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.

SYNTAX:

```
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;
```

Each statements are described below :

Statement	Description
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 the trigger should 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:

```
Create or replace Trigger np before insert on account for each row
Begin
  IF :NEW.bal < 0  THEN
    DBMS_OUTPUT.PUT_LINE('BALANCE IS NAGATIVE..');
  END IF;
End;
/
```

Output:

Run SQL Command Line

```
SQL>start D://t.sql
Trigger created.

SQL>insert into account values(1,-100);
BALANCE IS NAGATIVE..
1 row created.
```

Types of Triggers

A trigger's type is defined by the type of triggering transaction and by the level at which the trigger is executed. Oracle has the following types of triggers depending on the different applications.

Row Level Triggers

Statement Level Triggers

Before Triggers

After Triggers

Row Level Triggers :

Row level triggers execute once for each row in a transaction.
The commands of row level triggers are executed on all rows that are affected by the command that enables the trigger.

For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement.
If the triggering statement affects no rows, the trigger is not executed at all.
Row level triggers are created using the FOR EACH ROW clause in the CREATE TRIGGER command.

Application:

Consider a case where our requirement is to prevent updati on of empno 100 record cannot be updated, then whenever UPDATE statement update records, there must be PL/SQL block that will be fired automatically by UPDATE statement to check that it must not be 100, so we have to use Row level Triggers for that type of applications.

Statement Level Triggers :

Statement level triggers are triggered only once for each transaction.
For example when an UPDATE command update 15 rows, the commands contained in the trigger are executed only once, and not with every processed row.
Statement level trigger are the default types of trigger created via the CREATE TRIGGER command.

Application:

Consider a case where our requirement is to prevent the DELETE operation during Sunday. For this whenever DELETE statement deletes records, there must be PL/SQL block that will be fired only once by DELETE statement to check that day must not be Sunday by referencing system date, so we have to use Statement level Trigger for which fires only once for above application.

Before Trigger :

Since triggers are executed by events, they may be set to occur immediately before or after those events.
When a trigger is defined, you can specify whether the trigger must occur before or after the triggering event i.e. INSERT, UPDATE, or DELETE commands.

BEFORE trigger execute the trigger action before the triggering statement.
These types of triggers are commonly used in the following situation:

1.BEFORE triggers are used when the trigger action should determine whether or not the triggering statement should be allowed to complete.

2.By using a BEFORE trigger, you can eliminate unnecessary processing of the triggering statement.

For example: To prevent deletion on Sunday, for this we have to use Statement level before trigger on DELETE statement.

3.BEFORE triggers are used to derive specific column values before completing a triggering INSERT or UPDATE statement.

After Trigger :

AFTER trigger executes the trigger action after the triggering statement is executed.
AFTER triggers are used when you want the triggering statement to complete before executing the trigger action.
For example: To perform cascade delete operation, it means that user delete the record fro one table, but the corresponding records in other tables are delete automatically by a trigger which fired after the execution of delete statement issued by the user.
When combining the different types of triggering actions, there are mainly 4 possible Valid trigger types available to us.

The possible configurations are:
BEFORE statement Trigger
BEFORE row Trigger
AFTER statement Trigger
AFTER row Trigger

Dropping Triggers :

Triggers may be dropped via the drop trigger command. In order to drop a trigger, you must either own the trigger or have the DROP ANY TRIGGER system privilege.

Syntax:

```
DROP TRIGGER trigger_name;
```

Example:

Run SQL Command Line

```
SQL>DROP TRIGGER emp;

Trigger dropped.
```

Advantages of Triggers

1. Triggers provide a way to check the integrity of the data. When there is a change in the database the triggers can adjust the entire database.
2. Triggers help in keeking User Interface lightweight. Instead of putting the same function call all over the application you can put a trigger and it will be executed.

Disadvantages of Triggers

1.Triggers may be difficult to troubleshoot as they execute automatically in the database. If there is some error then it is hard to find the logic of trigger because they are fired before or after updates/inserts happen.

2. The triggers may increase the overhead of the database as they are executed every time any field is updated.