4. Create a row level for CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary between the old and new salary.

AIM: To create a row-level trigger that fires on INSERT, UPDATE, or DELETE operations and displays the salary difference between the old and new salary for the CUSTOMERS table, you can use the following SQL script.

• CREATE TABLE CUSTOMERS (

ID INT PRIMARY KEY,

NAME VARCHAR(255),

AGE INT,

ADDRESS VARCHAR(255),

SALARY DECIMAL(10, 2) );

• CREATE OR REPLACE TRIGGER trg\_customers\_salary\_diff

AFTER INSERT OR UPDATE OR DELETE

ON CUSTOMERS

FOR EACH ROW

DECLARE

old\_salary NUMBER;

new\_salary NUMBER;

salary\_diff NUMBER;

BEGIN

-- Handle INSERT operation

IF INSERTING THEN

old\_salary := NULL;

new\_salary := :NEW.salary;

salary\_diff := new\_salary;

DBMS\_OUTPUT.PUT\_LINE('Inserted row: Salary = ' || new\_salary);

-- Handle DELETE operation

ELSIF DELETING THEN

old\_salary := :OLD.salary;

new\_salary := NULL;

salary\_diff := old\_salary;

DBMS\_OUTPUT.PUT\_LINE('Deleted row: Salary = ' || old\_salary);

-- Handle UPDATE operation

ELSIF UPDATING THEN

old\_salary := :OLD.salary;

new\_salary := :NEW.salary;

salary\_diff := new\_salary - old\_salary;

DBMS\_OUTPUT.PUT\_LINE('Updated row: Old Salary = ' || old\_salary || ', New Salary = ' || new\_salary || ', Difference = ' || salary\_diff);

END IF;

END;/

Testing the trigger:

• Prerequisites: Ensure that you have enabled server output in your SQL environment.

SET SERVEROUTPUT ON;

• INSERT operation:

INSERT INTO CUSTOMERS VALUES (1,'John Doe', 30, '123 Elm St', 50000.00);

Inserted row: Salary = 50000.00

1 row created.

• UPDATE operation:

UPDATE CUSTOMERS SET SALARY = 55000.00 WHERE NAME = 'John Doe';

Updated row: Old Salary = 50000.00, New Salary = 55000.00, Difference = 5000.00

1 row updated.

• DELETE operation:

DELETE FROM CUSTOMERS WHERE NAME = 'John Doe';

Deleted row: Salary = 55000.00

1 row deleted.

5. Create cursor for Employee table & extract the values from the table. Declare the variables, Open the cursor & extract the values from the cursor. Close the cursor.

Employee(E\_id, E\_name, Age, Salary)

AIM: To create cursor for Employee table & extract the values from the table. Declare the variables, Open the cursor & extract the values from the cursor. Close the cursor.

Table creation:

• CREATE TABLE Employee (

E\_id NUMBER PRIMARY KEY,

E\_name VARCHAR2(255),

Age NUMBER,

Salary NUMBER );

• INSERT INTO Employee (E\_id, E\_name, Age, Salary) VALUES (1, 'Alice', 30, 50000);

• INSERT INTO Employee (E\_id, E\_name, Age, Salary) VALUES (2, 'Bob', 25, 45000);

• INSERT INTO Employee (E\_id, E\_name, Age, Salary) VALUES (3, 'Charlie', 35, 60000);

• COMMIT;

Creating the Cursor :

• Prerequisites: Ensure that you have enabled server output in your SQL environment.

SET SERVEROUTPUT ON;

• Steps for creating Cursor:

1. Declare Variables: Variables are declared to hold the values fetched from the Employee table.

2. Declare Cursor: A cursor is declared to select the E\_id, E\_name, Age, and Salary columns from the Employee table.

3. Open Cursor: The cursor is opened.

4. Fetch Values: Values are fetched into the declared variables inside a loop. The loop continues until no more rows are fetched.

5. Close Cursor: The cursor is closed to release the resources.

• CODE:

DECLARE

-- Declare variables to hold the values from the Employee table.

v\_E\_id Employee.E\_id%TYPE;

v\_E\_name Employee.E\_name%TYPE;

v\_Age Employee.Age%TYPE;

v\_Salary Employee.Salary%TYPE;

-- Declare the cursor

CURSOR emp\_cursor IS

SELECT E\_id, E\_name, Age, Salary

FROM Employee;

BEGIN

-- Open the cursor

OPEN emp\_cursor;

-- Loop through each row fetched by the curso

LOOP

FETCH emp\_cursor INTO v\_E\_id, v\_E\_name, v\_Age, v\_Salary;

-- Exit the loop when no more rows are fetched

EXIT WHEN emp\_cursor%NOTFOUND;

-- Here you can process the fetched values.

DBMS\_OUTPUT.PUT\_LINE('E\_id: ' || v\_E\_id || ', E\_name: ' || v\_E\_name || ', Age: ' || v\_Age || ', Salary: ' || v\_Salary);

END LOOP;

-- Close the cursor

CLOSE emp\_cursor;

END;

/

OUTPUT:

E\_id: 1, E\_name: Alice, Age: 30, Salary: 50000

E\_id: 2, E\_name: Bob, Age: 25, Salary: 45000

E\_id: 3, E\_name: Charlie, Age: 35, Salary: 60000

PL/SQL procedure successfully completed.

Experiment 6 : Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N\_RollCall with the data available in the table O\_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

Step 1: Create tables

CREATE TABLE N\_RollCall (

student\_id NUMBER PRIMARY KEY,

student\_name VARCHAR2(100),

birth\_date DATE

);

CREATE TABLE O\_RollCall (

student\_id NUMBER PRIMARY KEY,

student\_name VARCHAR2(100),

birth\_date DATE

);

Step 2:

Insert test data into N\_rollcall

INSERT INTO N\_rollcall VALUES (1, 'John Doe', '20-JAN-1990');

INSERT INTO N\_rollcall VALUES (2, 'Jane Smith', '21-MAR-1992');

INSERT INTO N\_rollcall VALUES (3, 'Bob', '20-july-2003');

Insert test data into O\_rollcall

INSERT INTO O\_rollcall VALUES (4, 'James', '20-JAN-1997');

INSERT INTO O\_rollcall VALUES (2, 'Jane Smith', '21-MAR-1992');

Step 3: Enable DBMS\_OUTPUTLINE

SET SERVEROUTPUT ON;

Step 4: Execute the PL/SQL block

DECLARE

V\_count NUMBER;

CURSOR c\_new\_rollcall IS

SELECT student\_id, student\_name, birth\_date

FROM n\_rollcall;

BEGIN

FOR new\_rec IN c\_new\_rollcall LOOP

SELECT COUNT(\*)

INTO v\_count

FROM o\_rollcall

WHERE student\_id = new\_rec.student\_id;

IF v\_count = 0 THEN

INSERT INTO o\_rollcall (student\_id, student\_name, birth\_date)

VALUES (new\_rec.student\_id, new\_rec.student\_name, new\_rec.birth\_date);

DBMS\_OUTPUT.put\_line('Record inserted: ' || new\_rec.student\_id);

ELSE

DBMS\_OUTPUT.put\_line('Record skipped: ' || new\_rec.student\_id);

END IF;

END LOOP;

COMMIT;

END;

/

OUTPUT:

Record inserted: 1

Record skipped: 2

Record inserted: 3

PL/SQL procedure successfully completed.

SQL>select \* from O\_rollcall;

Student\_id Student\_name Birth\_date

4 James 20-JAN-97

2 Jane Smith 21-MAR-92

1 John Doe 20-JAN-90

3 Bob 20-JUL-03

Experiment 7. Install an open source NoSQL data base Mango DB & perform basic CRUD(create,Read,Update & Delete) Operations. Execute Mango DB basic queries using CRUD operations.

1.Create collection student:

1. db.createCollection(‘student’)

2.Insert values in to student :

2. db.sales.insertMany([

{\_id : 1, Name:’xyz’, Branch:’AIML’, USN:37, Age:19},

{\_id : 2, Name:’sam’, Branch:’CSE’, USN:49, Age:20},

{\_id : 3, Name:’Jack’, Branch:’Mech’, USN:48, Age:21},

{\_id : 4, Name:’Viola’, Branch:’AIML’, USN:60, Age:22},

{\_id : 5, Name:’Lucy’, Branch:’AIML’, USN:72, Age:23} ])

3. UPDATE

db.student.updateOne({\_id:1},{$set:{age:20}})

4. REMOVE

db.student.updateOne({\_id:2},{$unset:{usn:1}})

5. RENAME

db.student.updateOne({\_id:3},{$rename:{name:’fullname’}})

6. ADD

db.student.updateOne({\_id:1},{$set:{marks:600}})

7. DELETE

db.student.deleteOne({\_id:4})

db.student.deleteMany({branch:’cse’})

8. PROJECTIONS

db.student.find({},{age:1,\_id:0})

9. COMPARISION OPERATOR

db.student.find({age:{$gt:20}})

db.student.find({$in:[12-25]})

10. LOGICAL OPERATOR

a. AND : db.student.find({$and:[{age:{$eq:20}},{branch:{$eq:’aiml’}}]})

b. OR: db.student.find({$or:[{usn:{$eq:72}},{branch:{$eq:’civil’}}]})