Ex.No.11	TRIGGERS

AIM

To implement and demonstrate the use of database triggers to perform and control INSERT, UPDATE, and DELETE function.

CREATE TABLE

```
SQL> CREATE TABLE library (
  2      book_id NUMBER PRIMARY KEY,
  3      title VARCHAR2(100),
  4      author VARCHAR2(50)
  5  );

Table created.
```

INSERT VALUES TO TABLE

```
SQL> INSERT INTO library (book_id, title, author) VALUES (1, 'The Alchemist', 'Paulo
Coelho');

1 row created.

SQL> INSERT INTO library (book_id, title, author) VALUES (2, 'Wings of Fire', 'A. P. J.
Abdul Kalam');

1 row created.

SQL> INSERT INTO library (book_id, title, author) VALUES (3, 'To Kill a Mockingbird',
'Harper Lee');

1 row created.

SQL> CREATE TABLE audit_library (
  2      book_id NUMBER,
  3      action_time DATE,
  4      action_type VARCHAR2(10)
  5  );


Table created.

SQL> CREATE OR REPLACE TRIGGER trg_audit_library
```

```
  2  AFTER INSERT OR UPDATE OR DELETE ON library
  3  FOR EACH ROW
  4  BEGIN
  5      IF INSERTING THEN
  6          INSERT INTO audit_library(book_id, action_time, action_type)
  7          VALUES(:NEW.book_id, SYSDATE, 'INSERT');
  8      ELSIF UPDATING THEN
  9          INSERT INTO audit_library(book_id, action_time, action_type)
 10          VALUES(:NEW.book_id, SYSDATE, 'UPDATE');
 11      ELSIF DELETING THEN
 12          INSERT INTO audit_library(book_id, action_time, action_type)
 13          VALUES(:OLD.book_id, SYSDATE, 'DELETE');
 14      END IF;
 15  END;
 16  /
Trigger created.
SQL> INSERT INTO library (book_id, title, author) VALUES (4, '1984', 'George Orwell');
1 row created.
SQL> UPDATE library SET author = 'Kalam A. P. J.' WHERE book_id = 2;
1 row updated.
SQL> DELETE FROM library WHERE book_id = 3;
1 row deleted.
SQL> SELECT * FROM audit_library;

   BOOK_ID ACTION_TI ACTION_TYP
---------- --------- ----------
         4 06-MAY-25 INSERT
         2 06-MAY-25 UPDATE
         3 06-MAY-25 DELETE
```

EXAMPLE 1

INSERT, UPDATE, DELETE ON EMPLOYEES TABLE

```
SQL> CREATE TABLE employees (
  2      emp_id NUMBER PRIMARY KEY,
  3      emp_name VARCHAR2(50),
  4      position VARCHAR2(30)
  5  );
Table created.
SQL> CREATE TABLE audit_employees (
  2      emp_id NUMBER,
  3      action_time DATE,
  4      action_type VARCHAR2(10)
  5  );
Table created.
SQL> CREATE OR REPLACE TRIGGER trg_employees_all_actions
  2  AFTER INSERT OR UPDATE OR DELETE ON employees
  3  FOR EACH ROW
  4  BEGIN
  5      IF INSERTING THEN
  6          INSERT INTO audit_employees(emp_id, action_time, action_type)
  7          VALUES(:NEW.emp_id, SYSDATE, 'INSERT');
  8      ELSIF UPDATING THEN
  9          INSERT INTO audit_employees(emp_id, action_time, action_type)
 10          VALUES(:NEW.emp_id, SYSDATE, 'UPDATE');
 11      ELSIF DELETING THEN
 12          INSERT INTO audit_employees(emp_id, action_time, action_type)
 13          VALUES(:OLD.emp_id, SYSDATE, 'DELETE');
 14      END IF;
 15  END;
 16  /
```

```
Trigger created.

SQL> INSERT INTO employees (emp_id, emp_name, position)
  2  VALUES (1, 'Ravi', 'Manager');

1 row created.

SQL> UPDATE employees
  2  SET position = 'Senior Manager'
  3  WHERE emp_id = 1;

1 row updated.

SQL> DELETE FROM employees
  2  WHERE emp_id = 1;

1 row deleted.

SQL> SELECT * FROM audit_employees;

   EMP_ID ACTION_TI ACTION_TYP
---------- --------- ----------
         1 06-MAY-25 INSERT
         1 06-MAY-25 UPDATE
         1 06-MAY-25 DELETE
```

<u>EXAMPLE 2</u>

<u>PREVENT NULL VALUE FOR CUSTOMERS:</u>

```
SQL> CREATE TABLE customers (
  2      customer_id NUMBER PRIMARY KEY,
  3      name VARCHAR2(100),
  4      email VARCHAR2(100)
  5  );

Table created.
SQL> INSERT INTO customers (customer_id, name, email)
  2  VALUES (1, 'John Doe', NULL);
INSERT INTO customers (customer_id, name, email)
            *
ERROR at line 1:
ORA-20001: Email cannot be NULL.
ORA-06512: at "SYSTEM.TRG_PREVENT_NULL_EMAIL", line 3
ORA-04088: error during execution of trigger 'SYSTEM.TRG_PREVENT_NULL_EMAIL'
```

TYPES OF TRIGGERS:

1.Row-level triggers

```
CREATE OR REPLACE TRIGGER salary_update_row_level
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
IF :NEW.salary > 50000 THEN
DBMS_OUTPUT.PUT_LINE('High salary: ' || :NEW.salary);
END IF;
END;
/
Trigger created.
```

2.Statement-level triggers

```
CREATE OR REPLACE TRIGGER salary_update_row_level
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
IF :NEW.salary > 50000 THEN
DBMS_OUTPUT.PUT_LINE('High salary: ' || :NEW.salary);
END IF;
END;
/
Trigger created.
```

3.Schema Triggers

```
CREATE OR REPLACE TRIGGER track_table_creation
AFTER CREATE ON SCHEMA
BEGIN
DBMS_OUTPUT.PUT_LINE('A new table has been created in the schema.');
END;
/
Trigger created.
```

4.Database-level triggers

```
CREATE OR REPLACE TRIGGER log_login_activity
AFTER LOGON ON DATABASE
BEGIN
DBMS_OUTPUT.PUT_LINE('A user has logged into the database.');
END;
/
Trigger created.
```

5.BEFORE and AFTER triggers

BEFORE:

```
CREATE OR REPLACE TRIGGER salary_before_update
BEFORE UPDATE ON employees
```

```
FOR EACH ROW
BEGIN
IF :NEW.salary < 5000 THEN
RAISE_APPLICATION_ERROR(-20001, 'Salary cannot be less than 5000');
END IF;
END;
/
```

AFTER:

```
CREATE OR REPLACE TRIGGER update_salary_after_insert
AFTER INSERT ON employees
FOR EACH ROW

IF :NEW.salary > 5000 THEN
UPDATE employees SET salary = 5500 WHERE emp_id = :NEW.emp_id;
END IF;
END;
/
```

6.INSTEAD OF triggers

```
CREATE OR REPLACE TRIGGER update_employee_view
 INSTEAD OF UPDATE ON employee_view
 FOR EACH ROW
 BEGIN
 UPDATE employees
 SET salary = :NEW.salary
 WHERE emp_id = :OLD.emp_id;
 END;
 /
```

CREATE TABLE:

```
   CREATE TABLE salary_audit (

    emp_id NUMBER(10),

    old_salary NUMBER(10),

    new_salary NUMBER(10),

    change_date DATE

);

INSERT INTO salary_audit (emp_id, old_salary, new_salary, change_date)

VALUES (101, 5000, 6000, SYSDATE);

INSERT INTO salary_audit (emp_id, old_salary, new_salary, change_date)

VALUES (102, 4500, 5200, SYSDATE);
```

```
INSERT INTO salary_audit (emp_id, old_salary, new_salary, change_date)
VALUES (103, 7000, 8000, SYSDATE);
```

<u>TO DISPLAY THE CONTENTS OF THE TABLE REVISED</u>

```
SQL> SELECT * FROM salary_audit;
EMP_ID  OLD_SALARY  NEW_SALARY  CHANGE_DATE
------- ----------- ----------- -------------------
101     5000        6000        2025-05-04 10:30:00
102     4500        5200        2025-05-04 11:00:00
103     7000        8000        2025-05-04 11:15:00
```

<u>TO CREATE TRIGGER AND UPDATE THE SALARY VALUE</u>

```
SQL> CREATE OR REPLACE TRIGGER update_salary_after_insert
AFTER INSERT ON revised
FOR EACH ROW
BEGIN
    -- Update the salary if the inserted salary is greater than 5000
    IF :NEW.salary > 5000 THEN
        UPDATE revised
        SET salary = 25000
        WHERE empid = :NEW.empid;
    END IF;
END;
/
Trigger created.
```

CONTENTS	**MARKS ALLOTED**	**MARKS OBTAINED**
Aim,Algorithm,SQL,PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

RESULT

Thus, the experiment successfully showcased how **database triggers** can be used for **enforcing business rules and maintaining audit trails** automatically.