**LAB NO: 9** **Date:**

**CURSORS**

**Objectives:**
In this lab, student will be able to understand and use:

- Implicit and Explicit Cursors
- Cursor For Loops, Parameterized Cursors
- Transactions

**CURSORS:** A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set.

There are two kinds of cursors:

1. **Implicit Cursor**: A cursor that is constructed and managed by PL/SQL
2. **Explicit Cursor**: A cursor that the user constructs and manages.

**Implicit Cursor**

Oracle engine implicitly opens a cursor on the server to process each SQL statement and manages space, populates data and releases memory itself. Implicit cursor can be used to access information about the status of last insert, update, delete or single row select statements. SQL is the cursor name of the implicit cursor. The attributes of the implicit cursor are listed below.

| 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. |
| **%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**: A PL/SQL block to delete the student records of History Department in University database. Use implicit cursor attributes to check the success of delete operation.

```
DECLARE

dname CONSTANT student.dept_name%TYPE := 'History';

BEGIN

       DELETE FROM student WHERE dept_name = dname;
IF SQL%FOUND THEN
       DBMS_OUTPUT.PUT_LINE ('Delete succeeded for department:
       '|| dname);
ELSE
       DBMS_OUTPUT.PUT_LINE ('No students of department: ' ||
       dname);
END IF;

END;
/
```

**Output**: If there is no row that matches the delete query in *student* table.

```
No students of department: History
```

**Explicit Cursor**

Steps involved in using an explicit cursor and manipulating data in its active set are:

1. Declare a cursor mapped to a select statement.

2. Open the Cursor.
3. Fetch data from the cursor one row at a time into memory variables
4. Process the data held in the memory variables as required using a loop
5. Exit from loop after processing is complete
6. Close the cursor

Cursor is defined in the declarative part.

**Syntax**:

```
CURSOR [ parameter_list ] [ RETURN return_type ]
IS select_statement;
OPEN cursor_name;
Loop
FETCH cursor_name into variable_list;
End Loop

CLOSE cursor_name;
```

No memory is allocated at this point and only intimation is sent to the engine.

**Open** defines a private area named by the cursor name, executes the query , retrieves the data and creates the Active Data Set.

**Fetch** moves the data held in the active data set into the memory variable. It is placed in a Loop … End Loop which causes data to be fetched and processed until all rows are processed

**Close** disables the cursor and the active data set becomes undefined. After the fetch loop is executed the data needs to be closed. Close will release the memory occupied by the cursor.

**Example**: A PL/SQL block to list the student names of 'Comp. Sci.' department in University database.

```
DECLARE
dname CONSTANT student.dept_name%TYPE := 'Comp. Sci.';
CURSOR C1 is select name from Student where dept_name = dname;
sname student.name%TYPE;
BEGIN
```

```
DBMS_OUTPUT.PUT_LINE( '----------------------------------');
OPEN c1;
```

```
        LOOP
          -- Fetches student name into variable
          FETCH c1 INTO  sname;
          EXIT WHEN c1%NOTFOUND;
          DBMS_OUTPUT.PUT_LINE( RPAD('Name: ' || sname);
        END LOOP;
CLOSE c1;
DBMS_OUTPUT.PUT_LINE( '----------------------------------');
END;
/
```

**CURSOR FOR LOOPS**

The cursor FOR LOOP statement lets the user run a SELECT statement and then immediately loop through the rows of the result set. This statement can use either an implicit or explicit cursor (but not a cursor variable).
The cursor FOR LOOP statement implicitly declares its loop index as a %ROWTYPE record variable of the type that its cursor returns. This record is local to the loop and exists only during loop execution. Statements inside the loop can reference the record and its fields.

After declaring the loop index record variable, the FOR LOOP statement opens the specified cursor. With each iteration of the loop, the FOR LOOP statement fetches a row from the result set and stores it in the record. When there are no more rows to fetch, the cursor FOR LOOP statement closes the cursor. The cursor also closes if a statement inside the loop transfers control outside the loop or if PL/SQL raises an exception.

**Example**: A PL/SQL block to find the department having maximum Budget in University database, without using max() aggregate function.

```
DECLARE
CURSOR C1 is select * from Department;
vBudget Department.Budget%Type :=0;
vdname Department.dept_name%TYPE;
BEGIN
 For dept in C1
 LOOP
 IF dept.Budget > vBudget THEN
 vBudget := dept.Budget;
 vdname := dept.dept_name;
 END IF;
 END LOOP;
```

```
 Dbms_output.put_line('Max. Budget: ' || vBudget || '  Dept: '
                      || vdname);
END;
```

**WHERE CURRENT OF**

It states that the most recent row fetched from the table should be updated or deleted. Inside a cursor loop, WHERE CURRENT OF allows the current row to be directly updated. When the sessions open a cursor with the FOR UPDATE clause, all the rows in the return set will hold row level locks. When SELECT FOR UPDATE is associated with an explicit cursor, the cursor is called a FOR UPDATE cursor. Only a FOR UPDATE cursor can appear in the CURRENT OF clause of an UPDATE or DELETE statement.

**Example**: A PL/SQL block to increase the Budget of different departments in University database based upon current Budget. Increase the budget by 10%, 15% or 20% for the ranges 'greater than 100000', 'between 70000 and 100000' or 'less than or equal to 70000', respectively.

```
DECLARE
CURSOR C1 is select * from Department for update;
BEGIN
 For dept in C1
 LOOP
 IF dept.Budget <= 70000 THEN
 update Department SET Budget = Budget*1.2 where current of C1;
 ELSIF dept.Budget > 7000 and dept.Budget <= 100000 THEN
 update Department SET Budget = Budget*1.15 where current of C1;
 ELSE
 update Department SET Budget = Budget*1.1 where current of C1;
 END IF;
 END LOOP;
 END;
/
```

**PARAMATERIZED CURSORS**

PL/SQL allows parametrized cursor so that the cursor can be generic and the data that is retrieved from the table be changed according to need. The user can create an explicit cursor that has formal parameters, and then pass different actual parameters to the cursor each time it is opened. In the cursor query, a formal cursor parameter can be used

anywhere that a constant is used. Outside the cursor query, formal cursor parameters cannot be referenced.

**Syntax:**

```
CURSOR cursor_name( variable_name Datatype) is SELECT statement
```

**Example**: Based on the University database schema, the following example uses a parametrized cursor on the Instructor table to display the instructors of given department.

```
DECLARE
       cursor c(dname instructor.dept_name%TYPE) is select *
       from Instructor  where dept_name = dname;
BEGIN
       FOR tmp IN c('Comp. Sci.')
       LOOP
              dbms_output.put_line('EMP_No: '||tmp.ID);
              dbms_output.put_line('EMP_Name: '||tmp.name);
              dbms_output.put_line('EMP_Dept: '||tmp.dept_name);
              dbms_output.put_line('EMP_Salary:'||tmp.salary);
       DBMS_OUTPUT.PUT_LINE( '-------------------------------');
       END Loop;
END;
/
```

**TRANSACTIONS:**

A series of one or more statements that are logically related are termed as a Transaction. A transaction begins with the first executable SQL statement after a commit, rollback or connection made to the oracle engine. A transaction can be closed by using a commit or a rollback statement.

**COMMIT** ends the current transaction and makes permanent changes made during the transaction

**ROLLBACK** ends the transaction but undoes any changes made during the transaction

**SAVEPOINT** marks and saves the current point in the processing of a transaction.

**Syntax: Commit, Rollback & Savpoint**

```
COMMIT;
SAVEPOINT <Savepointname>
```

```
ROLLBACK TO <Savepointname>;

SAVEPOINT: Is optional and is used to rollback a transaction
partially as far as the specified savepoint
Savepointname: Is a savepoint created during the current
transaction
```

ROLLBACK can be fired with or without the SAVEPOINT clause. Rollback operation performed without the SAVEPOINT clause amounts to the following:

1. Ends the transaction
2. Undoes all the changes in the current transaction
3. Erases all savepoints in that transaction
4. Releases the transactional locks

**Example:**

Consider account (account_number, balance) table, populated with {(1, 200); (2, 3000); (3, 500)}. Withdraw an amount 200 and deposit 1000 for all the accounts. If the sum of all account balance exceeds 5000 then undo the deposit just made.

```
Declare
       Total_bal account.balance%TYPE;
Begin
       Update account set balance=balance-200;
       Savepoint deposit;
       Update account set balance=balance+1000;
       Select sum(balance) into total_bal from account;
       If total_bal > 5000 then
              Rollback to savepoint deposit;
       End If;
       Commit;
End;
```

**LAB EXERCISE:**

**Note:** Use University DB schema for the following, unless a different DB schema is explicitly specified

**Cursors:**

**CursorName %ISOPEN / FOUND / NOT FOUND:**

1. The HRD manager has decided to raise the salary of all the Instructors in a given department number by 5%. Whenever, any such raise is given to the instructor, a record for the same is maintained in the salary_raise table. It includes the Instuctor Id, the date when the raise was given and the actual raise amount. Write a PL/SQL block to update the salary of each Instructor and insert a record in the salary_raise table.

   **salary_raise**(Instructor_Id, Raise_date, Raise_amt)

**CursorName%ROWCOUNT:**

2. Write a PL/SQL block that will display the ID, name, dept_name and tot_cred of the first 10 students with lowest total credit.

**Cursor For Loops:**

3. Print the Course details and the total number of students registered for each course along with the course details - (Course-id, title, dept-name, credits, instructor_name, building, room-number, time-slot-id, tot_student_no )

4. Find all students who take the course with Course-id: CS101 and if he/ she has less than 30 total credit (tot-cred), deregister the student from that course. (Delete the entry in Takes table)

**Where Current of:**

5. Alter StudentTable(refer Lab No. 8 Exercise) by resetting column LetterGrade to F. Then write a PL/SQL block to update the table by mapping GPA to the corresponding letter grade for each student.

**Parameterized Cursors:**

6. Write a PL/SQL block to print the list of Instructors teaching a specified course.

7. Write a PL/SQL block to list the students who have registered for a course taught by his/her advisor.

**Transactions (COMMIT / ROLLBACK / SAVEPOINT):**

8. Write a PL/SQL block that updates the salary of 'Biology' department instructors by 20%. Subsequently, check the whether the department budget can support the raise. If not, undo the raise given to the instructors.

**ADDITIONAL EXERCISE:**

**Cursors**

1. Write a PL/SQL block that will display the name, department and salary of the top 10 highest paid instructors.

**Cursor For Loops:**

2. Repeat problem 1
   i. Using cursor for loops. ii. Using where current of.

**Parameterized Cursors:**

3. Write a PL/SQL block that would update the Bal_stock in the item_master(ItemId, Description, Bal_stock) table each time a transaction takes place with an entry in the item_transaction (TransID, ItemId, Quantity) table. The change in the item_master table depends on the itemId. If the item is present in the item_master table then Bal_stock is updated. Otherwise, itemID is inserted into the item_master table with ZERO as Bal_stock and raises an exception.

4. Write a PL/SQL block to find the number of courses offered and number of students of each department. Use a parameterized cursor which takes department name as a parameter and calculates the number of courses offered and number of students of that department.

**Transactions: COMMIT / ROLLBACK / SAVEPOINT:**

5. Write a PL/SQL block that will insert a new record in Takes (ID, course-id, sec-id, semester, year, grade) table. Check the total number of students registered for the course and if it exceeds 30, then undo the insert made to the Takes table.