## Triggers

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

Trigger is a database object. It will be permantaly saved in database. Trigger is also same as stored procedures and also it will automatically invoked whenever DML operation performed against table or view.

Trigger is invoked before the data entered as record in the table, this data taken by trigger. This is responsibility of trigger.

***Trigger is used for mainly two purposes (from developer view):***

- To maintain business data into uniformly case because submitted data from the user in any case upper or lower.
- To maintain audit information of any table data. Audit information means monitoring table data what will happen on data.

Database triggers can be used to:

- Audit data modifications
- Log events transparently
- Enforce complex business rules
- Derive column values automatically
- Implement complex security authorizations
- Maintain replicate tables

### Trigger Classification

Triggers can be classified based on the following parameters.

***Classification based on the timing***

- *BEFORE Trigger:* It fires before the specified event has occurred.
- *AFTER Trigger:* It fires after the specified event has occurred.

***Classification based on the level***

- *STATEMENT level Trigger:* It fires one time for the specified event statement.
- *ROW level Trigger:* It fires for each record that got affected in the specified event. (only for DML)

***Classification based on the Event***

- *DML Trigger:* It fires when the DML event is specified (INSERT/UPDATE/DELETE)
- *DDL Trigger:* It fires when the DDL event is specified (CREATE/ALTER)
- *DATABASE Trigger:* It fires when the database event is specified (LOGON/LOGOFF/STARTUP/SHUTDOWN)

So each trigger is the combination of above parameters.

***A trigger has three basic parts:-***

- trigger event or statement
- trigger restriction
- trigger action

- **Trigger event or statement:** A trigger event can be an insert, update or delete statement for a specific table.
- **Trigger Restriction:** It controls the trigger executions before or after event. Trigger executed.
- **Trigger Action:** it implements or represent the logic or functionality of trigger. i.e. what trigger doing.

**Keywords:**

- **: new-** it will get new value from column. New is associated with insert as well as update operation.
  Syntax- :new.column_name
  Example- :new.ename
- **:old -** it will get old value from column. old is associated with delete & before update operation.
  Syntax- :old.column_name
  Example- :old.sal

***There are two types of triggers supported by pl/sql***

- **Row/Record Level:** trigger body executed for each row for a DML operations. In row level trigger, a trigger body is executed for each row for a DML operations, that why we are using for each row clause in trigger specification and also data internally stored in 2 rollback segment qualifiers these are new, old. These qualifiers are used in either trigger specification or body.

- **Statement Level:** trigger body executed only one time for a DML operations. It doesn't worry about how records are affected by DML.

**Type of Triggers**

- BEFORE Trigger: BEFORE trigger execute before the triggering DML statement (INSERT, UPDATE, DELETE) execute. Triggering SQL statement is may or may not execute, depending on the BEFORE trigger conditions block.
- AFTER Trigger: AFTER trigger execute after the triggering DML statement (INSERT, UPDATE, DELETE) executed. Triggering SQL statement is executed as soon as followed by the code of trigger before performing Database operation.
- ROW Trigger: ROW triggers fire for each and every record which is performing INSERT, UPDATE, DELETE from the database table. If row deleting is define as trigger event, when trigger file, deletes the five rows each times from the table.
- Statement Trigger: Statement trigger fire only once for each statement. If row deleting is define as trigger event, when trigger file, deletes the five rows at once from the table.

Combination Trigger: Combination trigger are combination of two trigger type,

- Before Statement Trigger: Trigger fire only once for each statement before the triggering DML statement.
- Before Row Trigger: Trigger fire for each and every record before the triggering DML statement.
- After Statement Trigger: Trigger fire only once for each statement after the triggering DML statement executing.
- After Row Trigger: Trigger fire for each and every record after the triggering DML statement executing.

**PL/SQL Triggers Syntax**

```
CREATE [OR REPLACE] TRIGGER trigger_name
        BEFORE | AFTER
        [INSERT, UPDATE, DELETE [COLUMN NAME..]
        ON table_name
        Referencing [ OLD AS OLD | NEW AS NEW ]
        FOR EACH ROW | FOR EACH STATEMENT [ WHEN Condition ]
DECLARE
  [declaration_section
          variable declarations;
```

```
        constant declarations;    ]
BEGIN
      [executable_section
            PL/SQL execute/subprogram body    ]
EXCEPTION
  [exception_section
            PL/SQL Exception block    ]
END;
```

- CREATE [OR REPLACE] TRIGGER trigger_name: Create a trigger with the given name. If already have overwrite the existing trigger with defined same name.
- BEFORE | AFTER: Indicates when the trigger gets fire. BEFORE trigger execute before when statement execute before. AFTER trigger execute after the statement execute.
- [INSERT, UPDATE, DELETE [COLUMN NAME..] : Determines the performing trigger event. You can define more then one triggering event separated by OR keyword.
- ON table_name : Define the table name to performing trigger event.
- Referencing [OLD AS OLD | NEW AS NEW]: Give referencing to a old and new values of the data. :old means use existing row to perform event and :new means use executing new row to perform event. You can set referencing names user define name from old (or new).
  You can't referencing old values when inserting a record, or new values when deleting a record, because It's does not exist.
- FOR EACH ROW | FOR EACH STATEMENT: Trigger must fire when each row gets Affected (ROW Trigger). and fire only once when the entire sql statement is execute (STATEMENT Trigger).
- WHEN Condition: Optional. Use only for row level trigger. Trigger fire when specified condition is satisfy.

***Trigger for inserting name of employee in upper case***

```
create or replace trigger upper_case
before
 insert  on emp
for each row
begin
  :new.name:=upper(:new.name);
end;
```

***Trigger which not allow to delete 1 deptid in department***

```
create or replace trigger del_dept
 before delete on dept
 for each row
  begin
     if:old.dept_id=1 then
      raise_application_error(-20002,'Dept 40 is not allowed to delete');
    end if;
   end;
```

***Trigger which not allow to  insert or update 10 deptid in department***

```
create or replace trigger checkdept
before insert or update on dept
for each row
begin
```

```
  if:new.dept_id<10 then
    raise_application_error(-20001,'Dept is is not allowed less than  10');
   end if;
end;
```

***Trigger which not allow to update employee name in employee table***

```
create or replace trigger up_name
 before
 update of name on emp
 for each row
 begin
   raise_application_error(-20003,'Name can not be updated');
 end;
```

**Statement Level Trigger**

A statement-level trigger is fired whenever a trigger event occurs on a table regardless of how many rows are affected. In other words, a statement-level trigger executes once for each transaction.For example, if you update 1000 rows in a table, then a statement-level trigger on that table would only be executed once.

Due to its features, a statement-level trigger is not often used for data-related activities like auditing the data changes in the associated table. It's typically used to enforce extra security measures on the kind of transaction that may be performed on a table.

By default, the statement CREATE TRIGGER creates a statement-level trigger when you omit the FOR EACH ROW clause.

***Trigger which not allow to update employee salary at month end***

```
create or replace trigger emp_salary
  after update of salary on emp
declare
   n number;
 begin
 n:=extract(day from sysdate);
    if n between 28 and 31 then
      raise_application_error(-20100,'Can not update salary of employee');
    end if;
 end;
```

***Trigger which not allow to insert Negative employee number.***

```
create or replace trigger emp_stat
before  insert on emp
  begin
  if :new.empid <0 then
    raise_application_error(-20005,'Negatvier employee is not allowed');
 end if;
  end;
```

**DDL Trigger**

DDL triggers are the same as other triggers. The main difference is the event for which the trigger will execute (DDL statements) and perform the action described in the trigger body. We can use DDL triggers to get tasks done in response to a DDL event, like preventing an unauthorized change to the database, executing any specific action, or recording modifications completed in the database. In this tip, we will be working on the later action, and we will also use the database mail profile to send an email alert whenever a DDL statement is executed.

**Major Steps**

Below are the major steps we will perform to complete this task:

- Create a new table that will hold the audit data.
- Create a DDL trigger that will execute in the event of DDL statements.
- Execute the Alter statement on the database and table to test the newly created trigger code.

1) create table ddl _opt for DDL trigger

```
create table ddl_opt
  (
  ddl_date date,
  ddl_user varchar2(15),
  ddl_created varchar2(15),
  object_name varchar2(15),
 ddl_operation varchar2(15)
  );
```

2) create trigger

```
      create or replace trigger ddl_trigger
        after ddl on SCHEMA
    begin
          insert into ddl_opt values(
            sysdate ,
            sys_context('USERENV','current_user'),
            ora_dict_obj_type,
           ora_dict_obj_name,
           ora_sysevent);
   end;
```

3) peform create/alter/drop  opeatration on database then check ddl_opt table

***// DDL trigger for only for create***

```
create or replace trigger create_trigger1
after create on Database
 begin
  insert into ddl_opt values(
sysdate,
sys_context('USERENV','CURRENT_USER'),
'table',
ora_dict_obj_name,
'Create');
end;
create or replace view vw_std
 as
select * from std;
```

***//DDl  trigger for only drop***

```
create or replace trigger drop_trigger
after drop on Database
 begin
  insert into ddl_opt values(
sysdate,
sys_context('USERENV','CURRENT_USER'),
'table',
ora_dict_obj_name,
'drop');
```

```
end;
```

**Trigger Auditing**

In Oracle, trigger auditing involves using database triggers to automatically record and track changes made to tables, such as INSERT, UPDATE, and DELETE operations, along with the user who made the change and the timestamp.

Audit triggers are a type of database trigger that automatically execute a PL/SQL block when a specific event (like a DML operation) occurs on a table.

**Purpose**:

The primary purpose is to track changes to data, enabling auditing and accountability.

**How it works:**

1. An audit trigger is defined to monitor a specific table and event (e.g., BEFORE INSERT OR UPDATE ON my_table).
2. When the event occurs, the trigger's PL/SQL block executes.
3. The PL/SQL block typically inserts the audit information (old/new values, user, timestamp) into a separate audit table.

**Benefits:**

- Data Integrity: Helps ensure data accuracy and consistency by tracking changes.
- Security: Provides a record of who made changes and when, aiding in security investigations.
- Compliance: Facilitates compliance with regulatory requirements that mandate data change tracking.

Example:

```
create table log_table1(
log_id number primary key,
log_date date,
log_object_type varchar2(15));
create sequence my start with 1 increment by 1
```

***Trigger audit for std table performing (Insert Or Delete Or Update)***

```
create or replace trigger trig_audit
after INSERT OR DELETE OR UPDATE ON std
for each row
declare
  a varchar2(15);
begin
   if INSERTING THEN
     a:='INSERT';
 elsif DELETING THEN
    a:='DELETE';
elsif UPDATING THEN
    a:='UPDATE';
 end if;
  insert into log_table1 values(
   my1.nextval,
   systimestamp,
    a);
end;
```