Chapter 8

Lecture (Lec 11 & 12 for Week 7

8.1 Cursor

In response to any DML statement the database creates a memory area, known as *context area*, for processing an SQL statement, which contains all information needed for processing the statement, for example, number of rows processed, etc.

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the *active set*.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

- Implicit cursors.
- Explicit cursors.

8.1.1 Implicit Cursor

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the SQL cursor, which always has the attributes like `%FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT`.

Attribute	Description
%FOUND	Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
%NOTFOUND	The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
%ISOPEN	Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
%ROWCOUNT	Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Example: Implicit Cursor

```
 DECLARE
   total_rows number(2);
BEGIN
   UPDATE emp
   SET salary = salary + 500;
   IF sql%notfound THEN
      dbms_output.put_line('no customers selected');
   ELSIF sql%found THEN
      total_rows := sql%rowcount;
      dbms_output.put_line( total_rows || ' customers selected ');
   END IF;
END;
/
```

8.1.2 Explicit Cursors

Explicit cursors are programmer defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns one or more rows.

```
CURSOR cursor_name IS select_statement;
```

4 Steps for Cursors:

1. Declaring the cursor for initializing in the memory
2. Opening the cursor for allocating memory
3. Fetching the cursor for retrieving data
4. Closing the cursor to release allocated memory

Similar to typical file operation.

Example:

```
DECLARE
   c_id customers.id%type;
   c_name customers.name%type;
   c_addr customers.address%type;
   CURSOR c_customers is
      SELECT id, name, address FROM customers;
BEGIN
   OPEN c_customers;
   LOOP
      FETCH c_customers into c_id, c_name, c_addr;
      EXIT WHEN c_customers%notfound;
      dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
   END LOOP;
   CLOSE c_customers;
END;
/
```

CURSOR FOR Loop: The cursor FOR loop is an elegant and natural extension of the numeric FOR loop in PL/SQL. With a numeric FOR loop, the body of the loop executes once for every integer value between the low and high values specified in the range. With a cursor FOR loop, the body of the loop is executed for each row returned by the query.

Syntax

```
  FOR record_index in cursor_name
LOOP
   {...statements...}
END LOOP;
```

Example

```
 CREATE OR REPLACE Function TotalIncome
   ( name_in IN varchar2 )
   RETURN varchar2
IS
   total_val number(6);

   cursor c1 is
     SELECT monthly_income
     FROM employees
     WHERE name = name_in;

BEGIN

   total_val := 0;

   FOR employee_rec in c1
   LOOP
      total_val := total_val + employee_rec.monthly_income;
   END LOOP;

   RETURN total_val;

END;
```

8.1.2.1 Variables in Explicit Cursor Queries

An explicit cursor query can reference any variable in its scope. When you open an explicit cursor, PL/SQL evaluates any variables in the query and uses those values when identifying the result set. *Changing the values of the variables later does not change the result set.*

In the following Example, the explicit cursor query references the variable factor. When the cursor opens, factor has the value 2. Therefore, `sal_multiple` is always 2 times sal, despite that factor is incremented after every fetch.

```
 DECLARE
  sal           employees.salary%TYPE;
  sal_multiple  employees.salary%TYPE;
  factor        INTEGER := 2;

  CURSOR c1 IS
    SELECT salary, salary*factor FROM employees
    WHERE job_id LIKE 'AD_%';

BEGIN
```

```
  OPEN c1;  -- PL/SQL evaluates factor

  LOOP
    FETCH c1 INTO sal, sal_multiple;
    EXIT WHEN c1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('factor = ' || factor);
    DBMS_OUTPUT.PUT_LINE('sal          = ' || sal);
    DBMS_OUTPUT.PUT_LINE('sal_multiple = ' || sal_multiple);
    factor := factor + 1;  -- Does not affect sal_multiple
  END LOOP;

  CLOSE c1;
END;
/


Result:

factor = 2
sal          = 4451
sal_multiple = 8902
factor = 3
sal          = 26460
sal_multiple = 52920
factor = 4
sal          = 18742.5
sal_multiple = 37485
factor = 5
sal          = 18742.5
sal_multiple = 37485
```

8.1.2.2 Explicit Cursors that Accept Parameters

You can create an explicit cursor that has formal parameters, and then pass different actual parameters to the cursor each time you open it. In the cursor query, you can use a formal cursor parameter anywhere that you can use a constant. Outside the cursor query, you cannot reference formal cursor parameters.

Following Example creates an explicit cursor whose two formal parameters represent a job and its maximum salary. When opened with a specified job and maximum salary, the cursor query selects the employees with that job who are overpaid (for each such employee, the query selects the first and last name and amount overpaid). Next, the example creates a procedure that prints the cursor query result set. Finally, the example opens the cursor with one set of actual parameters, prints the result set, closes the cursor, opens the cursor with different actual parameters, prints the result set, and closes the

cursor.

```
DROP TABLE EMPLOYEES;

CREATE TABLE EMPLOYEES
(ID NUMBER PRIMARY KEY,
 FIRST_NAME VARCHAR2(10),
 LAST_NAME VARCHAR2(10),
 JOB_ID VARCHAR2(10),
 SALARY NUMBER,
 CONSTRAINTS CHK_JOBID CHECK (JOB_ID IN ('ADMIN','MANAGER','FACULTY'))
);

---now insert some data ---

INSERT INTO EMPLOYEES VALUES(1,'A','B','ADMIN',30000);
INSERT INTO EMPLOYEES VALUES(2,'C','D','ADMIN',25000);
INSERT INTO EMPLOYEES VALUES(3,'E','F','ADMIN',33000);


INSERT INTO EMPLOYEES VALUES(4,'G','H','FACULTY',80000);
INSERT INTO EMPLOYEES VALUES(5,'I','J','FACULTY',120000);


INSERT INTO EMPLOYEES VALUES(6,'K','L','MANAGER',52000);
INSERT INTO EMPLOYEES VALUES(7,'M','N','MANAGER',50000);

COMMIT;


 ---Now create an annonymous block to create a parameterized cursor

 DECLARE
  CURSOR c (job VARCHAR2, max_sal NUMBER) IS
    SELECT last_name, first_name, (salary - max_sal) overpayment
    FROM employees
    WHERE job_id = job
    AND salary > max_sal
    ORDER BY salary;

  PROCEDURE print_overpaid IS
    last_name_   employees.last_name%TYPE;
    first_name_  employees.first_name%TYPE;
    overpayment_      employees.salary%TYPE;
  BEGIN
```