Triggers

* It is a stored PL/SQL program unit associated with a specific database table
* Oracle executes (fires) the trigger automatically whenever a given SQL operation affects the table
* They are invoked implicitly
* They are useful for customizing a database
* They should be used only when necessary
* Can automatically generate derived column values
* Can prevent invalid transactions
* Can enforce complex security authorizations
* Can enforce referential integrity across nodes in a distributed database
* Can enforce complex business rules
* Can provide transparent event logging
* Can provide sophisticated auditing
* Can maintain synchronous table replicates
* Can gather statistics on table access
* Can derive column values automatically
* Can restrict DML operations to regular business hours

**Syntax --**

CREATE [OR REPLACE] TRIGGER <triggername>

BEFORE|AFTER

INSERT|DELETE|UPDATE OF <columnnames> ON <tablename>

[FOR EACH ROW]

WHEN (<condition>)

<PL/SQL Block>

Name in the ON clause identifies the database table associated with the trigger

The trigger event specifies the SQL DML statement (INSERT, DELETE or UPDATE) that affects the table

AFTER specifies that the trigger fires after the manipulation is done

* BEFORE specifies that the trigger fires before the manipulation is done
* By default, a trigger fires once per table
* FOR EACH ROW specifies that the trigger fires once per row
* For the trigger to fire, the Boolean expression in the WHEN clause must evaluate to TRUE
* REPLACE can be added to the CREATE statement to drop and re-create the trigger automatically

CREATE TRIGGER flight\_update

AFTER INSERT ON reservation

FOR EACH ROW

BEGIN

IF :new.class = ‘F’ THEN

statements;

ELSIF :new.class = ‘B’ THEN

statements;

END IF;

END;

* Prefix :new is a correlation name that refers to the newly updated column value
* Within a trigger, the :new and :old values of changing rows can be referenced
* A single trigger can handle more than one operation
* Use conditional predicates to identify the type of statement used to invoke the section of code

**Examples of Triggers -**

**Row Level**

**AFTER clause**

-- Main table

create table temp

as

select \* from emp;

-- Table to transfer inserted record

create table instemp

as

select ename,sal

from emp

where 1=2;

**Whenever a row is inserted in the temp table then that new row should be transferred in the instemp table**

Create or replace trigger trig1

After **INSERT** on temp

**For Each Row**

Begin

Insert into InsTemp

Values**(:new.**ename, **:new**.sal);

Dbms\_output.put\_line('Record inserted');

End;

**Table to transfer deleted record**

create table deltemp

as

select ename,sal

from emp

where 1=2;

**--Whenever a row is deleted from temp table then that row**

**should be transferred in Deltemp table**

Create or replace trigger Trig2

After **DELETE** on Temp

For Each Row

Begin

Insert into Deltemp

Values(**:old.**ename, **:old.**sal);

End;

**Table to transfer the old record before updations**

create table salary\_revision

(empid integer,

Empname varchar(30),

**Oldsalary number**,

**Newsalary number**);

Whenever salary is updated from the temp table then the previous

and the changed salary of that employee should go in table Salary\_Revision

create or replace trigger trig3

After **UPDATE** on temp

For Each Row

Begin

Insert into Salary\_Revision

Values(:old.empno, :old.ename, **:old.sal**, **:new.sal**);

End;

**To know the triggers based on a table:**

In **SQL PLUS** Editor only execute the following 2 commands:

col TRIGGER\_NAME format a5

col TRIGGERING\_EVENT format a7

**select trigger\_name, trigger\_type, triggering\_event**

**from user\_triggers**

**where lower(table\_name) = 'temp';**

* To disable all triggers of a table:

**Alter Table Temp**

**Disable All Triggers;**

* To enable all triggers of a table:

**Alter Table Temp**

**Enable All Triggers;**

* To disable one particular trigger:

**Alter Trigger Trig2 Disable;**

* To enable one particular trigger:

**Alter Trigger Trig2 Enable;**

**Use Of Before Clause**

**When a new record is inserted then it should have sal >= 5000**

Create or replace trigger trig4

Before INSERT on temp

For Each Row

Begin

If :new.sal < 5000 then

raise\_application\_error(-20009,'New record sal should be above 5000');

End If;

End;

**Statement OR Table Level Triggers**

**-- No transaction should take place on FRIDAY**

create or replace trigger trig5

**Before INSERT or UPDATE or DELETE**

on temp

-- There is no **For Each Row**, that means it is Statement OR Table Level

Begin

If Rtrim(to\_char(Sysdate,'DAY')) = 'FRIDAY' Then

raise\_application\_error(-20005,'No transactions on Friday');

End If;

End;

**Combining DML events in one Trigger**

Any DML on the table emp500 should be traced in the table KeepTrace

Create table emp500

As

Select \* from emp;

create table KeepTrace

(username varchar2(30),

Event\_Name varchar2(30),

Event\_Date date,

Time\_of\_Event varchar2(40));

Create or replace trigger trig6

After INSERT or DELETE or UPDATE

On emp500

For each row

Begin

If **Inserting** Then

Insert into KeepTrace

Values(user, **'Record is Inserted'**, Sysdate, to\_char(sysdate,'hh:mi:ss'));

ElsIf **Deleting** Then

Insert into KeepTrace

Values(user, **'Record is Deleted'**,Sysdate, to\_char(sysdate,'hh:mi:ss'));

ElsIf **Updating** Then

Insert into KeepTrace

Values(user,**'Record is Updated'**,Sysdate, to\_char(sysdate,'hh:mi:ss'));

End If;

End;

Note 🡪 **Inserting**, **Deleting** and **Updating** keywords are known as **“Conditional Predicates of Triggers.”**

### Calling a procedure inside a trigger

create table EmpSummary

(Total\_Salary number);

Procedure Code –

create or replace procedure TotSal

is

vsal number;

Begin

delete from EmpSummary;

Select sum(sal) into vsal

from emp;

Insert into EmpSummary

Values(vsal);

End;

create or replace trigger EmpUpdates

## After insert or update or delete on emp

**CALL TOTSAL**

Note: The Above trigger is Bodyless Trigger.

**Or if the trigger has it’s code then –**

create or replace trigger EmpUpdates

After insert or update on emp

Begin

-- Trigger Code

-- Trigger Code

TotSal**;**

-- Trigger Code

End

**Instead Of Triggers**

* Provides a transparent way of modifying views that cannot be modified directly through INSERT, UPDATE or DELETE statements because underlying tables contain joins
* It is called INSTEAD OF because Oracle fires the trigger instead of the triggering statement(s)
* Users can be transparent to the trigger because they write normal DML statements against the view and the INSTEAD OF trigger is used to take care of the modifications
* It can be placed on Object Views to insert, update or delete data in the underlying relational tables

CREATE TRIGGER emp\_insert

INSTEAD OF INSERT ON emp\_view

BEGIN

statements;

END;

**Instead of triggers can be used only with views.**

**Effective for joins which are based on equi join**

**To have an cascading effect of update on both the tables if columns are -matching**

**--Step 1 Creating tables s and r;**

create table s

(rollno number,

name varchar2(20));

create table r

(rollno number,

marks number);

**--Step 2 Inserting records in s and r.**

insert into s

values(1,'a');

insert into s

values(2,'b');

insert into r

values(1,90);

insert into r

values(2,87);

**--Step 3 Creating an Equijoin View on s and r**

Create or replace view SR

as

select s.rollno,s.name,r.marks

from s,r

where s.rollno =r.rollno;

Select \* from SR;

**Insert into SR Values(3,'c',99); -- Error**

**Example of Instead of Insert**

Create or replace trigger tig14

**Instead Of** INSERT on **SR**

**For Each Row**

Begin

/\*Inserting the new record into both the tables s and r.\*/

Insert into s

values(:new.rollno, :new.name);

Insert into r

values(:new.rollno, :new.marks);

End;

**Insert into SR Values(3,'c',99); -- Works fine now!!!**