Triggers

- A **trigger** is a statement that is executed automatically by the system as a side effect of a specified changes occur to the DBMS (modification to the database).
- Three parts:
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run) [Optional]
  - Action (what happens if the trigger runs)
- To design a trigger mechanism, we must:
  - Specify the conditions under which the trigger is to be executed.
  - Specify the actions to be taken when the trigger executes.
- Database stores triggers. Database system executes it whenever specified event occurs and corresponding condition is satisfied.

Triggers are in fact, written to be executed in response to any of the following events:

1. A database manipulation (DML) statement
   (DELETE, INSERT, or UPDATE).
2 A database definition (DDL) statement
   (CREATE, ALTER, or DROP).
3. A database operation
   (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).
4. Triggers could be defined on the table, view, schema, or database with which the event is associated.

- Benefits of Triggers
  - Generating some derived column values automatically
  - Enforcing referential integrity
  - Event logging and storing information on table access
  - Imposing security authorizations
  - Preventing invalid transactions

Trigger Example

- Suppose that instead of allowing negative account balances, the bank deals with overdrafts by
  - setting the account balance to zero
  - creating a loan in the amount of the overdraft
  - giving this loan a loan number identical to the account number of the overdrawn account
- The condition for executing the trigger is an update to the *account* relation that results in a negative *balance* value.

Syntax :

CREATE [OR REPLACE] TRIGGER <trigger_name>
{BEFORE | AFTER} {INSERT | DELETE | UPDATE} ON
<table_name> | <view_name>
[REFERENCING [NEW AS <new_row_name>] [OLD
AS <old_row_name>]]
[FOR EACH ROW [WHEN <trigger_condition>)]]

<trigger_body>

- CREATE or REPLACE TRIGGER trigger_name: Creates a trigger with the given name otherwise overwrites an existing trigger with the same name.
- {BEFORE , AFTER }: Indicates the where should trigger get fired. BEFORE trigger execute before when statement execute before time or AFTER trigger execute after when statement execute after time.
- {INSERT , UPDATE , DELETE}: Determines the performing trigger event. More than one triggering events allow can be used together separated by OR keyword.
  ON Table Name: Determine the perform trigger event in selected Table.

- [Referencing {old AS old, new AS new}]:

  Reference the old and new values of the data being changed.

  :old use to existing row perform and

  :new use to execute new row to perform.

  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: Trigger must fire when each row gets Affected (Row Level Trigger) or just once when the entire SQL statement is executed (statement level Trigger).
- WHEN (condition): Valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

Triggering Events and Actions in SQL

- Triggering event can be **insert**, **delete** or **update**
- Triggers on update can be restricted to specific attributes
  - **E.g. create trigger** *overdraft-trigger* **after update of** *balance* **on** *account*
- Values of attributes before and after an update can be referenced
  - **referencing old row as** : for deletes and updates
  - **referencing new row as** **:** for inserts and updates

Trigger example

```
CREATE OR REPLACE TRIGGER emp_alert_trig
   BEFORE INSERT ON emp
BEGIN
   DBMS_OUTPUT.PUT_LINE('New employees are
   about to be added');
END;
/
```

Trigger Example

T4 (a INTEGER, b CHAR(10));

T5 (c CHAR(10), d INTEGER);

– if integer value in less than 10 then whole row is inserted into T5 table also

```
CREATE TRIGGER trig1
AFTER INSERT ON T4
    REFERENCING NEW AS newRow
FOR EACH ROW     WHEN (newRow.a <= 10)
  BEGIN
      INSERT INTO T5
          VALUES(:newRow.b, :newRow.a);
END trig1;
```

Trigger Example

```
Create trigger Total_sal
  AFTER DELETE on emp
  FOR EACH ROW
       Update Dept set total_sal = total_sal – :OLD.salary
      where Dept.Dno = :OLD.Dno;


CREATE OR REPLACE TRIGGER employee_insert_update
       BEFORE INSERT OR UPDATE ON employee
        FOR EACH ROW
    DECLARE
       dup_flag  INTEGER;
    BEGIN
       :NEW.first_name := UPPER(:NEW.first_name);
    END;
```

– Triggers can be activated **before** an event, which can serve as extra constraints. E.g. convert blanks to null.

```
create trigger setnull-trigger before update on r
referencing new row as nrow
for each row
    when nrow.phone-number = ' '
     set :nrow.phone-number = null   // body part
```

Statement Level Triggers

- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction
  - Use **for each statement** instead of **for each row**
  - Use **referencing old table** or **referencing new table** to refer to temporary tables (called ***transition tables***) containing the affected rows
  - Can be more efficient when dealing with SQL statements that update a large number of rows

- Row Level Trigger
  - Row Level Trigger is fired each time row is affected by Insert, Update or Delete command. If statement doesn't affect any row, no trigger action happens.
- Statement Level Trigger
  - This kind of trigger activate once per SQL statement
  - The trigger activates and performs its activity irrespective of number of rows affected due to SQL statement.