

Database Management System

EXPERIMENT 13 TRIGGERS

NAME : PRATHESHA J

ROLL NO : 241001172

1. Create a trigger that prevents all DML operations on ITEMPLS table

```
CREATE OR REPLACE TRIGGER prevent_dml_trigger
BEFORE INSERT OR UPDATE OR DELETE ON itempls
FOR EACH ROW
BEGIN
    RAISE_APPLICATION_ERROR(-20010, 'You cannot perform DML
operations on this table');
END prevent_dml_trigger;
/
```

Expected Output: Trigger

created.

Test the trigger with INSERT:

```
INSERT INTO itempls VALUES ('aaa', 14, 34000);
```

Expected Output:

ERROR at line 1:

ORA-20010: You cannot perform DML operations on this table

```
ORA-06512: at "STUDENT.PREVENT_DML_TRIGGER", line 2 ORA-  
04088: error during execution of trigger  
'STUDENT.PREVENT_DML_TRIGGER'
```

Test the trigger with DELETE:

```
DELETE FROM itempls WHERE ename = 'xxx';
```

Expected Output:

```
ERROR at line 1:  
ORA-20010: You cannot perform DML operations on this table  
ORA-06512: at "STUDENT.PREVENT_DML_TRIGGER", line 2  
ORA-04088: error during execution of trigger  
'STUDENT.PREVENT_DML_TRIGGER'
```

Test the trigger with UPDATE:

```
UPDATE itempls SET eid = 15 WHERE ename = 'yyy';
```

Expected Output:

```
ERROR at line 1:  
ORA-20010: You cannot perform DML operations on this table  
ORA-06512: at "STUDENT.PREVENT_DML_TRIGGER", line 2  
ORA-04088: error during execution of trigger  
'STUDENT.PREVENT_DML_TRIGGER'
```

2. Drop the trigger

```
DROP TRIGGER prevent_dml_trigger;
```

Expected Output:

Trigger dropped.

3. Create a trigger that prevents salary changes and invalid inserts

```
CREATE OR REPLACE TRIGGER salary_restriction_trigger
BEFORE INSERT OR UPDATE OF salary ON itempls
FOR EACH ROWDECLARE      v_reference_salary
itempls.salary%TYPE; BEGIN
    -- Get reference salary from employee with eid=12
    SELECT salary INTO v_reference_salary
    FROM itempls
WHERE eid = 12;

    -- Check if new salary is different from reference
    IF (:NEW.salary <> v_reference_salary) THEN
        RAISE_APPLICATION_ERROR(-20100, 'Salary cannot be
changed from reference value');    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        NULL; -- Handle case when reference employee doesn't
exist
END salary_restriction_trigger;
/
```

Expected Output: Trigger

created.

4. Cursor FOR LOOP to display employee IDs and names

```
DECLARE
BEGIN
    FOR emp_rec IN (SELECT eid, ename FROM ssempp) LOOP
        DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_rec.eid
||
                                ', Employee Name: ' ||
emp_rec.ename);
    END LOOP;
END;
/
```

Expected Output:

```
Employee ID: 1, Employee Name: nala
Employee ID: 2, Employee Name: kala
Employee ID: 5, Employee Name: ajay
Employee ID: 6, Employee Name: vijay
Employee ID: 3, Employee Name: nila
```

PL/SQL procedure successfully completed.

5. Update salaries using Cursor FOR LOOP

```
DECLARE
    CURSOR emp_cursor IS
        SELECT eid, ename, sal, dno
        FROM ssempp
        WHERE dno = 11;
BEGIN
    FOR emp_rec IN emp_cursor LOOP
```

```

        UPDATE ssemp
        SET sal = emp_rec.sal + 5000
        WHERE eid = emp_rec.eid;
    END LOOP;
    COMMIT;
END;
/

```

Expected Output: PL/SQL procedure
successfully completed.

Verify the update:

```
SELECT * FROM ssemp;
```

Expected Output:

	EID	ENAME	JOB	SAL
DNO				

1	nala		lecturer	39000
11				
2	kala		seniorlecturer	20000
12				
5	ajay		lecturer	35000
11				
6	vijay		lecturer	23000
11				
3	nila		professor	

60000

12

6. Explicit cursor to display employee details

DECLARE

CURSOR emp_cursor IS

SELECT eid, sal

FROM ssempp

WHERE dno = 11; v_emp_id

ssempp.eid%TYPE; v_salary

ssempp.sal%TYPE;

BEGIN

OPEN emp_cursor;

LOOP

FETCH emp_cursor INTO v_emp_id, v_salary;

EXIT WHEN emp_cursor%NOTFOUND;

DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_emp_id ||
' , Salary: ' || v_salary);

END LOOP;

CLOSE emp_cursor;

END;

/

Expected Output:

Employee ID: 1, Salary: 39000

Employee ID: 5, Salary: 35000

Employee ID: 6, Salary: 23000

PL/SQL procedure successfully completed.

7. Implicit cursor to update salaries and check results

```
DECLARE      v_rows_updated
NUMBER;
BEGIN
    UPDATE ssemp
    SET sal = sal + 10000
    WHERE job = 'lecturer';
    v_rows_updated :=
SQL%ROWCOUNT;

    IF v_rows_updated > 0 THEN
        DBMS_OUTPUT.PUT_LINE('Number of rows updated: ' ||
v_rows_updated);    END IF;

    IF SQL%FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Update operation successful');
    ELSEIF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('No records found for update');
    END IF;

    COMMIT;
END;
/
```

Expected Output:

Number of rows updated: 3
Update operation successful

PL/SQL procedure successfully completed.

Verify final table state:

```
SELECT * FROM ssempp;
```

Expected Output:

	EID	ENAME	JOB	SAL
DNO				

- - - - -				
1	nala		lecturer	49000
	11			
2	kala		seniorlecturer	20000
	12			
5	ajay		lecturer	45000
	11			
6	vijay		lecturer	33000
	11			
3	nila		professor	
60000		12		

Additional Trigger Examples:

BEFORE INSERT trigger for auditing

```
CREATE OR REPLACE TRIGGER audit_employee_insert
BEFORE INSERT ON ssempp
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('Inserting new employee: '
|| :NEW.ename);
    DBMS_OUTPUT.PUT_LINE('Salary: ' || :NEW.sal);
END;
```

/

AFTER UPDATE trigger for logging changes

```
CREATE OR REPLACE TRIGGER log_salary_changes
AFTER UPDATE OF sal ON ssempp
FOR EACH ROW
BEGIN
    IF :OLD.sal != :NEW.sal THEN
        DBMS_OUTPUT.PUT_LINE('Salary changed for '
|| :OLD.ename);
        DBMS_OUTPUT.PUT_LINE('Old salary: ' || :OLD.sal ||
', New salary: ' || :NEW.sal);
    END IF;
END;
/
```

Key Trigger Concepts Demonstrated:

1. **BEFORE Triggers** - Execute before the DML operation
2. **AFTER Triggers** - Execute after the DML operation
3. **ROW Level Triggers** - Execute for each row affected
4. **STATEMENT Level Triggers** - Execute once per statement
5. **:OLD and: NEW** - Reference old and new column values 6.
RAISE_APPLICATION_ERROR - Custom error messages 7. **Cursor Types:**
 - a. Implicit cursors (SQL%)
 - b. Explicit cursors (user-defined)
 - c. Cursor FOR loops