**Name: Niranjan Vinod Patil.**

**Batch: B-3.**

**Roll No: SCSB307.**

**Subject: DBMS.**

**Theory Assignment No: 02**

**Q.1** Consider an Employee (Emp\_Id, Emp\_Name, Address, Age,Salary). Write a trigger on Employee table which will display salary difference.

**Rubrics:**

Syntax of trigger: 3 marks

Executable statements in Trigger body: 3 marks

**Answer:**

1. **Syntax of trigger:**

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

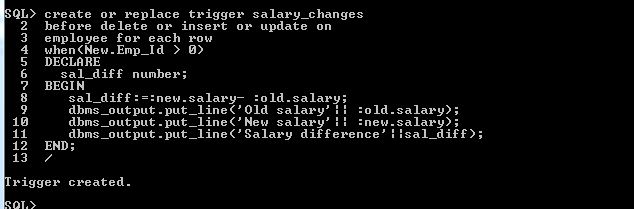
Exception-handling-statements

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.
* [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
* [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.

1. **Executable statements in Trigger body:**



create or replace trigger salary\_changes

before delete or insert or update on

employee for each row

when(New.Emp\_Id > 0)

DECLARE

sal\_diff number;

BEGIN

sal\_diff:=:new.salary- :old.salary;

dbms\_output.put\_line('Old salary'|| :old.salary);

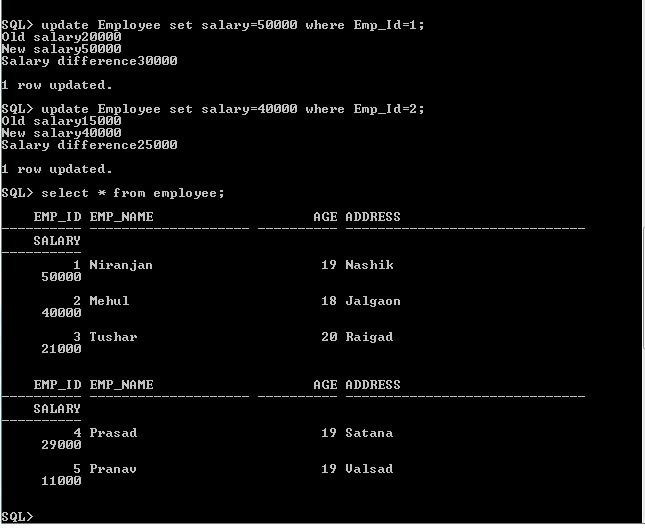
dbms\_output.put\_line('New salary'|| :new.salary);

dbms\_output.put\_line('Salary difference'||sal\_diff);

END;

/

OUTPUT:

**Records of table Employee before trigger is generated:****After updating Records of table Employee and trigger is generated:**

**Q.2.** Consider an Employee (Emp\_Id, Emp\_Name, Address, Age,Salary). Write an Exception when Employee entered Invalid Id(i.e. if Emp\_Id <=0) the system should generate Invalid\_Id exception.

**Rubrics:**

Syntax of Exception : 2 marks

Invaid\_Id Exception : 2 marks

**Answer:**

**1) Syntax of Exception:**

**Raising Exceptions**

Exceptions are raised by the database server automatically whenever there is any internal database error, but exceptions can be raised explicitly by the programmer by using the command **RAISE**. Following is the simple syntax for raising an exception −

DECLARE

exception\_name EXCEPTION;

BEGIN

IF condition THEN

RAISE exception\_name;

END IF;

EXCEPTION

WHEN exception\_name THEN

statement;

END;

**2)Invaid\_Id Exception :**

DECLARE

E\_Id Employee.EMP\_Id%type:=&E\_id;

E\_name Employee.EMP\_Name%type;

-- user defined exception

ex\_invalid\_id EXCEPTION;

BEGIN

IF E\_Id <= 0 THEN

RAISE ex\_invalid\_id;

ELSE

SELECT Emp\_Id,Emp\_name INTO E\_Id,E\_name

FROM Employee

WHERE Emp\_Id = E\_Id;

dbms\_output.put\_line('Employee ID ' ||' ' ||E\_Id ||' '|| 'Name is:' ||' ' ||E\_name );

END IF;

EXCEPTION

WHEN ex\_invalid\_id THEN

dbms\_output.put\_line('Employee ID must be greater than zero!');

WHEN no\_data\_found THEN

dbms\_output.put\_line('No such Employee!');

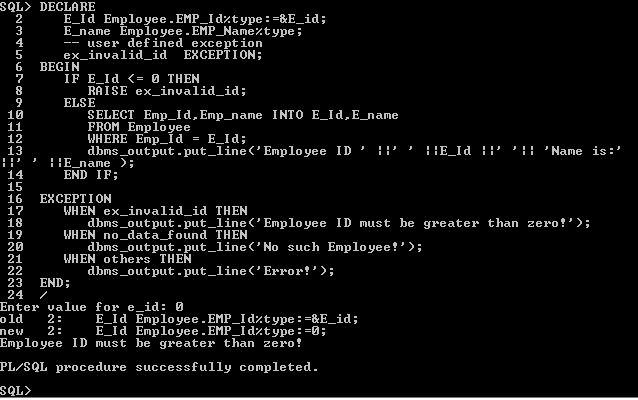
WHEN others THEN

dbms\_output.put\_line('Error!');

END;

/

**OUTPUT of EXCEPTION:**

****