CURSORS AND TRIGGERS

Q1.

a) What is a cursor? What types of cursors are supported in PL/SQL? Explain Cursor attributes.
b) What is the purpose of cursor in PL/SQL? Name the types of cursors used in PL/SQL? Explain Cursor attributes.
c) What is the importance of cursor for loop? How it simplifies the operation? Explain with suitable example.

Answers:

(a) The oracle uses a work area of memory to execute SQL statements. This work area is also called *context area* and data stored in this *context area* is called the **Active Data Set**. A cursor is a pointer to this *context area.*

Types of cursors:

There are two types of cursors: (i) Implicit Cursor (ii) Explicit Cursor

Implicit Cursor The oracle implicitly (internally or automatically) opens a cursor for each SQL statement. Since the implicit cursors are opened and managed by oracle internally. So there is no need to open and manage the cursors by the user. Implicit cursor attributes can be used to access information about status of last *insert, update, delete* or single row *select* statement.

Explicit Cursor When individual records in a table have to be processed inside a PL/SQL code block a *cursor* is used. This cursor will be declared and mapped to an SQL query in the Declare Section of the PL/SQL block and used within the Executable Section. This cursor needs to be opened before the reading of the rows can be done, after which the cursor is closed. Cursor marks the current position in an active set. A cursor thus created and used is known as an *Explicit Cursor.*

Cursor Attributes Cursors have the following four attributes:

Attribute Name	**Description**
%ISOPEN	It is true if cursor is open and FALSE if cursor is not open or cursor is closed. It is used only with Explicit Cursors.
%FOUND	TRUE if at least one row was processed or a record was fetched successfully from the opened cursor and FALSE otherwise.
%NOTFOUND	TRUE if no row were processed or if record was not fetched successfully and FALSE otherwise.
%ROWCOUNT	It returns the number of rows/records processed by a cursor.

(b) The most commonly used loop with in a PL/SQL block is the **FOR** *variable* **IN** *value* construct. This is an example of machine defined loop exit i.e. when all the *values* in **FOR** construct are exhausted looping stops.

Syntax: **FOR** *variable* **IN** *cursorname*

Here, the verb **FOR** automatically creates the *variable* of type *%rowtype.* Each record in the opened cursor becomes a value for the memory variable of *%rowtype.* The **FOR** verb ensures that a row from the cursor is loaded in declared *variable* and loop executes once. This goes until all the rows of the cursor have been loaded into the *variable.* After this loop stops.

A CURSOR FOR LOOP automatically does the following:

- Implicitly declares its loop index as a *%rowtype* record.
- Open a cursor.
- Fetches a row from the active set for each loop iteration.
- Closes the cursor when all rows have been processed.

Cursor can be closed even when an *exit* or *goto* statement is used to leave the loop prematurely, or an exception is raised inside the loop.

Example

The HRD manager has decided to raise the salary for all employees in department number 30 by 0.05.

Table Name:

Empno	**Deptno**	**Sal**
100	10	5000

101	30	6000
102	20	4000
103	30	5500

```
DECLARE
        CURSOR c_emp IS
        SELECT empno,sal
        FROM emp
        WHERE deptno=30;
BEGIN
        FOR rec IN c_emp
        LOOP
                UPDATE emp
                        SET sal=rec.sal+(rec.sal * .05)
                        WHERE empno=rec.empno;
        END LOOP;
        COMMIT;
END;
```

The above PL/SQL code block will function as follows:
The block implicitly declares *rec* as belonging to type *c_emp%rowtypw* and retrieves all the records having deptno 20.the salary for each record will be updated as required one by one and loaded into *rec* by the **FOR** verb. The cursor closes automatically when all the records in the cursor have been processed. This situation is sensed by **FOR** verb which causes the loop to exit.

(c) (i) Active Data Set (ii) SQL%ISOPEN (iii) Explicit (iv) END LOOP.

Q2. Answers the questions based on the table EMP given below:

Table: EMP

Column Name	Data Type	Size	Description
Empno	NUMBER	4	Employee's Identification Number
Ename	VARCHAR2	30	Employee's Name
Job	VARCHAR2	15	Employee's Designation
Sal	NUMBER	8,2	Employee's Salary
DeptNo	NUMBER	2	Employee's Department id
Commission	NUMBER	7,2	Employee's Commission

(a) Write a PL/SQL code to display the Empno, Ename and Job of employees of DeptNo 10 with CURSOR FOR LOOP Statement.
(b) Write a PL/SQL code to increase the salary of employees according to the following conditions:
    Salary of DeptNo 10 employees increased by 1000.
    Salary of DeptNo 20 employees increased by 500.
    Salary of DeptNo 30 employees increased by 800.
  Also store the *EmpNo*, *old salary* and *new salary* in a Table TEMP having three columns Empid, Old and New.
(c) Create a trigger on EMP table which verify that no record has the Commission greater than salary of an employee in table Emp.
(d) Suppose the table Result in the database, which has the following structure: Result (RollNo, Sub1, Sub2, Sub3, Sub4, Total, Percentage). The RollNo and marks in each subject is stored in database. Write a PL/SQL code to calculate the Total and Percentage of each student and update the database.

Ans.
(a) DECLARE
```
        CURSOR C1 IS SELECT EMPNO, ENAME, JOB FROM EMP WHERE DEPTNO = 10;
        BEGIN
                FOR REC IN C1 LOOP
                        DBMS_OUTPUT.PUT_LINE ('EMPNO'|| REC.EMPNO);
                        DBMS_OUTPUT.PUT_LINE ('ENAME'|| REC.ENAME);
                        DBMS_OUTPUT.PUT_LINE ('JOB'|| REC.JOB);
                END LOOP;
        END;
```

(b) DECLARE

```
                OLDSAL NUMBER;
                NEWSAL NUMBER;
                CURSOR C1 IS SELECT * FROM EMP;
                REC C1%ROWTYPE;
        BEGIN
                OPEN C1;
                LOOP
                        FETCH C1 INTO REC
                        EXIT WHEN C1%NOTFOUND;
                        SELECT SAL INTO OLDSAL FROM EMP WHERE EMPNO=REC.EMPNO;
                        IF REC.DEPTNO=10 THEN
                                UPDATE EMP SET SAL=SAL+1000 WHERE EMPNO=REC.EMPNO;
                                SELECT SAL INTO NEWSAL FROM EMP WHERE EMPNO=REC.EMPNO;
                        END IF;
                        IF REC.DEPTNO=20 THEN
                                UPDATE EMP SET SAL=SAL+500 WHERE EMPNO=REC.EMPNO;
                                SELECT SAL INTO NEWSAL FROM EMP WHERE EMPNO=REC.EMPNO;
                        END IF;
                        IF REC.DEPTNO=30 THEN
                                UPDATE EMP SET SAL=SAL+800 WHERE EMPNO=REC.EMPNO;
                                SELECT SAL INTO NEWSAL FROM EMP WHERE EMPNO=REC.EMPNO;
                        END IF;
                        INSERT INTO TEMP (EMPID, OLD, NEW)
                        VALUES (REC.EMPNO, OLDSAL, NEWSAL);
                END LOOP;
                CLOSE;
        END;
```

(c) CREATE OR REPLACE TRIGGER UPDATE_CHECK

```
        BEFORE UPDATE ON EMP FOR EACH ROW
        BEGIN
                IF :NEW.SAL<:OLD.SAL THEN
                        RAISE_APPLICATION_ERROR (-20001,'NEW SALARY CANNOT BE LESS
                        THAN OLD SALARY');
                END IF;
        END;
```

(d) DECLARE

```
                T NUMBER;
                PER NUMBER;
                CURSOR C1 IS SELECT * FROM RESULT;
                REC C1%ROWTYPE;
        BEGIN
                OPEN C1;
                LOOP
                        FETCH C1 INTO REC;
                EXIT WHEN C1%NOTFOUND;
                T := REC.S1+REC.S2+REC.S3+REC.S4;
                PER:=T/4;
                UPDATE RESULT SET TOTAL=T, PERCENTAGE=PER WHERE RNO=REC.RNO;
            END LOOP;
            CLOSE C1;
        END;
```

Q 3.

(a) What is trigger? Write the syntax for defining a trigger.
(b) Differentiate between *row-level* and *statement-level* triggers.
(c) What are the advantages/uses of using triggers?
(d) What is the default type of triggers?
(e) What is instead of trigger?

Answer

(a) A trigger is a stored procedure that is automatically fired (executed) when an INSERT, UPDATE or DELETE statements issued against the associated table. The trigger is not explicitly called by user. A trigger defines an action the database should take when some database related events occur. A trigger can include SQL and PL/SQL statements to execute it as a unit and it can invoke other stored procedures.

Syntax:

```
CREATE [or REPLACE] TRIGGER trigger_name
        BEFORE | AFTER
        DELETE OR INSERT OR UPDATE [OF column_name,…]        Triggering
        ON table_name                                                    event
        [FOR EACH ROW [WHEN condition]] }Triggering Restriction

  DECLARE
     variable declaration;
     constant declaration;
   BEGIN
   PL/SQL subprogram body;                          Triggering Action
   [EXCEPTION
      exception PL/SQl block;]
   END;
```

(b) **Row Level Trigger**-> a *row level* trigger is fired each time a row in the table affected by the triggering statement. For example, if an UPDATE statement updates multiple rows of a table, a *row trigger* is fired once for each row affected by the UPDATE statement. If the triggering statement affects no rows, the trigger is not executed at all. *Row level* triggers are created by using FOR EACH ROW clause in the CREATE TRIGGER command.

Statement Level Trigger -> A *statement trigger* is fired once on behalf of the triggering statement, independent of the number of rows the triggering statement affects. A *statement trigger* fires even if no row affected. For example when an UPDATE command update 10 rows, the commands contained in the trigger executed only once and not for every processed row. *Statement level* triggers are the *default types* of triggers created by the CREATE TRIGGER command.

(c) Advantages of Triggers:

- To enforce complex integrity constraints.
- To prevent invalid transactions.
- To maintain replicate tables.
- To audit data modifications.
- To enforce complex security authorizations.
- To enforce complex business rules.

(d) *Statement level* triggers are the *default types* of triggers created by the CREATE TRIGGER command.

(e) An *INSTEAD OF* trigger is defined on a *view* rather than on a *table*. Such triggers are used to overcome the restrictions placed by Oracle on any view, which is non-updateable. Prior to version 7.3, it was impossible to issue DML statements- INSERT, DELETE, UPDATE- against any view which contained a join. In oracle 8, INSTEAD OF triggers are defined on the same event as their table counterparts: INSERT, DELETE and UPDATE.*INSTEAD OF* triggers give us the ability to perform DML against any view definition. There are few restrictions on *INSTEAD OF* triggers:

- They are available only at *row level.*
- They can be applied only on views and not to tables.

Q 4.

a) Create a trigger, which verify that updated salary of employee must be greater than his/her previous salary.
b) Find the errors from the following PL/SQL code and rewrite the corrected code underlining the correction made:

```
CREATE ASWELLAS REPLACE TRIGGER DEPT_UP
    AFTER UPDATE ON EMP FOR EVERY ROW
DECLARE
   v_num NUMBER (3);
BEGIN
SELECT COUNT (*) INTO v_num FROM EMP WHERE deptno=20;
IF v_num > 5
  Raise_application_error (-20001,'Cannot Exceed 5');
END;
```

Answer

(a)
```
CREATE OR REPLACE TRIGGER EMP_SUNDAY
BEFORE INSERT OR UPDATE OR DELETE ON EMP
BEGIN
    IF RTRIM (UPPER (TO_CHAR (SYSDATE,'DAY'))) = 'SUNDAY' THEN
    RAISE_APPLICATION_ERROR (-20002,'NO OPERATION CAN BE
    PERFORMED ON SUNDAY');
    END IF;
END;
```

(b)
```
CREATE OR REPLACE TRIGGER UPDATE_CHECK
BEFORE UPDATE ON EMP
BEGIN
    IF :NEW.SAL <  :OLD.SAL THEN
    RAISE_APPLICATION_ERROR (-20003,'NEW SALARY CAN NOT BE
    LESS THAN OLD SALARY');
    END IF;
END;
```

(c)
```
CREATE OR REPLACE TRIGGER DEPT_UP
    AFTER UPDATE ON EMP FOR EACH ROW
DECLARE
    v_num NUMBER (3);
BEGIN
    SELECT COUNT (*) INTO v_num FROM EMP WHERE deptno=20;
    IF v_num > 5 THEN
    Raise_application_error (-20001,'Cannot Exceed 5');
END IF;
END;
```