TASK 7: Triggers, Views, and Exception Handling

AIM: To understand and implement Triggers, Views, and Exception Handling for managing CRUD operations in an Oracle database**.**

Part 0: Setup (Base Tables)

Before starting, create base tables to work with.

```
DROP TABLE students CASCADE CONSTRAINTS;
DROP TABLE departments CASCADE CONSTRAINTS;
-- Drop tables if they already exist
DROP TABLE student_log CASCADE CONSTRAINTS;

-- Create departments table
CREATE TABLE departments (
    dept_id      NUMBER PRIMARY KEY,
    dept_name    VARCHAR2(50),
    hod_name     VARCHAR2(50)
);

-- Create students table
CREATE TABLE students (
    student_id   NUMBER PRIMARY KEY,
    student_name VARCHAR2(50),
    age          NUMBER,
    dept_id      NUMBER,
    marks        NUMBER(5,2),
    CONSTRAINT fk_dept FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);
```

Part 1: Implementing Triggers

1. Prevent Insertion of Underage Students

Create a trigger that prevents inserting students under 18 years old.

```
CREATE OR REPLACE TRIGGER trg_prevent_underage_students
```

```
BEFORE INSERT ON students
FOR EACH ROW
BEGIN
   IF :NEW.age < 18 THEN
      RAISE_APPLICATION_ERROR(-20001, 'Student must be at least 18 years old.');
   END IF;
END;
/
```

✅ Test:

```
INSERT INTO students VALUES (1, 'Alex', 17, NULL, 85);
-- Should raise error: Student must be at least 18 years old
```

---

2. Create a Log Table

This table and trigger will log insert, update, and delete operations on the students table.

```
-- Log table
CREATE TABLE student_log (
   log_id       NUMBER GENERATED ALWAYS AS IDENTITY,
   operation    VARCHAR2(10),
   student_id   NUMBER,
   log_date     DATE,
   old_marks    NUMBER(5,2),
   new_marks    NUMBER(5,2)
);

-- Trigger for logging
CREATE OR REPLACE TRIGGER trg_student_log
AFTER INSERT OR UPDATE OR DELETE ON students
FOR EACH ROW
BEGIN
   IF INSERTING THEN
```

```
    INSERT INTO student_log (operation, student_id, log_date)
    VALUES ('INSERT', :NEW.student_id, SYSDATE);
  ELSIF UPDATING THEN
    INSERT INTO student_log (operation, student_id, log_date, old_marks, new_marks)
    VALUES ('UPDATE', :OLD.student_id, SYSDATE, :OLD.marks, :NEW.marks);
  ELSIF DELETING THEN
    INSERT INTO student_log (operation, student_id, log_date, old_marks)
    VALUES ('DELETE', :OLD.student_id, SYSDATE, :OLD.marks);
  END IF;
END;
/
```

✅ **Test:**

```
INSERT INTO students VALUES (2, 'John', 20, NULL, 88);
UPDATE students SET marks = 90 WHERE student_id = 2;
DELETE FROM students WHERE student_id = 2;

SELECT * FROM student_log;
```

Part 2: Creating Views

1. View for Top Students

Create a view showing students with marks greater than or equal to 85.

```
CREATE OR REPLACE VIEW vw_top_students AS
SELECT student_id, student_name, marks
FROM students
WHERE marks >= 85;
```

✅ Query:

```
SELECT * FROM vw_top_students;
```

---

2. View for Department Summary

Show department name, total students, and average marks.

```
CREATE OR REPLACE VIEW vw_department_summary AS
SELECT d.dept_name,
       COUNT(s.student_id) AS total_students,
       AVG(s.marks) AS avg_marks
FROM departments d
LEFT JOIN students s ON d.dept_id = s.dept_id
GROUP BY d.dept_name;
```

✅ **Query:**

```
SELECT * FROM vw_department_summary;
```

Part 3: Exception Handling

1. Stored Procedure with Exception Handling for Inserting Student Records

Create a procedure that inserts a student and handles possible errors (like foreign key or age violation).

```
CREATE OR REPLACE PROCEDURE add_student(
    p_id NUMBER,
    p_name VARCHAR2,
    p_age NUMBER,
    p_dept NUMBER,
    p_marks NUMBER
)
IS
BEGIN
    INSERT INTO students (student_id, student_name, age, dept_id, marks)
    VALUES (p_id, p_name, p_age, p_dept, p_marks);

    DBMS_OUTPUT.PUT_LINE('Student inserted successfully.');

EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('Error: Student ID already exists.');
```

```
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

✅ **Test:**

```
EXEC add_student(3, 'Maria', 19, 1, 92);
EXEC add_student(3, 'Duplicate', 22, 1, 80);  -- should raise duplicate ID error
```

---

2. Function to Fetch Student Details with Error Handling

Return student name and marks based on student ID.

```
CREATE OR REPLACE FUNCTION get_student_details(p_id NUMBER)
RETURN VARCHAR2
IS
    v_name students.student_name%TYPE;
    v_marks students.marks%TYPE;
BEGIN
    SELECT student_name, marks INTO v_name, v_marks
    FROM students
    WHERE student_id = p_id;

    RETURN 'Name: ' || v_name || ', Marks: ' || v_marks;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN 'Error: Student not found.';
    WHEN OTHERS THEN
        RETURN 'Error: ' || SQLERRM;
END;
/
```

✅ **Test:**

```
SELECT get_student_details(3) FROM dual;

SELECT get_student_details(99) FROM dual;  -- should return error message
```

✅ Summary of Deliverables

Part	Task	Object Created
1.1	Prevent underage students	TRIGGER trg_prevent_underage_students
1.2	Log operations	TABLE student_log, TRIGGER trg_student_log
2.1	Top students view	VIEW vw_top_students
2.2	Department summary view	VIEW vw_department_summary
3.1	Procedure with exception handling	PROCEDURE add_student
3.2	Function with error handling	FUNCTION get_student_details

RESULT:

To understand and implement Triggers, Views, and Exception Handling for managing CRUD operations in an Oracle database.