

#### Lab Exercise 4:

Aim:

Write a program to demonstrate the use of Trigger in PL/SQL.

Procedure:

**Step 1: CREATE [OR REPLACE] TRIGGER trigger\_n** – This is for creating, replacing, or updating a trigger having a name as trigger\_n.

**Step 2: {BEFORE | AFTER | INSTEAD OF}** – This is for determining the time when the trigger will be fired. The INSTEAD OF is for creating a trigger that has a view.

**Step 3: {INSERT [OR] | UPDATE [OR] | DELETE}** – This is for executing the DML actions.

**Step 4: [OF column\_n]** – This is for mentioning the column name that shall be modified.

**Step 5: [ON table\_n]** – This is for mentioning the table name that is attached to the trigger.

**Step 6: [REFERENCING OLD AS NEW AS n]** – This is for referring to the old and new values by the DML statement like UPDATE, INSERT or DELETE.

**Step 7: [FOR EACH ROW]** – This determines a row-level trigger, i.e., the trigger will be fired for each row that is modified, else the trigger will fire just once when the SQL statement is executed, which is known as a table-level trigger.

**Step 8: WHEN (condition)** – This gives a condition for rows for which the trigger would be executed. This is applicable only row-level triggers

#### Create the Employee Table:-

```
CREATE TABLE employees ( emp_id NUMBER PRIMARY KEY, emp_name
VARCHAR2(50), deptno NUMBER, sal NUMBER, comm NUMBER );
```

#### Insert Employee Table:-

```
INSERT INTO employees VALUES (1, 'Ramesh', 10, 50000, 5000);
```

```
INSERT INTO employees VALUES (2, 'Suresh', 20, 60000, 6000);
```

#### Create the salary\_audit Table:-

```
CREATE TABLE salary_audit (audit_id NUMBER PRIMARY KEY, emp_id
NUMBER, old_salary NUMBER, new_salary NUMBER, change_date
TIMESTAMP);
```

#### Create Sequence for Audit Table:-

```
CREATE SEQUENCE salary_audit_seq START WITH 1 INCREMENT BY 1;
```

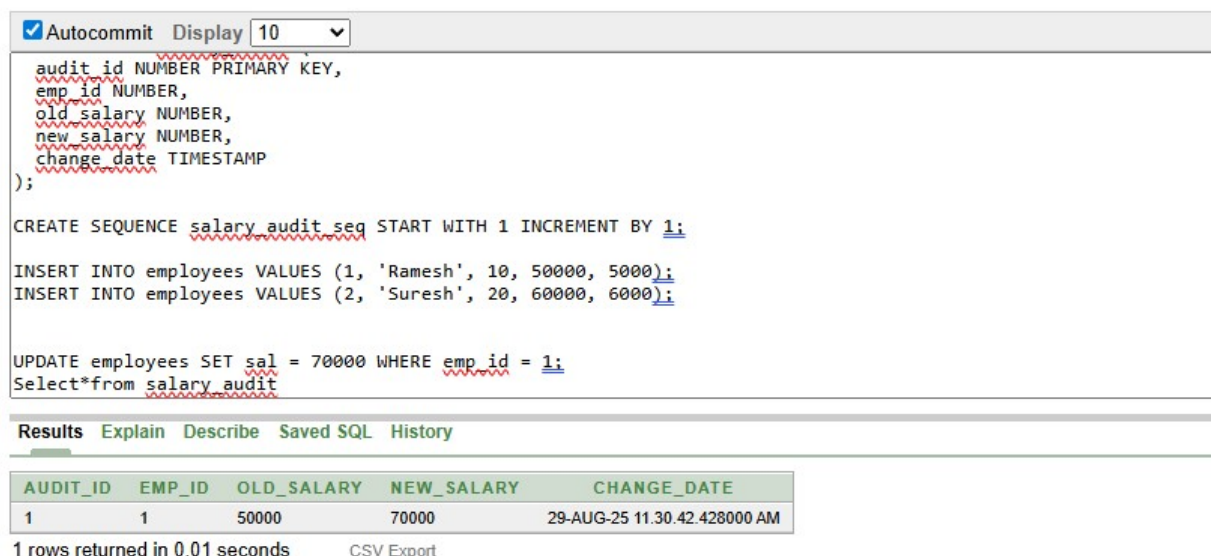
### Create an Update Trigger:-

```
CREATE OR REPLACE TRIGGER trg_salary_audit
AFTER UPDATE OF sal ON employees
FOR EACH ROW
WHEN (NEW.sal <> OLD.sal)
DECLARE v_audit_id NUMBER;
BEGIN SELECT salary_audit_seq.NEXTVAL INTO v_audit_id FROM DUAL;
INSERT INTO salary_audit (audit_id, emp_id, old_salary, new_salary,
change_date) VALUES (v_audit_id, :OLD.emp_id, :OLD.sal, :NEW.sal,
SYSTIMESTAMP);
END;
/
```

### Check The Trigger:-

```
UPDATE employees SET sal = 70000 WHERE emp_id = 1;
SELECT * FROM salary_audit;
```

### Output:



The screenshot shows a SQL Developer window with the following SQL script executed:

```
audit_id NUMBER PRIMARY KEY,
emp_id NUMBER,
old_salary NUMBER,
new_salary NUMBER,
change_date TIMESTAMP
);

CREATE SEQUENCE salary_audit_seq START WITH 1 INCREMENT BY 1;

INSERT INTO employees VALUES (1, 'Ramesh', 10, 50000, 5000);
INSERT INTO employees VALUES (2, 'Suresh', 20, 60000, 6000);

UPDATE employees SET sal = 70000 WHERE emp_id = 1;
Select*from salary_audit
```

Below the script, the 'Results' tab is active, displaying a table with the following data:

AUDIT_ID	EMP_ID	OLD_SALARY	NEW_SALARY	CHANGE_DATE
1	1	50000	70000	29-AUG-25 11.30.42.428000 AM

1 rows returned in 0.01 seconds [CSV Export](#)

**Result:**

**Thus the trigger with PL/SQL queries was executed successfully.**