Elective- I

Unit 3

Performing SQL operations from PL/SQL

Transaction :-

- READ : SELECT
- WRITE : INSERT INTO, UPDATE, DELETE
- COMMIT
- ROLLBACK

Properties of Transaction : ACID

1. Atomicity
2. Consistency
3. Isolation
4. Durability.

1. **Atomicity** :- All the operations of a transaction are considered as single unit i.e. either execute all the operations of a transaction or none of them.
(Do-all or Nothing)

2. **Consistency** :- If the transaction was in consistence state before transaction the database must preserve the consistency

3. **Isolation** : In concurrent transaction When multiple transaction Operation of one transaction must not interfair another operation.

4. Durability:- If a transaction performs its COMMIT operation then even the transaction fails all the change made during transaction will remain.

COMMIT: A commit terminates the current transaction by making all the changes made by Transaction permanent and releasing all the locks acquired by the transaction on the database

Syntax:

```
COMMIT;
```

ROLLBACK: A ROLLBACK statement is used to undo all the changes made by a Transaction and releases all the locks acquired by the transaction. ROLLBACK is useful to preserve the atomicity and consistency of a transaction ~~there a~~ when a transaction terminates without completing all of its operations.

Syntax:- ROLLBACK;

```
: ROLLBACK [TO SAVEPOINT SavepointName]
```

Savepoint option is useful when partial transaction is to be made permanent and partial transaction is to be ROLLBACKED. For this we have to create a SAVEPOINT in a Transaction as:

SAVEPOINT SavepointName;

A Rollback operation with a savepoint perform the following task:

1. Retains all the changes made by ~~transca~~ transaction upto the savepoint.
2. Cancels all the changes made by transaction after the savepoint
3. Release all the logs acquired by the transaction.

2078/07/14 Elective - I

Account (AccNo, AcName, AcType, Balance)
Transaction (TransID, AccNo, TDate, TType, Amount)
Now, write a PL/SQL block that withdraws Rs 10,000 from a particular account and deposit Rs 5,00,000 in the same account then check the balance of that account and if balance is greater than 50,00,000 then cendo the deposit.

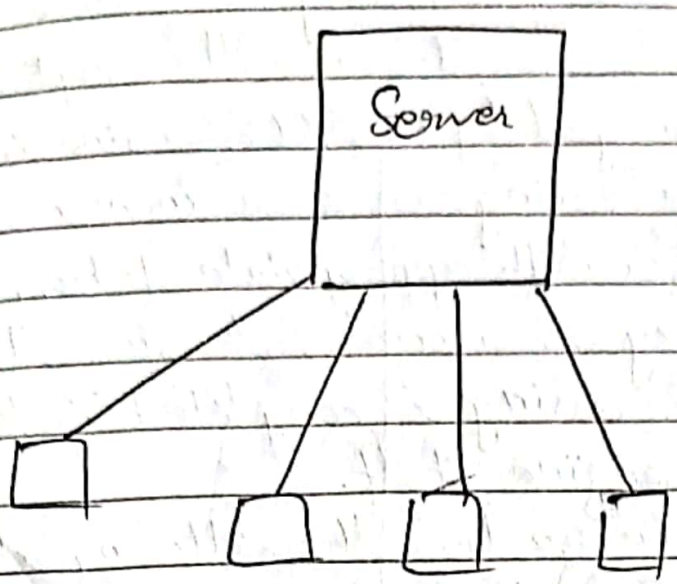
```
DECLARE
 acno NUMBER (10);
  bal FLOAT;
BEGIN
   acno := :acno;
  INSERT INTO Transaction
  VALUES (1005 (Ratnajaty), acno, SYSDATE, 'W', 10,000);

  UPDATE Account SET balance = balance - 10000
   WHERE AccNo = acno;
  SAVEPOINT no_deposit;

  INSERT INTO Transaction
  VALUES (1006, acno, SYSDATE, 'D', 500000);
  UPDATE Account
   SET balance = balance + 500000
   WHERE AccNo = acno;
  SELECT balance INTO bal
  FROM Account WHERE Accno = acno;
```

IF bal < 50 00 000 THEN
ROLLBACK TO SAVEPOINT
no_deposit;
END IF;
COMMIT;
END;

PL/SQL



SQL Cursor

its internal processing

The ORACLE engine uses a work area in memory for inorder to execute SQL statements. This work area is private to SQL operations and known as cursor.

The data stored in the cursor is called Active Data set. Conceptually the size of the cursor in memory is the size required to hold the number of records in active dataset. The actual size is however by determined by ORACLE engine's built in memory management capabilities and the amount of RAM available.

Cursors :-

Cursors are classified depending on the situations under which they are opened. There are two types of cursors as :-

a. **Implicit Cursor :**

This type of cursor is opened by ORACLE engine itself for its internal processing everytime when an SQL statement is processed. Since the implicit cursor is opened and managed by ORACLE engine internally the functions of reserving an area in memory, populating this area with appropriate data, processing the data in memory area and releasing the memory area, when the processing is completed is taken care by ORACLE engine itself.

Implicit cursor has the following attributes:

i) %ISOPEN : It returns TRUE if the cursor is open otherwise false. It is written as SQL%ISOPEN.

ii) %FOUND : It evaluates to TRUE if ~~insert~~ INSERT, DELETE or UPDATE operation affect at least one record, otherwise FALSE. It is written as SQL%FOUND

iii) %NOTFOUND : It is the logical opposite of %FOUND and it evaluates to TRUE if INSERT, UPDATE or DELETE operation does not affect any record, otherwise FALSE. It is written as SQL%NOTFOUND.

iv) %ROWCOUNT : It returns the number of records affected by INSERT, UPDATE, DELETE or SELECT statement. It is written as SQL%ROWCOUNT.

PL/SQL

For eg:- A PL/SQL block to change the department number of particular employee

Emp (EmpNo, EName, Post, Salary, Commision, Hiredate, deptno)

for eg :-

```
DECLARE
    eno VARCHAR(10);
    dno INT;
BEGIN
    eno := :eno;
    dno := :dno;
    UPDATE Emp SET deptno=dno
    WHERE EmpNo=eno;

    IF SQL% FOUND THEN
        dbms_output.put_line ('Employee Transferred
                               successfully');
    ELSE
        dbms_output.put_line('Employee does not exist');
    END IF;
    END;
```

* A PL/SQL block to input department number and increase salary by 5000 and counting the number of records affected.

```
DECLARE
    dno INTEGER;
    rows VARCHAR(15);
BEGIN
    dno := :dno;
    UPDATE Emp
    SET Salary = Salary + 5000
    WHERE Deptno = dno;
rows = TO_CHAR(SQL%.ROWCOUNT);
    IF SQL%.ROWCOUNT > 0 THEN
        dbms_output.put_line (rows || 'records affect');
    ELSE
        dbms_output.put_line('No employee is
                              given department');
    END IF;
    END;
```

4. Explicit Cursor :

When individual records in a table have to be processed inside a PL/SQL block, a cursor is used. The cursor known as Explicit or user defined cursor will declare an map to SQL query inside the declare section. The basic steps involved in using Explicit cursor and manipulating data in its active dataset are :-

1. Declare a cursor map to a SQL select statement that retrives data for processing.
2. Open the cursor using Open statement
3. Fetch data from the cursor one record at a time into memory variables using fetch command.
4. Process the data of memory variables as required using a loop.
5. Exit from the loop after the processing is complete.
6. Close the cursor using close statement.

Elective - I

Explicit Cursor:- Explicit Cursor also have 4 attributes similar to Implicit cursor and they are:-

i) %ISOPEN : Syntax : CursorName %ISOPEN
ii) %FOUND : Syntax : CursorName %FOUND
iii) %NOTFOUND : Syntax : CursorName %NOTFOUND
iv) %ROWCOUNT : Syntax : CursorName %ROWCOUNT

Cursor Declaration :

An explicit cursor is declared inside the declare section of PL/SQL block using a cursor name as:

CURSOR CursorName IS SELECT Statement;

Cursor Declaration does not occupy any space in memory.

Opening a Cursor:- OPEN → Active Data Set.

Opening the cursor executes the query and creates the active dataset that contains all the records of query result. An OPEN statement retrives record from database table and places the record in the cursor which is the private SQL area in memory

OPEN CursorName;

Fetching a Record from a Cursor :- FETCH →

The FETCH statement retrives the record from the active dataset into memory variables declared in a PL/SQL Block one record at a time. Each time a FETCH is executed the cursor pointer is advanced to the next record in the active dataset. A fetch statement must be inside a loop.

Syntax :-

FETCH CursorName INTO Variable1, Variable2,...;

Closing the Cursor : CLOSE

The ~~close~~ CLOSE statement deactives the active dataset and releases the memory space occupied by cursor in both server and client.

Syntax :-

CLOSE CursorName;

for eg :-

Write a PL/SQL code Block to calculate simple Interest of all saving accounts at the rate of 6.5%, Insert this information into the transaction table and UPDATE the account table

Account (AccNo, AcName, AcType, balance)
Transaction (TransID, AccNo, TDate, TType, Amount)

```
DECLARE
  CURSOR Interest_Calc IS SELECT AccNo, balance
        FROM Account WHERE AcType = 'Saving';
  acno Account.Accno%TYPE;
  bal Account.balance %TYPE;
  interest FLOAT;
  i INTEGER;
BEGIN
    OPEN Interest_Calc;
     IF Interest_Calc %ISOPEN THEN
       i:=1;
    LOOP
     FETCH Interest_Calc INTO acno, bal;
     EXIT WHEN Interest_Calc %NOTFOUND;

    IF Interest_Calc % FOUND THEN

     interest := (bal * 6.5)/(100 * 12);
     INSERT INTO Transaction
     VALUES (i, acno, SYSDATE, 'D', (interest);
     UPDATE Account SET balance = balance + interest
      WHERE Accno = acno;
     END IF;
    i := i+1;
    COMMIT
```

```
END LOOP;
ELSE
    System.output.put_line)
      ('Unable to open cursor')
END IF;
CLOSE Interest_Calc;
END;
```

PL/SQL Elective-I

Parameterized Cursor:

CURSOR CursorName (Variable datatype,.....) As SELECT Statement;

In a explicit cursor we can pass arguments or parameters and such cursors are known as parameterized cursor. It is declared as:

A parameterized cursor is opened as :-

OPEN CursorName (Value/expression);

Write a PL/SQL block having a parameterized cursor that increases the salary of all employee working in sales department by 5000 assuming that we have following table with some records.

Dept (deptno, dname, dloration)

Emp (Empno, EName, Post, hiredate, Salary, comm, deptno)

Imp

For eg:-

```
DECLARE
    CURSOR dno_find IS SELECT deptno
        FROM Dept WHERE dname='Sales';
    CURSOR inc_sal(dno INTEGER) IS
        SELECT Empno * FROM Emp
        WHERE Deptno = dno;
dpn dept.deptno%TYPE;
eno emp.empno%TYPE;
BEGIN
    OPEN dno_find;
    Loop
      FETCH dno_find into dpn;
      EXIT WHEN dno_find%NOTFOUND;
      OPEN inc_sal(dpn);
    Loop
      FETCH inc_sal INTO eno;
    EXIT WHEN inc_sal%NOTFOUND;
    UPDATE Emp SET salary = Salary + 5000
        WHERE Empno=eno;
    END Loop;
END Loop;
    CLOSE inc_sal;
    CLOSE dno_find;
     COMMIT;
     END;
```