-- Create the emp\_mstr table

CREATE TABLE emp\_mstr (

e\_id NUMBER(2) NOT NULL,

first\_name VARCHAR2(30),

last\_name VARCHAR2(30),

salary NUMBER(8,2),

CONSTRAINT pk\_emp\_mstr PRIMARY KEY (e\_id)

);

-- Insert 5 records into the emp\_mstr table

INSERT INTO emp\_mstr (e\_id, first\_name, last\_name, salary)

VALUES (1, 'John', 'Doe', 55000.00);

INSERT INTO emp\_mstr (e\_id, first\_name, last\_name, salary)

VALUES (2, 'Jane', 'Smith', 62000.00);

INSERT INTO emp\_mstr (e\_id, first\_name, last\_name, salary)

VALUES (3, 'Michael', 'Brown', 47000.00);

INSERT INTO emp\_mstr (e\_id, first\_name, last\_name, salary)

VALUES (4, 'Emily', 'Davis', 71000.00);

INSERT INTO emp\_mstr (e\_id, first\_name, last\_name, salary)

VALUES (5, 'David', 'Wilson', 53000.00);

-- Commit the changes

COMMIT;

Oracle plsql triggers:

A database trigger is a stored program which is automatically fired or executed when some events occur. A trigger can execute in response to any of the following events:  
1. A database manipulation (DML) statement like DELETE, INSERT or UPDATE.  
2. A database definition (DDL) statement like CREATE, ALTER or DROP.  
3. A database operation like SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN.  
**Note: A trigger can be defined on the table, view, schema or database with which the event is associated.**

Types of PL SQL triggers:

**1. Row level trigger** – An event is triggered at row level i.e. for each row updated, inserted or deleted.  
**2. Statement level trigger** – An event is triggered at table level i.e. for each sql statement executed.

Syntax for creating a trigger:

|  |
| --- |
| 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;  / |

Where:

**CREATE [OR REPLACE ] TRIGGER trigger\_name** – It creates a trigger with the given name or overwrites an existing trigger with the same name.  
**{BEFORE | AFTER | INSTEAD OF }** – It specifies the trigger get fired. i.e before or after updating a table. INSTEAD OF is used to create a trigger on a view.  
**{INSERT [OR] | UPDATE [OR] | DELETE}** – It specifies the triggering event. The trigger gets fired at all the specified triggering event.  
**[OF col\_name]** – It is used with update triggers. It is used when we want to trigger an event only when a specific column is updated.  
**[ON table\_name]** – It specifies the name of the table or view to which the trigger is associated.  
**[REFERENCING OLD AS o NEW AS n]** – It 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 old values cannot be referenced when inserting a record and new values cannot be referenced when deleting a record, because they do not exist.  
**[FOR EACH ROW]** – It is used to specify whether a trigger must fire when each row being affected (Row Level Trigger) or just once when the sql statement is executed (Table level Trigger).  
**WHEN (condition)** – It is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

Example:

**Existing data:**Select \* from employees;

|  |
| --- |
| EMP\_ID NAME AGE ADDRESS SALARY  1 Shveta 23 Delhi 50000  2 Bharti 22 Karnal 52000  3 Deepika 24 UP 54000  4 Richi 25 US 56000  5 Bharat 21 Paris 58000  6 Sahdev 26 Delhi 60000 |
|  |

**Trigger:**

|  |
| --- |
| CREATE OR REPLACE TRIGGER show\_salary\_differences  BEFORE DELETE OR INSERT OR UPDATE ON employees  **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;  / |

**Note: The above trigger will execute for every INSERT, UPDATE or DELETE operations performed on the EMPLOYEES table.**

**OUTPUT:**

**SQL> set serveroutput on;**

**SQL>  insert into emp values(4, 'jenny', 'ceo', 10000, '28-feb-20');**

**Old salary:**

**New salary: 10000**

**Salary difference:**

**1 row created.**

**SQL> UPDATE EMP SET SALARY= SALARY+500 WHERE ID=1;**

**Old salary: 10000**

**New salary: 10500**

**Salary difference: 500**

**1 row updated.**

**SQL>**

Drop a trigger:

|  |
| --- |
| DROP TRIGGER trigger\_name; |