Lab 9 - Trigger

A trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

Syntax for Creating a Trigger

```
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)

BEGIN

 --- sql statements

END;
```

- *CREATE [OR REPLACE] TRIGGER trigger_name* - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
- *{BEFORE | AFTER | INSTEAD OF }* - This clause indicates at what time should the trigger get fired. i.e for example: before or after updating a table. INSTEAD OF is used to create a trigger on a view. before and after cannot be used to create a trigger on a view.
- *{INSERT [OR] | UPDATE [OR] | DELETE}* - This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.
- *[OF col_name]* - This clause is used with update triggers. This clause is used when you want to trigger an event only when a specific column is updated.
- *CREATE [OR REPLACE] TRIGGER trigger_name* - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
- *[ON table_name]* - This clause identifies the name of the table or view to which the trigger is associated.

- *[REFERENCING OLD AS o NEW AS n]* - This clause is used to reference the old and new values of the data being changed. By default, you reference the values as :old.column_name or :new.column_name. The reference names can also be changed from old (or new) to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.
- *[FOR EACH ROW]* - This clause is used to determine whether a trigger must fire when each row gets affected (i.e. a Row Level Trigger) or just once when the entire sql statement is executed(i.e.statement level Trigger).
- *WHEN (condition)* - This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

Example 1 :

CREATE TABLE:

CREATE TABLE EMPLOYEE (EMPLOYEE_NAME VARCHAR2(20),CITY VARCHAR2(10));

CREATE TRIGGER TRIGGER CODE:

```
create or replace TRIGGER Mytrigger
  BEFORE
    INSERT OR
    UPDATE OF employee_name, city OR
    DELETE
  ON employee
BEGIN
  CASE
    WHEN INSERTING THEN
      DBMS_OUTPUT.PUT_LINE('Inserting');
    WHEN UPDATING('CITY') THEN
      DBMS_OUTPUT.PUT_LINE('Updating CITY');
    WHEN UPDATING('employee_name') THEN
      DBMS_OUTPUT.PUT_LINE('Updating Employee_name');
    WHEN DELETING THEN
      DBMS_OUTPUT.PUT_LINE('Deleting');
  END CASE;
END;
```

PERFORM INSERT/UPDATE/DELETE:

set serveroutput on; /* dbms_output.put_line will display output on server. So, to view it on client side use this option. */
Now when you perform insert, update or delete , appropriate PUT_LINE message will be displayed.

Example 2: **The price of a product changes constantly. It is important to maintain the history of the prices of the products.**

We can create a trigger to update the 'product_price_history' table when the price of the product is updated in the 'product' table.

1) Create the 'product' table and 'product_price_history' table

```
CREATE TABLE product_price_history
(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2) );

CREATE TABLE product
(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2) );
```

2) Create the price_history_trigger and execute it.

```
CREATE or REPLACE TRIGGER price_history_trigger
BEFORE UPDATE OF unit_price
ON product
FOR EACH ROW
BEGIN
INSERT INTO product_price_history
VALUES
(:old.product_id,
 :old.product_name,
 :old.supplier_name,
 :old.unit_price);
END;
```

3) Lets update the price of a product.

UPDATE PRODUCT SET unit_price = 800 WHERE product_id = 100

Once the above update query is executed, the trigger fires and updates the 'product_price_history' table.

4)If you ROLLBACK the transaction before committing to the database, the data inserted to the table is also rolled back.

Example 3:

create following CUSTOMERS table

Select * from customers;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 |  Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 |  Delhi     |  1500.00 |
|  3 | kaushik  |  23 |  Kota      |  2000.00 |
|  4 | Chaitali |  25 |  Mumbai    |  6500.00 |
|  5 | Hardik   |  27 |  Bhopal    |  8500.00 |
|  6 | Komal    |  22 |  MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

The following program creates a **row-level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values −

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_diff number;
BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

Now try Following SQL Queries.
1) INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00);
2) UPDATE customers SET salary = salary + 500 WHERE id = 2;

Example 4:

Create Following Trigger:

```
/--drop trigger tr_smanagewhenjoined;

create or replace trigger tr_smanagewhenjoined
BEFORE
insert or update on "SECURITYMAN"
for each row
begin
  if inserting then
    :NEW.joining_date := sysdate;
    :NEW.agewhenjoined := floor(months_between(sysdate, :NEW.dob)/12);
 end if;

 if updating then
  :NEW.agewhenjoined := floor(months_between(:OLD.joining_date, :NEW.dob)/12);
 end if;

end;
```

Save this file as create_trigger.sql. Don't Compile.

In other sheet try Following SQL Queries:

```
select floor(months_between(sysdate, to_date('01-MAR-1988','DD-MON-YYYY'))/12) from dual;

drop table securityman;

create table securityman
(id number, name varchar2(20), dob date, agewhenjoined number, joining_date date);

select * from securityman;

@create_trigger.sql;

insert into securityman (id, name, dob) values (1,'James','01-JAN-1997');

insert into securityman (id, name, dob) values (2,'Karen','01-MAR-1990');

insert into securityman  (id, name, dob) values (3,'Naic','25-APR-1995');

update securityman set dob=to_date('15-AUG-1993','DD-MON-YYYY') where id=2;
```