TRIGGERS

A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs.

It is a special type of stored procedure that is automatically executed in response to certain database events such as an DML statements (INSERT, UPDATE, or DELETE operation) or DDL statements (CREATE, ALTER or DROP).

Triggers can be defined on tables, views, schema or database with which the event is associated. They can be defined to execute before or after the triggering event and can be defined to execute for every row or once for each statement.

Triggers can be used to perform actions such as data validation, enforcing business rules, or logging.

Triggers are a powerful feature of DBMS that allow developers to define automatic actions based on database events.

The trigger description contains three parts:

1. Event: a change to the database that activate the trigger (insertion/deletion/updating).
2. Condition: a query or test runs when trigger is activated.
3. Action: a procedure executes when an event occurs and the condition is true.

Syntax for creating a Trigger in DBMS:

CREATE [**OR REPLACE**] **TRIGGER** <trigger_name>
{**BEFORE** | **AFTER** | **INSTEAD OF**}
{**INSERT** [**OR**] | **UPDATE** [**OR**] | **DELETE**}
[**OF** <col_name>]
ON <table_name>
[**REFERENCING OLD AS o NEW AS n**]
[**FOR EACH ROW**]
WHEN (<condition>)
DECLARE
    <Declaration-statements>
BEGIN
    <Executable-statements>
EXCEPTION
    <Exception-handling-statements>
END;

where

- **CREATE [OR REPLACE] TRIGGER <trigger_name>**: It creates or replaces an existing trigger with the trigger_name.
- **{BEFORE | AFTER | INSTEAD OF}**: This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- **{INSERT [OR] | UPDATE [OR] | DELETE}**: This specifies the DML operation.
- **[OF <col_name>]**: This specifies the column name that would be updated.
- **[ON <table_name>]**: This specifies the name of the table associated with the trigger.
- **[REFERENCING OLD AS o NEW AS n]**: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- **[FOR EACH ROW]**: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise, the trigger will execute just once when the SQL statement is executed, which is called a **table level trigger**.
- **WHEN (<condition>)**: This provides a condition for rows for which the trigger would fire. This clause is valid only for **row level triggers**.

Triggers Program

Exp 1. To write a trigger that inserts or updates values of e_name and job as uppercase strings even if we give lowercase strings.

```
CREATE OR REPLACE TRIGGER "EMPT1"
BEFORE
insert or update on "EMPLOYEE"
for each row
begin
        :new.e_name:=upper(:new.e_name);
        :new.job:=upper(:new.job);
end;
```

Trigger Created

Sql> Select * from employee;

E_ID	E_NAME	JOB	MANAGER	HIREDATE	SALARY	COMM	D_ID
7369	Smith	Clerk	7902	17-DEC-80	2000	-	20
7499	Allen	Salesman	7698	20-FEB-81	1600	300	30
7521	Ward	Salesman	7698	22-FEB-81	1250	500	30
7566	Janes	Manager	7839	02-APR-81	2975	-	20
7654	Martin	Salesman	7698	28-SEP-81	1437.5	1400	30

sql> Update employee set e_name='Anil' where e_id = 7521;

1 row(s) updated.

Sql> Select * from employee;

E_ID	E_NAME	JOB	MANAGER	HIREDATE	SALARY	COMM	D_ID
7369	Smith	Clerk	7902	17-DEC-80	2000	-	20
7499	Allen	Salesman	7698	20-FEB-81	1600	300	30
7521	ANIL	SALESMAN	7698	22-FEB-81	1250	500	30
7566	Janes	Manager	7839	02-APR-81	2975	-	20
7654	Martin	Salesman	7698	28-SEP-81	1437.5	1400	30

sql> Insert into employee(e_id, e_name, job) values (7000, 'Suman', 'Manager');

1 row(s) inserted.

Sql> Select * from employee;

E_ID	E_NAME	JOB	MANAGER	HIREDATE	SALARY	COMM	D_ID
7000	SUMAN	MANAGER	-	-	-	-	-
7369	Smith	Clerk	7902	17-DEC-80	2000	-	20
7499	Allen	Salesman	7698	20-FEB-81	1600	300	30
7521	ANIL	SALESMAN	7698	22-FEB-81	1250	500	30

Exp 2. To write a TRIGGER to ensure that DEPTARTMENT TABLE does not contain duplicate or null values in DEPARTMENT NUMBER (D_ID) column.

```
CREATE OR REPLACE TRIGGER "DEPT_T1"
BEFORE
insert on "DEPARTMENT"
for each row
declare
 a number;
 begin
  if(:new.d_id is Null) then
   raise_application_error(-20001,'error::department number(d_id) cannot be null');
  else
   select count(*) into a from department where d_id=:new.d_id;
   if(a=1) then
    raise_application_error(-20002,'error:: cannot have duplicate deptno');
   end if;
  end if;
 end;
```

Trigger created.

Sql>select * from department;

D_ID	D_NAME	LOCATION
10	Accounting	New York
20	Research	Dallas
30	Sales	Chicago
40	Operations	Boston

Sql>Insert into department(d_id, dname) values (10, ‘Marketing’);

ORA-20002: error:: cannot have duplicate deptno
ORA-06512: at "SYSTEM.DEPT_T1", line 9
ORA-04088: error during execution of trigger 'SYSTEM.DEPT_T1'

Sql> Insert into department(d_name, location) values ('Marketing', 'California');

ORA-20001: error::department number(d_id) cannot be null
ORA-06512: at "SYSTEM.DEPT_T1", line 5
ORA-04088: error during execution of trigger 'SYSTEM.DEPT_T1'

Exp3. To write a trigger that doesn't allow a salary to be updated if the employee commission is null.

```
create or replace trigger "empupdatesal" before
update of salary on employee
for each row
begin
        if :old.comm is null then
        raise_application_error(-20100,'commission is null, salary cannot be updated');
        end if;
end;
```

Trigger created.

Sql> Select * from employee;

E_ID	E_NAME	JOB	MANAGER	HIREDATE	SALARY	COMM	D_ID
7369	Smith	Clerk	7902	17-DEC-80	2000	-	20
7499	Allen	Salesman	7698	20-FEB-81	1600	300	30
7521	ANIL	SALESMAN	7698	22-FEB-81	1250	500	30
7566	Janes	Manager	7839	02-APR-81	2975	-	20
7654	Martin	Salesman	7698	28-SEP-81	1437.5	1400	30
7698	Blake	Manager	7839	01-MAY-81	2850	-	30
7782	Clark	Manager	7839	09-JUN-81	2450	-	10
7788	Scott	Analyst	7566	09-DEC-82	3000	-	20
7839	King	President	-	17-NOV-81	5000	-	10
7844	Turner	Salesman	7698	08-SEP-81	1725	-	30

Sql>update employee set salary=salary+1000 where e_id=7369;

ORA-20100: commission is null, salary cannot be updated
ORA-06512: at "SYSTEM.empupdatesal", line 3
ORA-04088: error during execution of trigger 'SYSTEM.empupdatesal'
1. update employee set salary=salary+1000 where e_id=7369;

Sql>update employee set salary=salary+1000 where e_id=7499;

1 row(s) updated.

E_ID	E_NAME	JOB	MANAGER	HIREDATE	SALARY	COMM	D_ID
7369	Smith	Clerk	7902	17-DEC-80	2000	-	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	2600	300	30
7521	ANIL	SALESMAN	7698	22-FEB-81	1250	500	30
7566	Janes	Manager	7839	02-APR-81	2975	-	20
7654	Martin	Salesman	7698	28-SEP-81	1437.5	1400	30

If you observe EMPT1 trigger also executes because of update operation on employee table where employee name and job will be change into uppercase.

PL/SQL Cursor

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by **Cursor**. A cursor contains information on a select statement and the rows of data accessed by it.

A **cursor** in SQL is a database object used to **retrieve, process, and manipulate data one row at a time**.

A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

1. Implicit Cursors
2. Explicit Cursors

1. PL/SQL Implicit Cursors

The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don't use an explicit cursor for the statement.

These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.

Attribute and its Description

- %FOUND: True if the SQL operation affects at least one row.
- %NOTFOUND: True if no rows are affected.
- %ROWCOUNT: Returns the number of rows affected, always returns FALSE for implicit cursors
- %ISOPEN: Checks if the cursor is open.

Ex1. Update the salary of each employee by 1500 and check how many roes effected using implicit cursors.

```
DECLARE
        total_rows number;
BEGIN
         UPDATE Employee
         SET Salary = Salary + 1500 where comm is not NULL;
      IF sql%notfound THEN
            dbms_output.put_line('no employee salary updated');
      ELSIF sql%found THEN
            total_rows := SQL%ROWCOUNT;
            dbms_output.put_line(total_rows || ' rows updated.');
      END IF;
END;
```

Output:
3 rows updated.

Note:-

In below program, a trigger has already been created where salary can not be increased if comm is NULL, that's why it is giving above error.

```
DECLARE
        total_rows number;
BEGIN
        UPDATE Employee
        SET Salary = Salary + 1500;
        IF sql%notfound THEN
                dbms_output.put_line('no employee salary updated');
        ELSIF sql%found THEN
                total_rows := SQL%ROWCOUNT;
                dbms_output.put_line(total_rows || ' rows updated.');
        END IF;
END;
```

Output:

ORA-20100: commission is null, salary cannot be updated
ORA-06512: at "SYSTEM.empupdatesal", line 3
ORA-04088: error during execution of trigger 'SYSTEM.empupdatesal'

```
1. DECLARE
2.          total_rows number;
3. BEGIN
4.       UPDATE Employee
5.                SET Salary = Salary + 1500;
```

2. PL/SQL Explicit Cursors

- The Explicit cursors are defined by the programmers to gain more control over the context area.
- These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Syntax of explicit cursor

CURSOR cursor_name **IS** select_statement;;

Steps to follow while working with an explicit cursor.:

1. Declare the cursor to initialize in the memory.
2. Open the cursor to allocate memory.
3. Fetch the cursor to retrieve data.
4. Close the cursor to release allocated memory.

Ex. Write a program to print employee number, employee name, salary, department number of all employees from employee table using cursors.

```
DECLARE
        -- STEP1: Declare the cursor to initialize in the memory
        CURSOR emp_cursor is select e_id, e_name, d_id, salary from employee;
        i employee.e_id%type;
        j employee.e_name%type;
        k employee.d_id %type;
        l employee.salary%type;
BEGIN
        --STEP 2: Open the cursor
        open emp_cursor;
        dbms_output.put_line('Empno,    name,    deptno,    salary of employees are := ');
        loop
                --STEP 3: Fetch rows from the cursor
                fetch emp_cursor into i, j, k, l;
                exit when emp_cursor%notfound;
                dbms_output.put_line(i||'    '||j||'    '||k||'    '||l);
        end loop;
        -- STEP 4:  Close the cursor
        close emp_cursor;
END;
```

Output:

```
Empno,    name,    deptno,    salary of employees are :=
7369    Smith    20    2000
7499    ALLEN    30    5600
7521    ANIL    30    4250
7566    Janes    20    2975
7654    MARTIN    30    4437.5
7698    Blake    30    2850
7782    Clark    10    2450
7788    Scott    20    3000
7839    King    10    5000
7844    Turner    30    1725
7876    Adams    20    1410
7900    James    30    950
7902    Ford    20    3000
7934    Miller    10    1300
8000    SUMAN
7000    SUMAN

Statement processed.
```