**PRACTICAL NO 8**

**CREATING TRIGGERS IN PL/SQL**

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

* A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
* A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
* A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Advantages of Triggers

These are the following advantages of Triggers:

* Trigger generates some derived column values automatically
* Enforces referential integrity
* Event logging and storing information on table access
* Auditing
* Synchronous replication of tables
* Imposing security authorizations
* Preventing invalid transactions

**The syntax for creating a trigger is –**

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)

DECLARE

Declaration-statements

BEGIN

Executable-statements

EXCEPTION

Exception-handling-statements

END;

Where,

* CREATE [OR REPLACE] TRIGGER trigger\_name − Creates or replaces an existing trigger with the *trigger\_name*.
* {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
* {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
* [OF col\_name] − This specifies the column name that will be updated.
* [ON table\_name] − This specifies the name of the table associated with the trigger.
* \*\*[FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
* WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

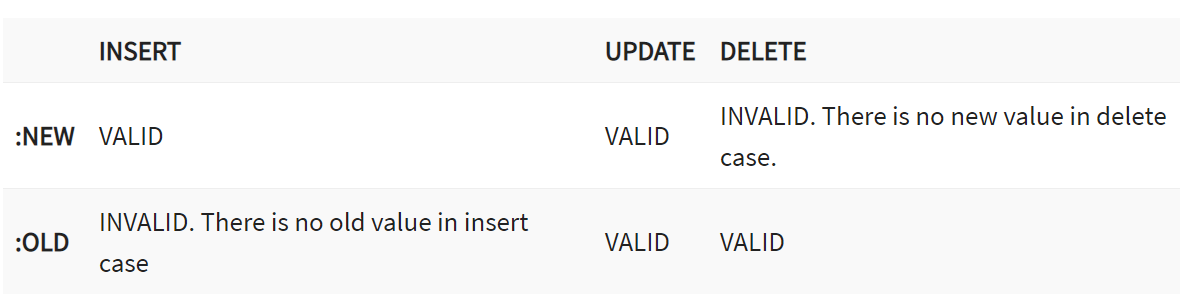
**:NEW and :OLD Clause**

In a row level trigger, the trigger fires for each related row. And sometimes it is required to know the value before and after the DML statement.

Oracle has provided two clauses in the RECORD-level trigger to hold these values. We can use these clauses to refer to the old and new values inside the trigger body.

* :NEW – It holds a new value for the columns of the base table/view during the trigger execution
* :OLD – It holds old value of the columns of the base table/view during the trigger execution

This clause should be used based on the DML event. Below table will specify which clause is valid for which DML statement (INSERT/UPDATE/DELETE).



### **Types Of Triggers In PL/SQL**

* **ROW Level trigger:** It gets executed for each record that got updated by a DML statement.
* **STATEMENT Level trigger:** It gets executed only once by the event statement.
* **INSTEAD OF:**  It is used when any DML event is going to occur on the complex view.

1. **CREATE 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.

* 1. **Create a trigger: after insert, update, delete statement level.**

**Creation of table: -**

create table Student ( Sno int, Sname varchar2(15), Marks int, Result varchar2(10) );

**Creation of trigger for insert command: -**

set serveroutput on;

create trigger student\_Insert\_Trigger

After insert on Student

begin

if inserting then

dbms\_output.put\_line('After insertion trigger fired');

end if;

end;

/

**Insert 10 rows :-**

Insert into student values(1,’manish’,85,’pass’);

**Creation of trigger for delete command: -**

set serveroutput on;

create trigger student\_Delete\_Trigger

After DELETE on student

begin

if deleting then

dbms\_output.put\_line('After deletion trigger fired');

end if;

end;

/

1. **CREATE ROW LEVEL STATEMENT**

A row-level trigger **fires once for each row that is affected by a triggering event**

CREATE OR REPLACE TRIGGER display\_salary\_changes

BEFORE DELETE OR UPDATE ON cust\_trigger

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;

/

UPDATE cust\_trigger

SET salary = salary + 500

WHERE id = 2;

1. **Instead of trigger**

INSTEAD OF triggers control insert, update, merge, and delete operations on views, not tables. They can be used to make nonupdateable views updateable and to override the default behavior of views that are updateable.

**Syntax:-**

CREATE [OR REPLACE] TRIGGER *trigger\_name*

INSTEAD OF *operation*

ON *view\_name*

FOR EACH ROW

BEGIN

...*code goes here*..

END;

1. **Create table emp and dept. insert values into respective tables.**

**Create a view with fields from both tables.**

**Create a instead of trigger for updating value in the view.**

CREATE TABLE emp(

emp\_no NUMBER,

emp\_name VARCHAR2(50),

salary NUMBER,

manager VARCHAR2(50),

dept\_no NUMBER);

/

CREATE TABLE dept(

Dept\_no NUMBER,

Dept\_name VARCHAR2(50),

LOCATION VARCHAR2(50));

/

BEGIN

INSERT INTO DEPT VALUES(10,’HR’,’USA’);

INSERT INTO DEPT VALUES(20,’SALES’,’UK’);

INSERT INTO DEPT VALUES(30,’FINANCIAL’,’JAPAN’);

COMMIT;

END;

/

BEGIN

INSERT INTO EMP VALUES(1000,’raj’,15000,’bharat’,30);

INSERT INTO EMP VALUES(1001,’alex’,18000,’raj’,20) ;

INSERT INTO EMP VALUES(1002,’simran’,20000,’alex’,10);

COMMIT;

END;

/

**CREATE VIEW**

CREATE VIEW dept\_emp\_view(Employee\_name,dept\_name,location) AS

SELECT emp.emp\_name,dept.dept\_name,dept.location

FROM emp,dept

WHERE emp.dept\_no=dept.dept\_no;

/

**Create trigger**

CREATE TRIGGER dept\_view\_modify\_trg

INSTEAD OF UPDATE

ON dept\_emp\_view

FOR EACH ROW

BEGIN

UPDATE dept

SET location=:new.location

WHERE dept\_name=:old.dept\_name;

END;

/

**Update location to ‘rome’ for dept\_name=’sales’**

BEGIN

UPDATE dept\_emp\_view SET location=’rome’ WHERE Dept\_name =’sales’;

COMMIT;

END;

/