Name: Prashant Suresh Shirgave
Roll No: 3 **Batch:** T1
Class: TY(CSE-AIML)

Experiment No.10

Title: Cursors, and triggers using PL/SQL

Objective: Demonstrate Cursors, and triggers using PL/SQL.

Theory:
Cursor:

A cursor is a pointer to this context area. PL/SQL controls the context area through a Cursor. 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 byit. Therefore, cursors are used as to speed the processing time of queries in large databases.

Cursors can be of two types:

- **Implicit cursors**
- **Explicit cursors**

Implicit cursors are automatically created by Oracle whenever an SQL statement is executedwhen there is no explicit cursor defined for the statement.

Cursor attributes like:

- **%FOUND,**
- **%ISOPEN,**
- **%NOTFOUND,**
- **%ROWCOUNT.**

Following image describes these attributes briefly:

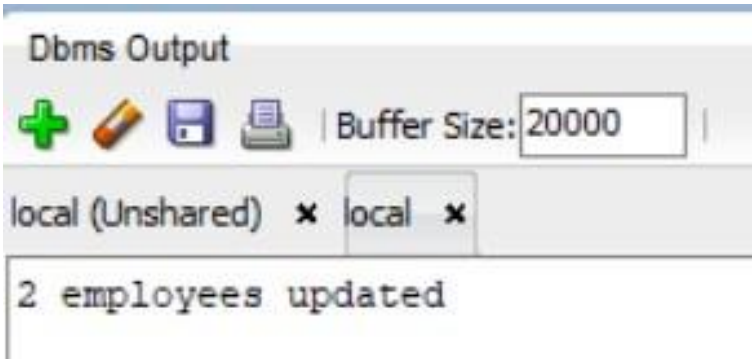
Attribute	Description
%FOUND	Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
%NOTFOUND	The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected 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.

1)Write a cursor that will increase salary by 1000 of those whose age is less than 30.

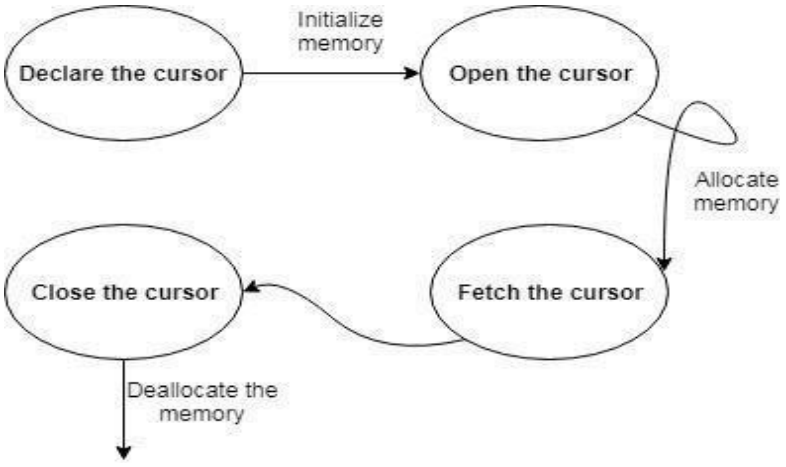
```
DECLARE
   total_rows NUMBER(2);
BEGIN
   UPDATE Employee
   SET salary = salary + 1000
   WHERE age < 30;

   IF SQL%NOTFOUND THEN
      DBMS_OUTPUT.PUT_LINE('No employees found under age 30');
   ELSIF SQL%FOUND THEN
      total_rows := SQL%ROWCOUNT; DBMS_OUTPUT.PUT_LINE(total_rows || '
      employees updated');
   END IF;

   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```



Following image denotes the life cycle of a typical explicit cursor:



2) Write a cursor that will increase salary by 10% if salary is > 10000 otherwise by 5%.

```
    DECLARE
  e_id  employe.emp_id%TYPE; e_name
  employe.emp_name%TYPE; e_salary
  employe.emp_salary%TYPE;

  CURSOR s1 IS
    SELECT emp_id, emp_name, emp_salary FROM employe;
BEGIN
  OPEN s1;

  LOOP
    FETCH s1 INTO e_id, e_name, e_salary;
    EXIT WHEN s1%NOTFOUND;

    IF (e_salary > 10000) THEN
      e_salary := e_salary * 1.10;
        ELSE
      e_salary := e_salary * 1.05;
    END IF;

    UPDATE employe
    SET emp_salary = e_salary
    WHERE emp_id = e_id;
  END LOOP;

      CLOSE s1;
       COMMIT;
    EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;

SELECT * FROM employe;
```

	EMP_ID	EMP_NAME	EMP_SALARY
1	1	Prashant	13200
2	2	Rushi	9450
3	3	Parshav	16500

	EMP_ID	EMP_NAME	EMP_SALARY
1	1	Prashant	12000
2	2	Rushi	9000
3	3	Parshav	15000

Trigger:

A trigger is a statement that the system executes automatically as a side effect of a modificationtothe Database.

A trigger is invoked before or after a data row is inserted, updated, or deleted.

1. A trigger is associated with a database table.
2. Each database table may have one or more triggers.
3. A trigger is executed as part of the transaction that triggered it.

Triggers are critical to proper database operation and management. For example:

1. Triggers can be used to enforce constraints that cannot be enforced at the DBMS design and implementation levels.
2. Triggers add functionality by automating critical actions and providing appropriate warningsand suggestions for remedial action. In fact, one of the most common uses for triggers is to facilitate the enforcement of referential integrity.
3. Triggers can be used to update table values, insert records in tables, and call other stored procedures.

Oracle recommends triggers for:

1. Auditing purposes (creating audit logs).
2. Automatic generation of derived column values.
3. Enforcement of business or security constraints.
4. Creation of replica tables for backup purposes.

Syntax:

```
CREATE OR REPLACE TRIGGER trigger_name
[BEFORE / AFTER] [DELETE / INSERT / UPDATE OF column_name] ON table_name
[FOR EACH ROW]
[DECLARE]
[variable_namedata type[:=initial_value] ]
BEGIN
PL/SQL instructions;
.....
END;
```

A trigger definition contains the following parts

- **The triggering timing:** BEFORE or AFTER. This timing indicates when the trigger's PL/SQL code executes; in this case, before or after the triggering statement is completed.
- **The triggering event:** the statement that causes the trigger to execute (INSERT, UPDATE, or DELETE).

- **The triggering level:** There are two types of triggers: statement-level triggers and row-level triggers.

- A statement-level trigger is assumed if you omit the FOR EACH ROW keywords. This type of trigger is executed once, before or after the triggering statement is completed. This is the default case.

- A row-level trigger requires use of the FOR EACH ROW keywords. This type of trigger is executed once for each row affected by the triggering statement. (In other words, if you update10 rows, the trigger executes 10 times.)

- **The triggering action:** The PL/SQL code enclosed between the BEGIN and END keywords.Each statement inside the PL/SQL code must end with a semicolon ";".

Example Triggers

1.Before Insert Trigger Example

This trigger prevents negative values in the age column of the people table.

```
CREATE TABLE people (age INT, name VARCHAR2(150));

CREATE OR REPLACE TRIGGER agecheck
BEFORE INSERT ON people
FOR EACH ROW
BEGIN
   IF :NEW.age < 0 THEN
      :NEW.age := 0;END
   IF;
END; /
INSERT INTO people VALUES (-20, 'aaa');
INSERT INTO people VALUES (30, 'bbb');
INSERT INTO people VALUES (-20, 'prashant');
INSERT INTO people VALUES (30, 'rushi');
SELECT * FROM people;
```

	AGE	NAME
1	0	aaa
2	30	bbb
3	0	prashant
4	30	rushi

2. Before Update Trigger Example

This trigger restricts the amount column to a range between 0 and 100 in the account table.

```
CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));INSERT
INTO account (acct_num, amount) VALUES (137, 50); INSERT INTO account
(acct_num, amount) VALUES (141, 80); SELECT * FROM account;
    CREATE TRIGGER upd_check BEFORE UPDATE ON accountFOR
    EACH ROW
    BEGIN
      IF :NEW.amount < 0 THEN
        :NEW.amount := 0;
      ELSIF :NEW.amount > 100 THEN
        :NEW.amount := 100;END
      IF;
    END;

    UPDATE account SET amount = 150 WHERE acct_num = 137;
    SELECT * FROM account;
```

	ACCT_NUM	AMOUNT
1	137	100
2	141	80

	ACCT_NUM	AMOUNT
1	137	50
2	141	80

3.This example creates an account table and a trigger ins_sum to maintain a running totalof inserted amounts.

```
    CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
    CREATE TRIGGER ins_sum BEFORE INSERT ON account FOR EACH ROW SET @sum =@sum +
    NEW.amount;
    SET @sum = 0;
    INSERT INTO account VALUES (137, 14.98), (141, 1937.50), (97, -100.00);
    SELECT @sum AS 'Total amount inserted';
```

	Total amount inserted
▶	1852.48

Outcome: Students will able to Cursors, and triggers using PL/SQL.