
ORDER BY sueldo, cod;
```

Una prueba pequeña para probar el rendimiento de la solución en PL/SQL y de las dos consultas dadas:

```
BEGIN
 DELETE emp;
 FOR i IN 1..10000 LOOP
  INSERT INTO emp
  VALUES (i, 'Mariah'||i,
          CEIL(DBMS_RANDOM.VALUE(1,100)),
          CEIL(DBMS_RANDOM.VALUE(1,100000)));
 END LOOP;
END;
/
```

BULK COLLECT

- BULK COLLECT: permite llenar un *array* con las filas retornadas por una consulta sin necesidad de hacer un ciclo explícito.
- El *array* se llena desde la posición 1

BULK COLLECT

```
DECLARE
TYPE emp_type IS TABLE OF emp%ROWTYPE;
arr emp_type;
suma NUMBER(20) := 0;
BEGIN
--Se llena el array de empleados por medio de BULK COLLECT
SELECT * BULK COLLECT INTO arr FROM emp ORDER BY sueldo, cod;
IF arr.FIRST IS NOT NULL THEN -- Hay al menos un empleado
--Se recorre y se imprime el array de empleados
FOR i IN arr.FIRST .. arr.LAST LOOP
  suma := suma + arr(i).sueldo;
  DBMS_OUTPUT.PUT_LINE('Cod: ' || arr(i).cod || ' ' || suma);
END LOOP;
END IF;
END;
/
```