* CREATE DATABASE school\_db.
* CREATE TABLE students(studentId int PRIMARY KEY AUTO\_INCREMENT,student\_Name varchar(20),age int ,class varchar(20) , address varchar(50));
* INSERT into students(student\_Name,age,class,address)values("Vatsal","21","Python","Surat");

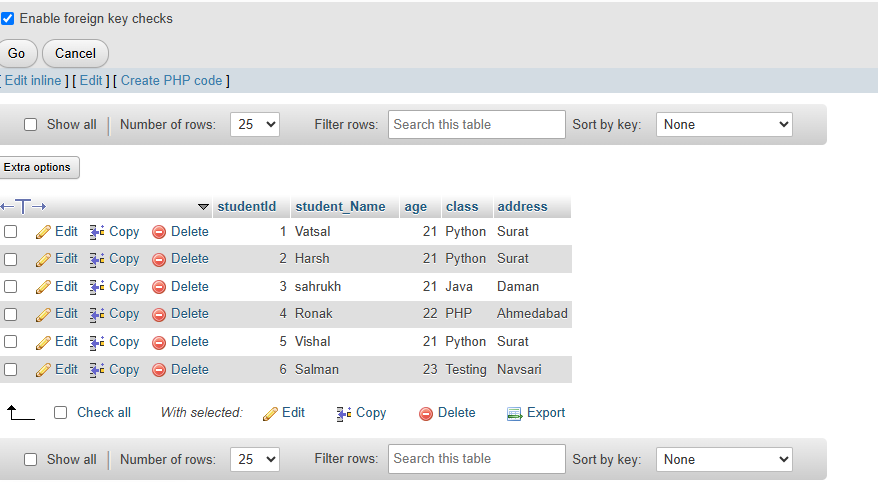
INSERT into students(student\_Name,age,class,address)values("Harsh","21","Java","Surat");

INSERT into students(student\_Name,age,class,address)values("Sahrukh","21","Java","Daman");

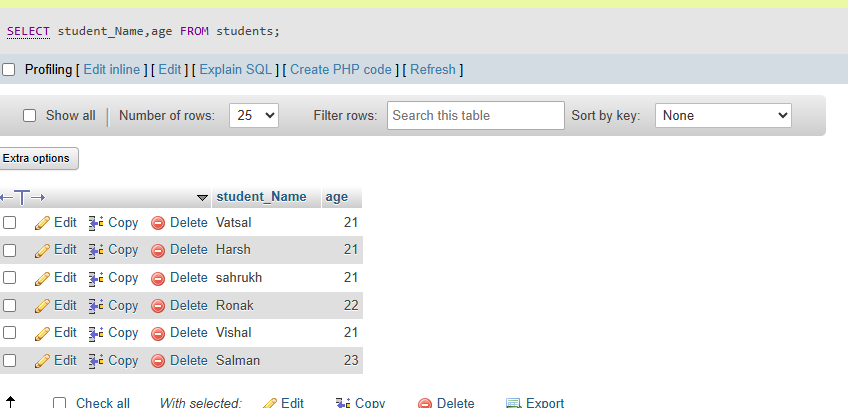
INSERT into students(student\_Name,age,class,address)values("Ronak","22","PHP","Ahmedabad");

INSERT into students(student\_Name,age,class,address)values("Vishal","21","Python","Surat");

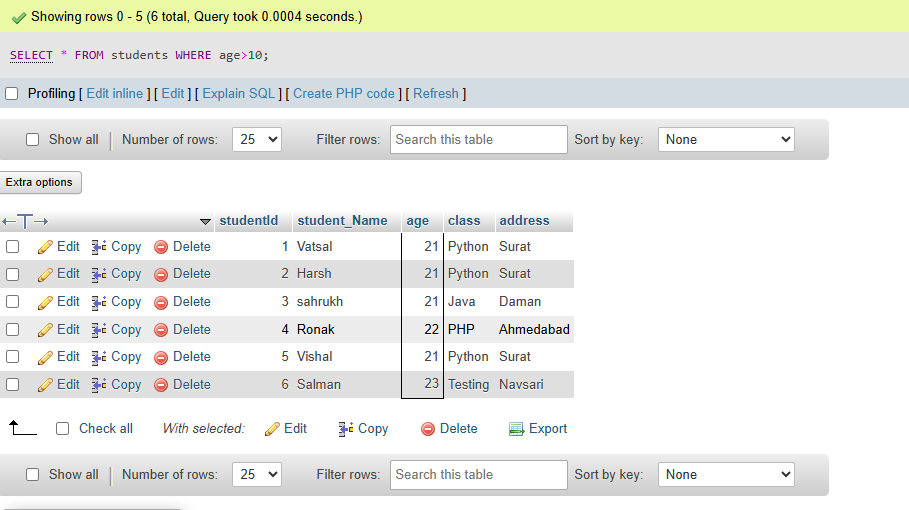
INSERT into students(student\_Name,age,class,address)values("Salman","23","Testing","Navsari");



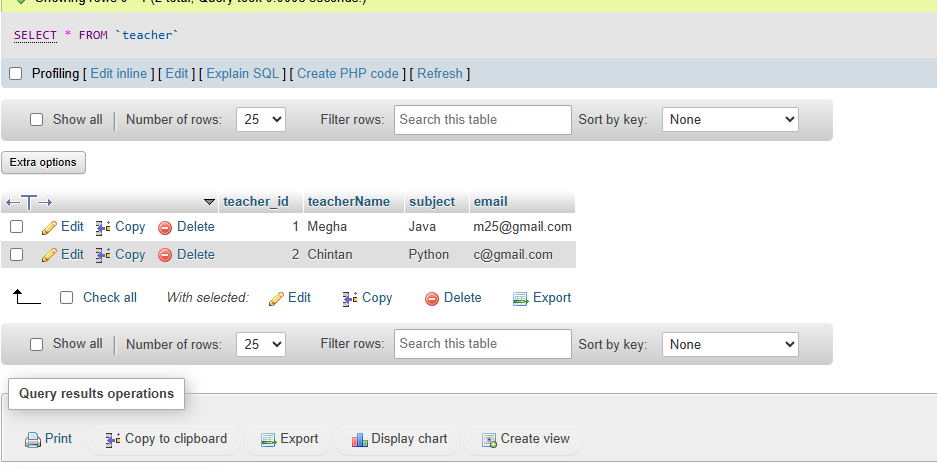
* SELECT student\_Name,age FROM students



* SELECT \* FROM students WHERE age>10



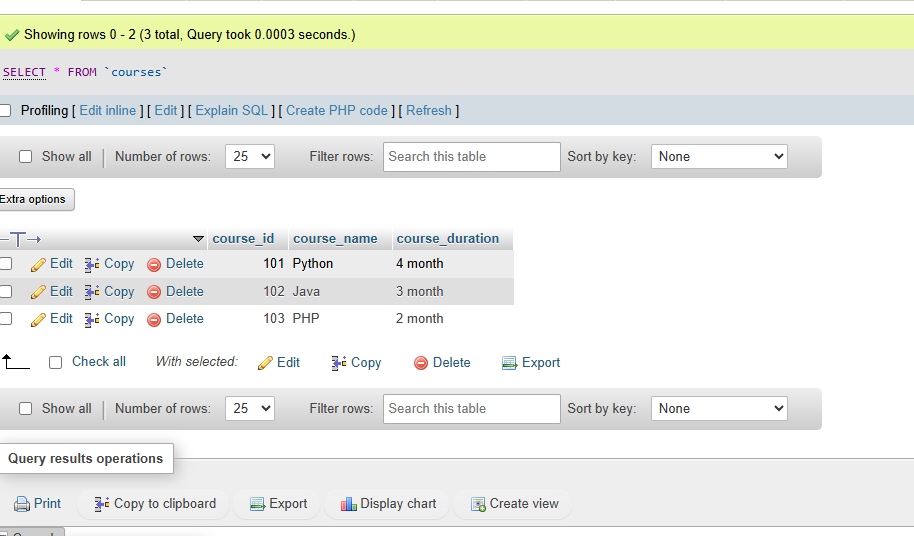
* CREATE TABLE teacher(teacher\_id int PRIMARY key AUTO\_INCREMENT,teacherName varchar(40)not null,subject varchar(40)not null , email varchar(20) UNIQUE);



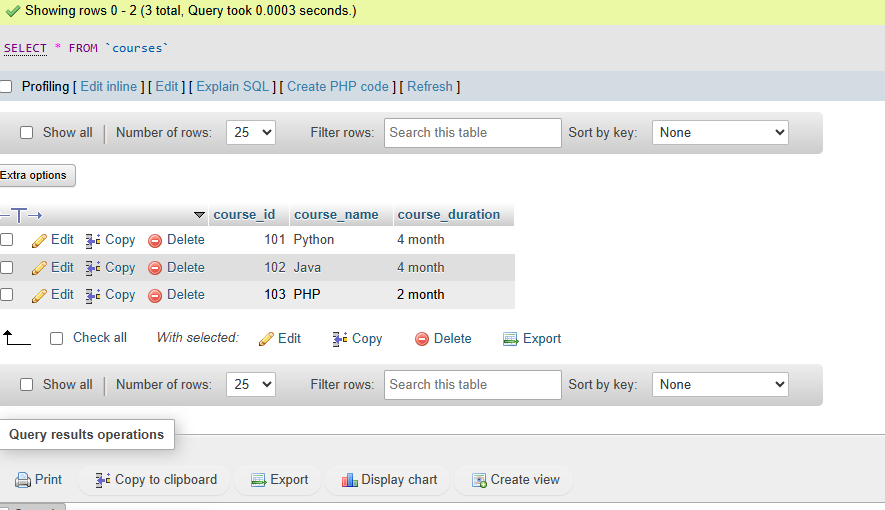
* alter table students add CONSTRAINT FOREIGN KEY (teacherId) REFERENCES teacher(teacher\_id);
* CREATE TABLE courses(course\_id int PRIMARY key,course\_name varchar(20),course\_credit int );
* alter table courses add COLUMN course\_duration;
* ALTER TABLE courses DROP COLUMN course\_credit
* drop table students
* drop table teacher
* INSERT INTO `courses`(`course\_id`, `course\_name`, `course\_duration`) VALUES (101,"Python", "4 month");

INSERT INTO `courses`(`course\_id`, `course\_name`, `course\_duration`) VALUES (102,"Java", "3 month");

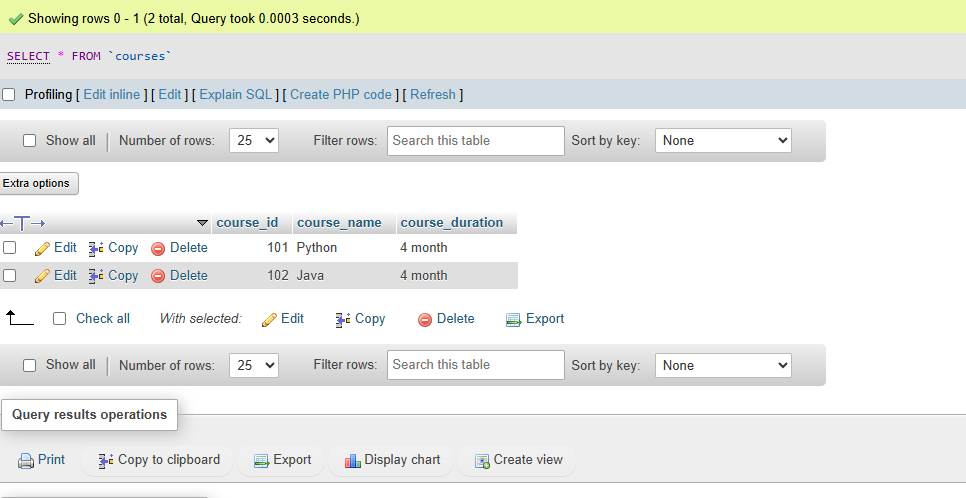
INSERT INTO `courses`(`course\_id`, `course\_name`, `course\_duration`) VALUES (103,"PHP", "2 month");



* UPDATE courses set `course\_duration`="4 month" where `course\_id`=102

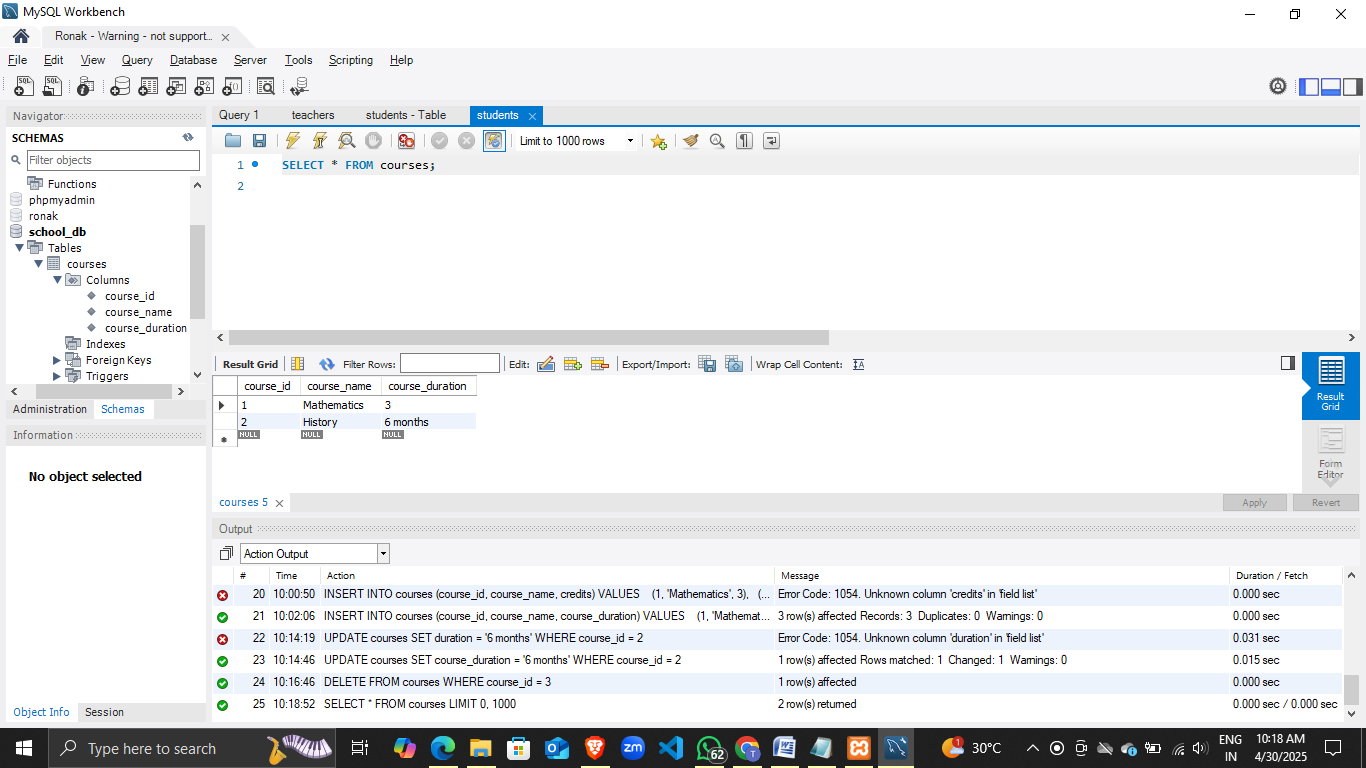


* DELETE FROM `courses` WHERE `course\_id`=103



Lab 1: Retrieve all courses from the courses table using the SELECT statement.

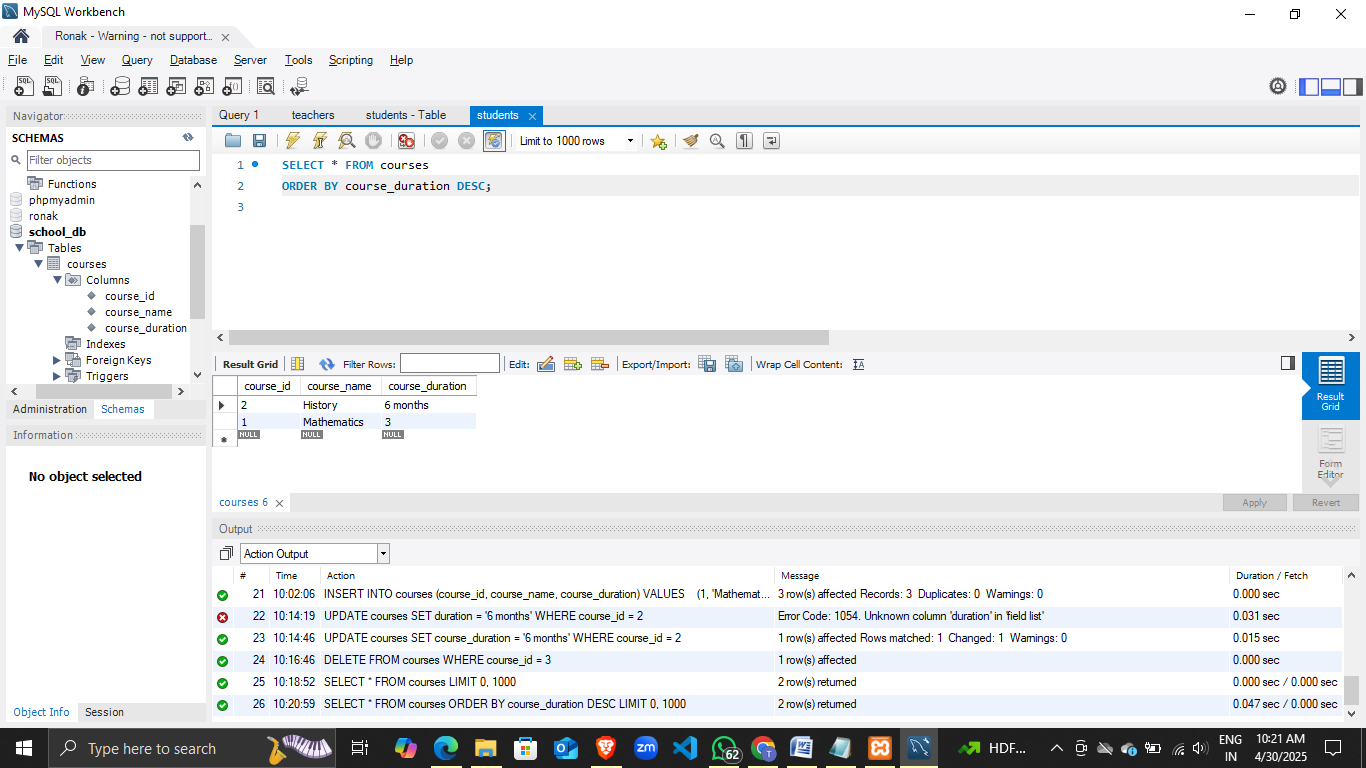
Ans. SELECT \* FROM courses;



Lab 2: Sort the courses based on course\_duration in descending order using ORDER BY.

Ans. SELECT \* FROM courses

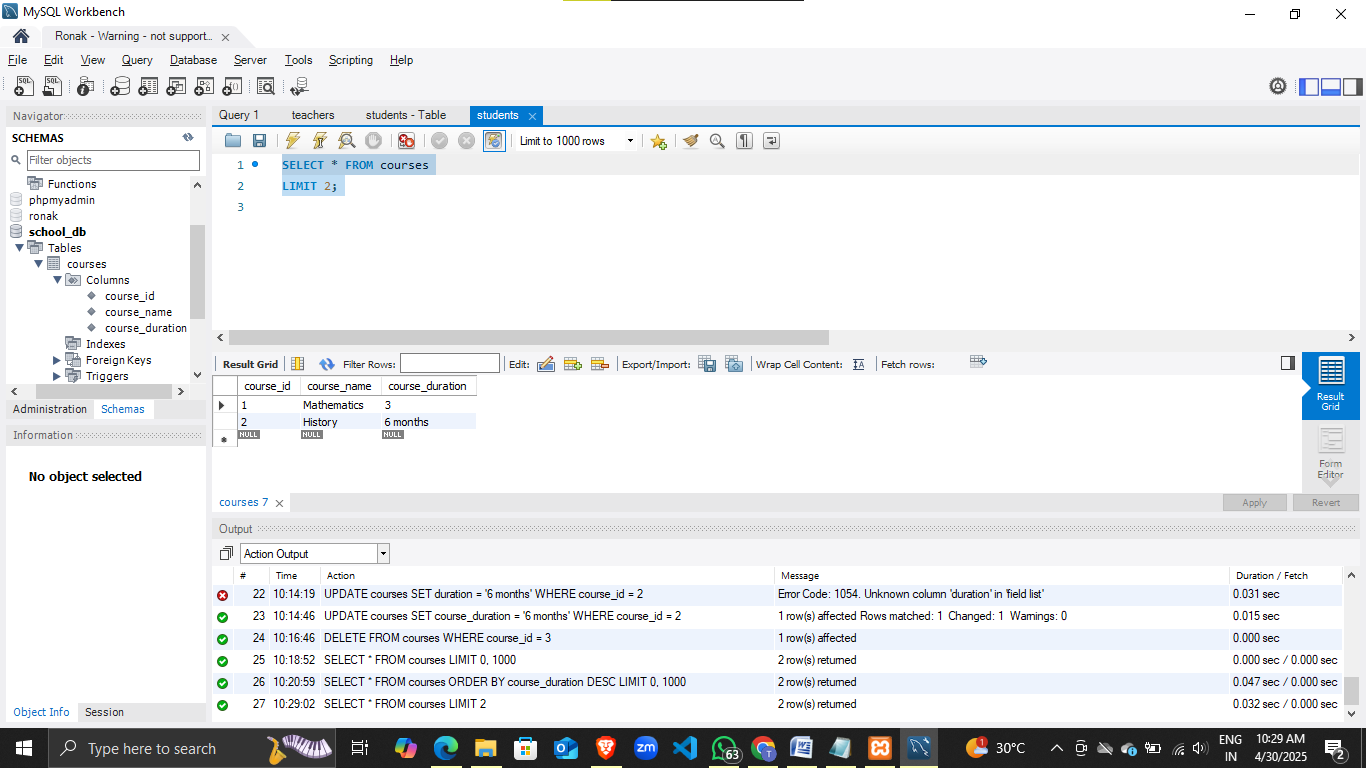
ORDER BY course\_duration DESC;



Lab 3: Limit the results of the SELECT query to show only the top two courses using LIMIT.

Ans. SELECT \* FROM courses

LIMIT 2;

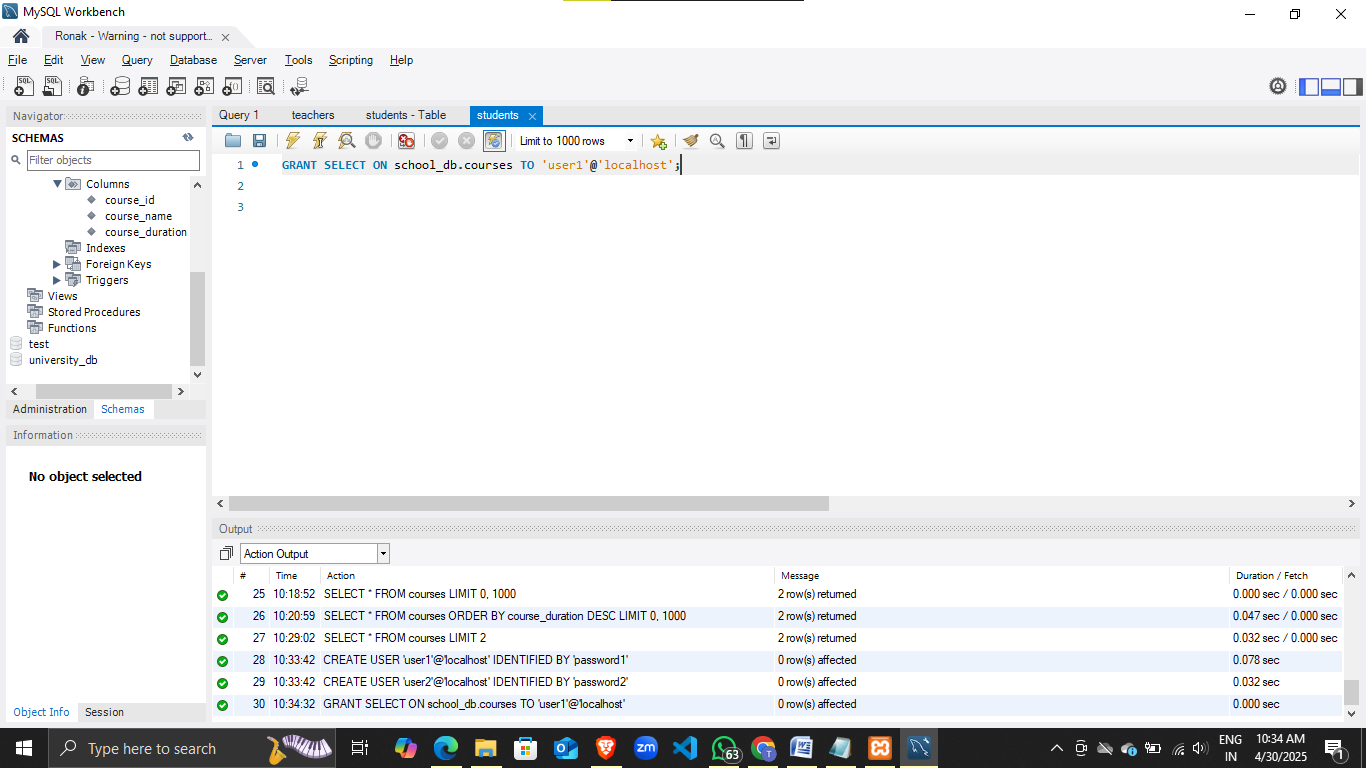


Lab 1: Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.

Ans. CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password1';

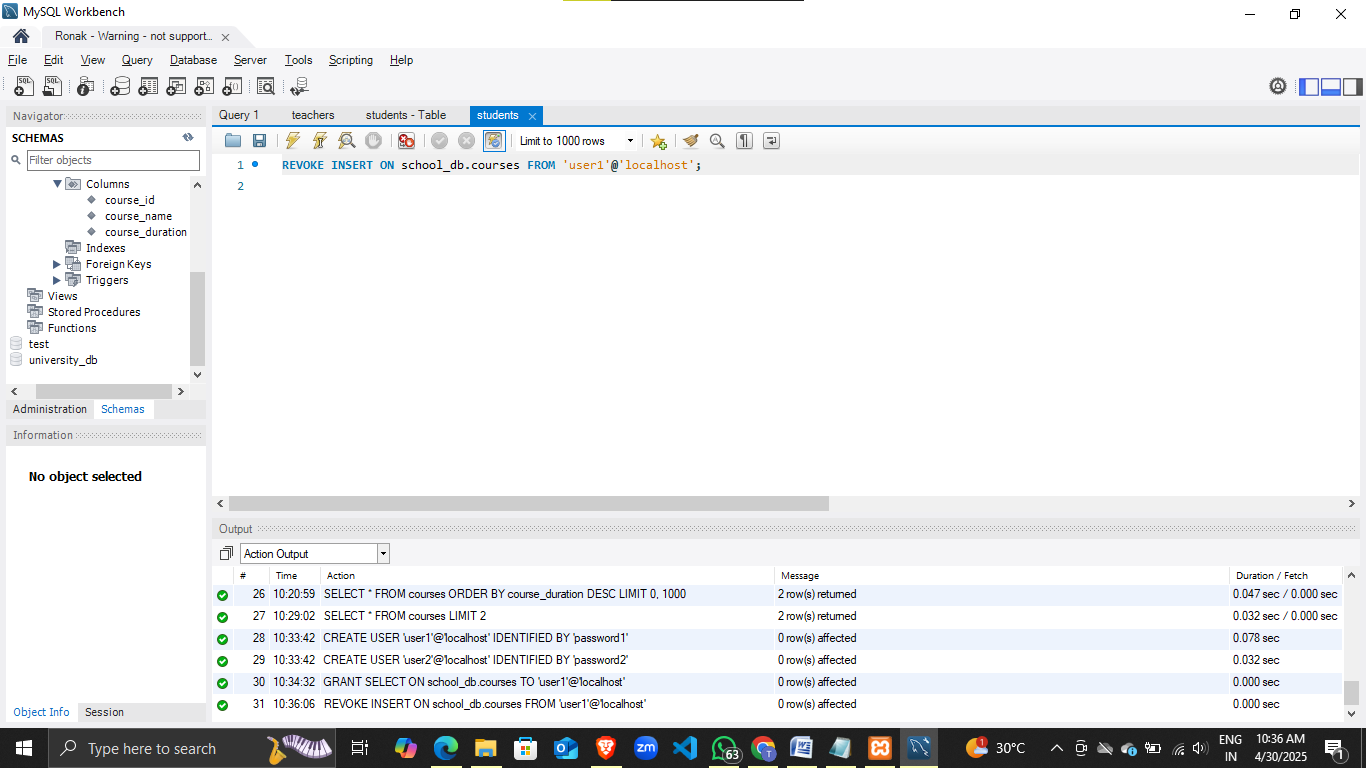
CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';

GRANT SELECT ON school\_db.courses TO 'user1'@'localhost';

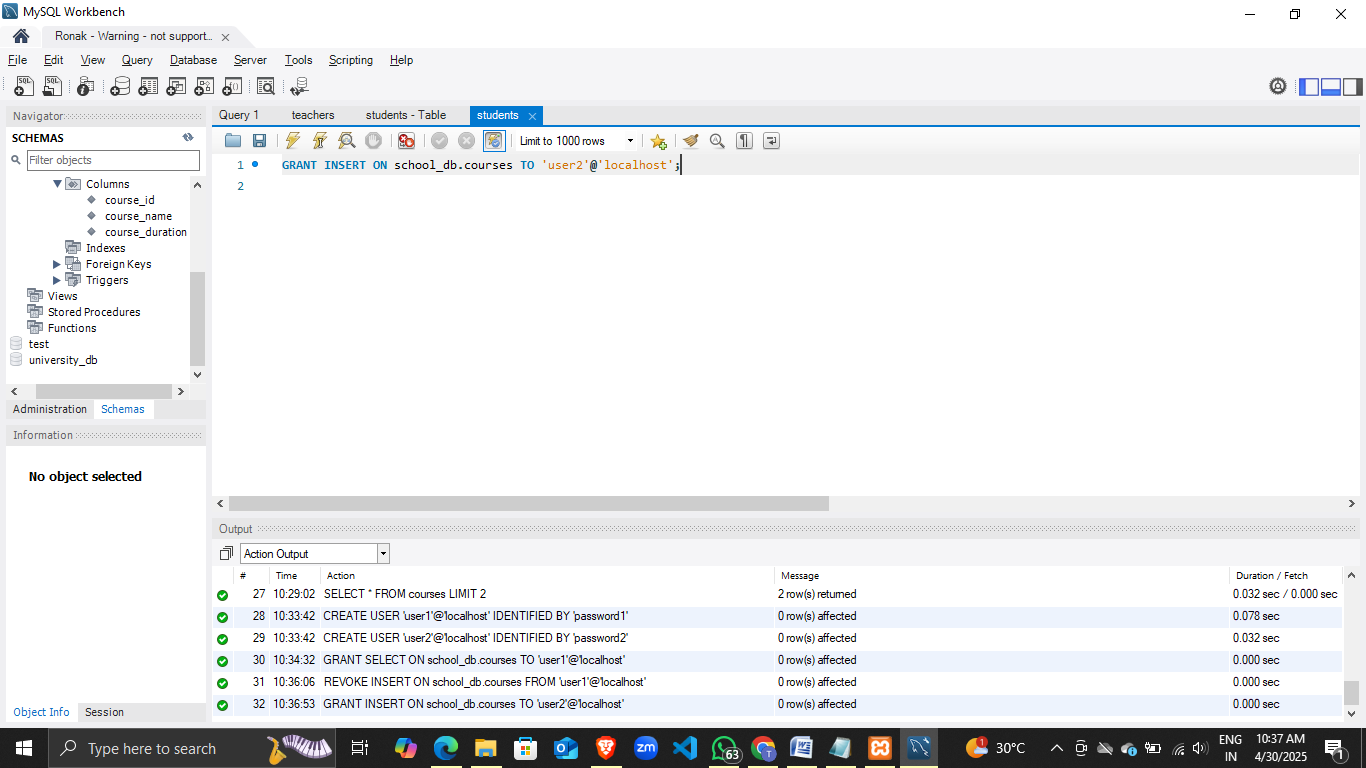


Lab 2: Revoke the INSERT permission from user1 and give it to user2.

Ans. REVOKE INSERT ON school\_db.courses FROM 'user1'@'localhost';



GRANT INSERT ON school\_db.courses TO 'user2'@'localhost';



Lab 1: Insert a few rows into the courses table and use COMMIT to save the changes.

Ans. START TRANSACTION;

INSERT INTO courses (course\_id, course\_name, course\_duration)

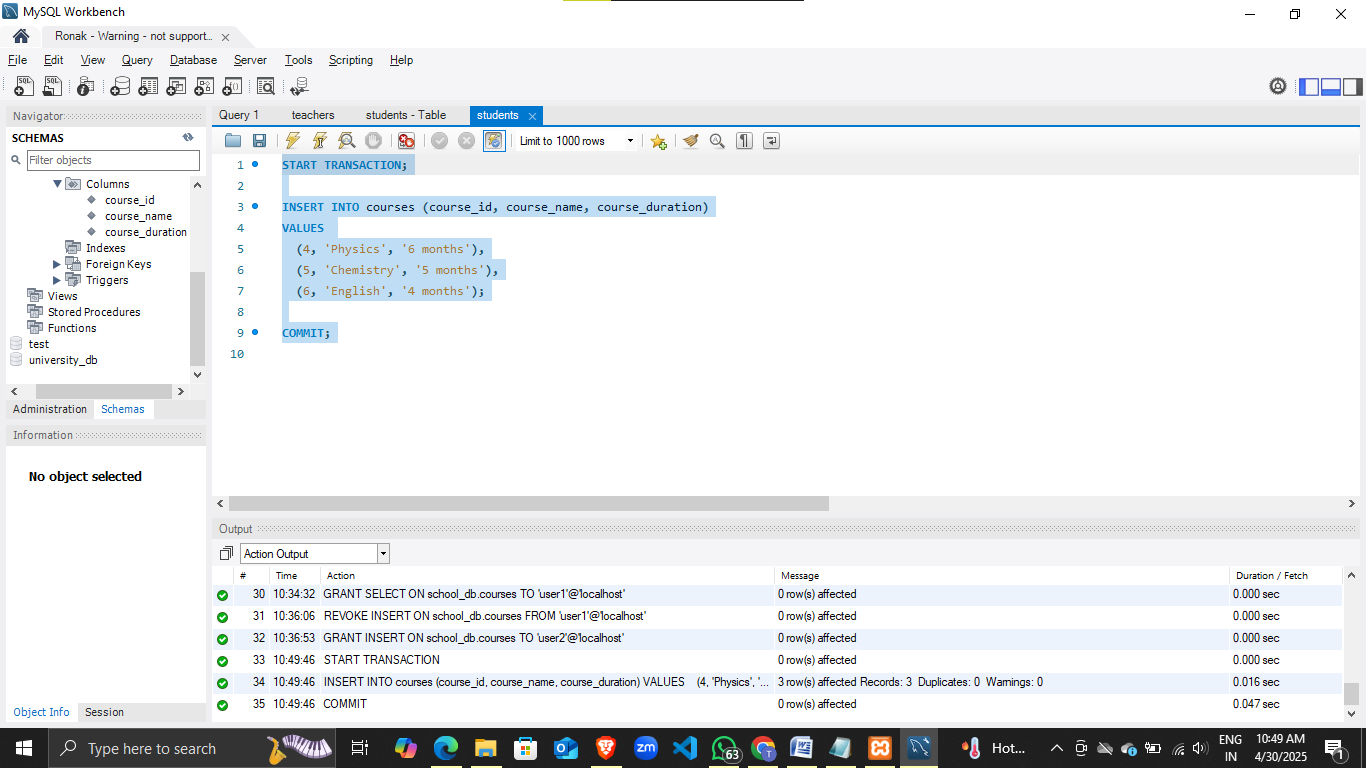
VALUES

(4, 'Physics', '6 months'),

(5, 'Chemistry', '5 months'),

(6, 'English', '4 months');

COMMIT;



Lab 2: Insert additional rows, then use ROLLBACK to undo the last insert operation.

Ans. START TRANSACTION;

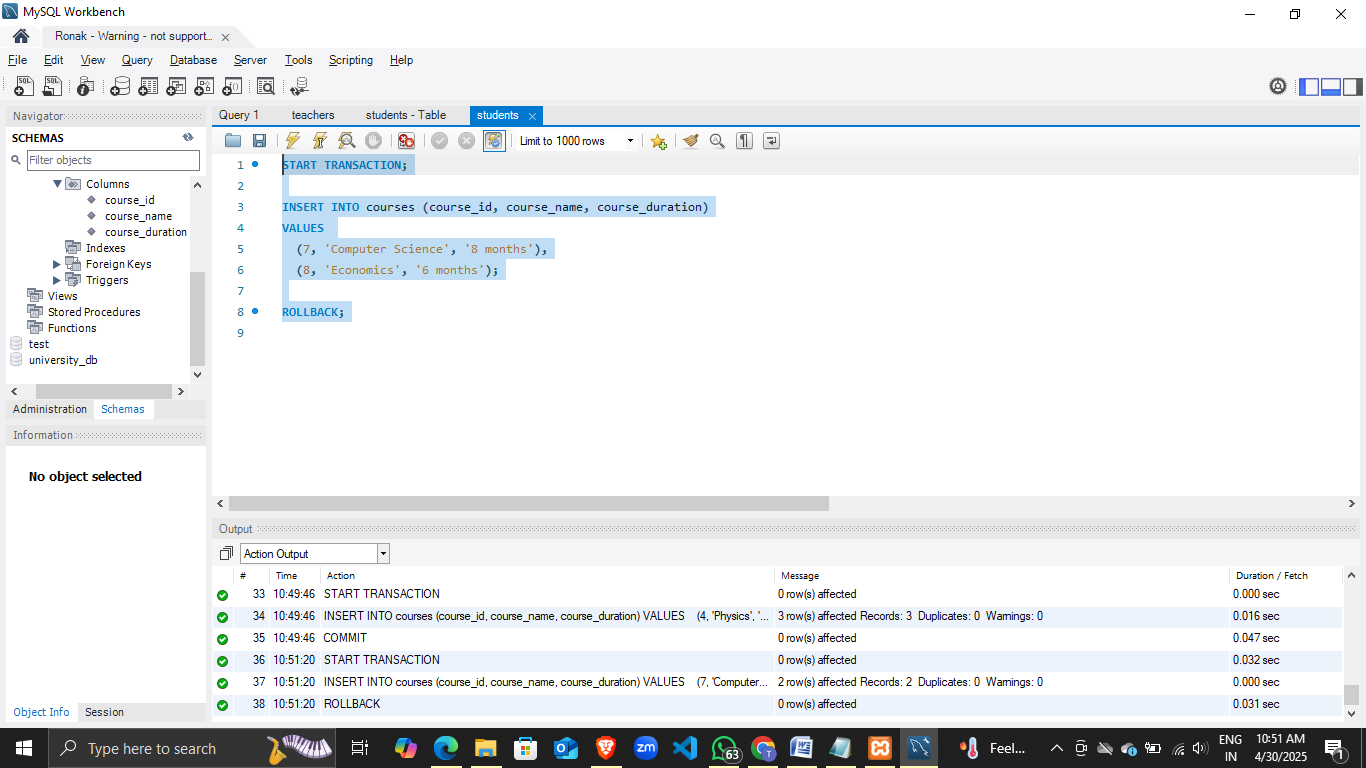
INSERT INTO courses (course\_id, course\_name, course\_duration)

VALUES

(7, 'Computer Science', '8 months'),

(8, 'Economics', '6 months');

ROLLBACK;



Lab 3: Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.

Ans. START TRANSACTION;

SAVEPOINT before\_update;

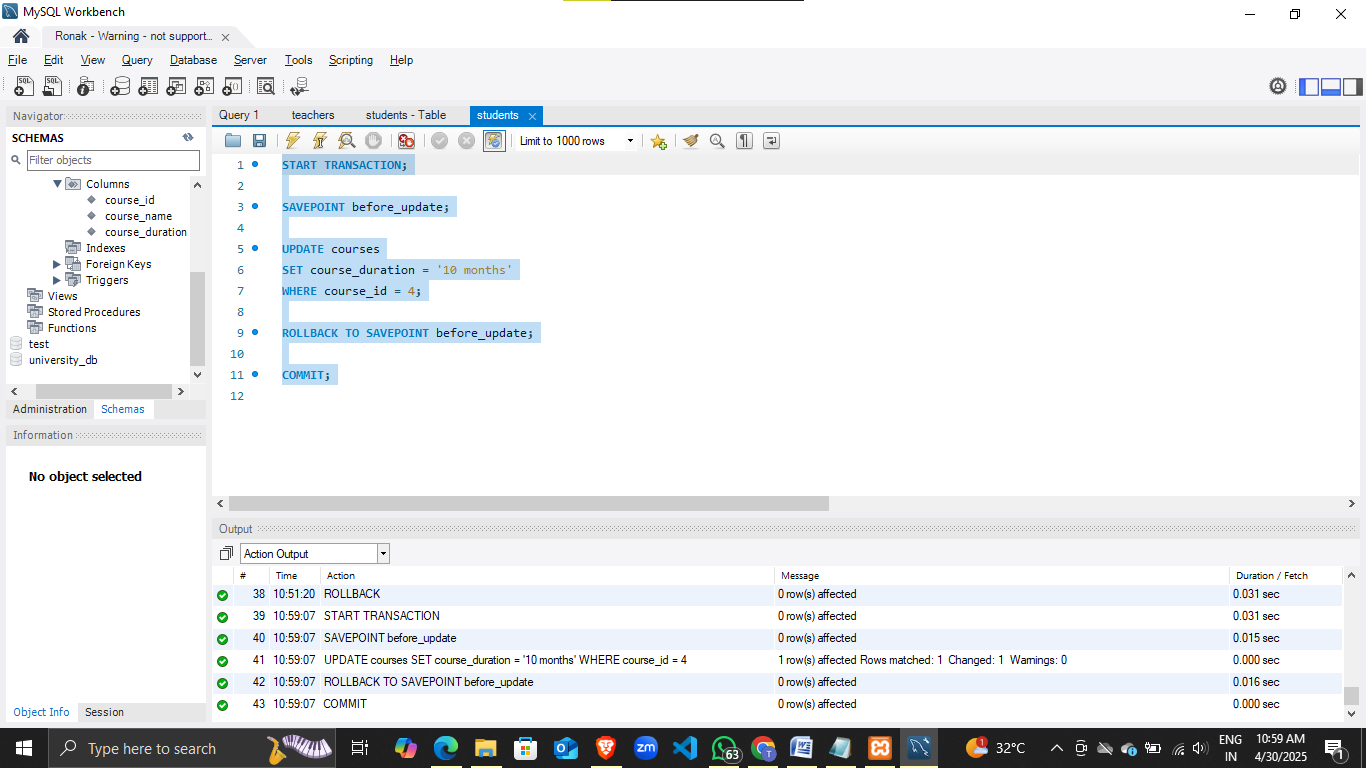
UPDATE courses

SET course\_duration = '10 months'

WHERE course\_id = 4;

ROLLBACK TO SAVEPOINT before\_update;

COMMIT;



Lab 1: Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.

Ans. CREATE TABLE departments (

department\_id INT PRIMARY KEY,

department\_name VARCHAR(100)

);

CREATE TABLE departments (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(50)

);

-- Create employees table

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(50),

dept\_id INT,

FOREIGN KEY (dept\_id) REFERENCES departments(dept\_id)

);

-- Insert sample data

INSERT INTO departments (dept\_id, dept\_name) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT'),

(4, 'Sales');

INSERT INTO employees (emp\_id, emp\_name, dept\_id) VALUES

(101, 'Amit Kumar', 1),

(102, 'Priya Sharma', 2),

(103, 'Rahul Mehta', 3),

(104, 'Sneha Verma', 3);

-- INNER JOIN: show only employees who belong to departments

SELECT e.emp\_id, e.emp\_name, d.dept\_name

FROM employees e

INNER JOIN departments d

ON e.dept\_id = d.dept\_id;

Lab 2: Use a LEFT JOIN to show all departments, even those without employees.

sql

Copy code

SELECT d.dept\_id, d.dept\_name, e.emp\_id, e.emp\_name

FROM departments d

LEFT JOIN employees e

ON d.dept\_id = e.dept\_id;

**Module 12: SQL Group By**

**Lab 1:** Group employees by department and count employees.

SELECT dept\_id, COUNT(emp\_id) AS total\_employees

FROM employees

GROUP BY dept\_id;

**Lab 2:** Find the average salary of employees in each department.

SELECT dept\_id, AVG(salary) AS avg\_salary

FROM employees

GROUP BY dept\_id;

**Module 13: SQL Stored Procedure**

**Lab 1:** Retrieve all employees by department.

DELIMITER //

CREATE PROCEDURE GetEmployeesByDept(IN dept INT)

BEGIN

SELECT \* FROM employees WHERE dept\_id = dept;

END //

DELIMITER ;

**Lab 2:** Retrieve course details by course\_id.

DELIMITER //

CREATE PROCEDURE GetCourseDetails(IN cid INT)

BEGIN

SELECT \* FROM courses WHERE course\_id = cid;

END //

DELIMITER ;

**Module 14: SQL View**

**Lab 1:** Show employees with department names.

CREATE VIEW emp\_dept\_view AS

SELECT e.emp\_name, d.dept\_name

FROM employees e

JOIN departments d

ON e.dept\_id = d.dept\_id;

**Lab 2:** Exclude employees earning below 50,000.

CREATE OR REPLACE VIEW emp\_dept\_view AS

SELECT e.emp\_name, d.dept\_name

FROM employees e

JOIN departments d

ON e.dept\_id = d.dept\_id

WHERE e.salary >= 50000;

**Module 15: SQL Triggers**

**Lab 1:** Log changes when a new employee is added.

CREATE TABLE employee\_log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

emp\_id INT,

action VARCHAR(50),

log\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TRIGGER after\_employee\_insert

AFTER INSERT ON employees

FOR EACH ROW

INSERT INTO employee\_log(emp\_id, action)

VALUES (NEW.emp\_id, 'Inserted');

**Lab 2:** Update last\_modified on employee update.

ALTER TABLE employees ADD last\_modified TIMESTAMP;

CREATE TRIGGER before\_employee\_update

BEFORE UPDATE ON employees

FOR EACH ROW

SET NEW.last\_modified = NOW();

**Module 16: Introduction to PL/SQL**

**Lab 1:** Count total employees.

DECLARE

total\_employees INT;

BEGIN

SELECT COUNT(\*) INTO total\_employees FROM employees;

DBMS\_OUTPUT.PUT\_LINE('Total Employees: ' || total\_employees);

END;

**Lab 2:** Calculate total sales from orders table.

DECLARE

total\_sales NUMBER;

BEGIN

SELECT SUM(amount) INTO total\_sales FROM orders;

DBMS\_OUTPUT.PUT\_LINE('Total Sales: ' || total\_sales);

END;

**Module 17: PL/SQL Control Structures**

**Lab 1:** IF condition on employee department.

DECLARE

v\_dept VARCHAR2(50);

BEGIN

SELECT dept\_id INTO v\_dept FROM employees WHERE emp\_id = 101;

IF v\_dept = 'IT' THEN

DBMS\_OUTPUT.PUT\_LINE('Employee is in IT Department');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Employee is not in IT Department');

END IF;

END;

**Lab 2:** FOR LOOP to display employee names.

BEGIN

FOR rec IN (SELECT emp\_name FROM employees) LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee: ' || rec.emp\_name);

END LOOP;

END;

**Module 18: SQL Cursors**

**Lab 1:** Explicit cursor for employees.

DECLARE

CURSOR emp\_cursor IS SELECT emp\_id, emp\_name FROM employees;

v\_id employees.emp\_id%TYPE;

v\_name employees.emp\_name%TYPE;

BEGIN

OPEN emp\_cursor;

LOOP

FETCH emp\_cursor INTO v\_id, v\_name;

EXIT WHEN emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('ID: ' || v\_id || ' Name: ' || v\_name);

END LOOP;

CLOSE emp\_cursor;

END;

**Lab 2:** Cursor for courses.

DECLARE

CURSOR c\_cursor IS SELECT course\_name FROM courses;

v\_course VARCHAR2(50);

BEGIN

OPEN c\_cursor;

LOOP

FETCH c\_cursor INTO v\_course;

EXIT WHEN c\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Course: ' || v\_course);

END LOOP;

CLOSE c\_cursor;

END;

**Module 19: Rollback and Commit Savepoint**

**Lab 1:** Rollback to savepoint.

START TRANSACTION;

INSERT INTO courses VALUES (106, 'Biology', 3, 4);

SAVEPOINT save1;

INSERT INTO courses VALUES (107, 'Chemistry', 3, 5);

ROLLBACK TO save1;

**Lab 2:** Commit part, rollback rest.

START TRANSACTION;

INSERT INTO courses VALUES (108, 'Economics', 3, 4);

SAVEPOINT save2;

INSERT INTO courses VALUES (109, 'Geography', 2, 3);

COMMIT;

ROLLBACK TO save2;

**📌 Extra Lab Practice (Library DB)**

**Extra SQL – Introduction to SQL**

**Lab 3:** Create library\_db with books.

CREATE DATABASE library\_db;

USE library\_db;

CREATE TABLE books (

book\_id INT PRIMARY KEY,

title VARCHAR(100),

author VARCHAR(50),

publisher VARCHAR(50),

year\_of\_publication INT,

price DECIMAL(8,2)

);

INSERT INTO books VALUES

(1, 'DBMS Concepts', 'Korth', 'McGrawHill', 2015, 550),

(2, 'SQL Fundamentals', 'James', 'Pearson', 2018, 650),

(3, 'Oracle PL/SQL', 'Scott', 'Wiley', 2012, 700),

(4, 'Database Systems', 'Navathe', 'Pearson', 2020, 800),

(5, 'SQL Queries', 'Martin', 'O’Reilly', 2019, 600);

**Lab 4:** Create members table.

CREATE TABLE members (

member\_id INT PRIMARY KEY,

member\_name VARCHAR(50),

date\_of\_membership DATE,

email VARCHAR(100)

);

INSERT INTO members VALUES

(1, 'Anita', '2020-01-15', 'anita@gmail.com'),

(2, 'Ravi', '2019-07-12', 'ravi@yahoo.com'),

(3, 'Meera', '2021-03-20', 'meera@hotmail.com'),

(4, 'Ajay', '2018-11-05', 'ajay@gmail.com'),

(5, 'Kiran', '2022-06-10', 'kiran@gmail.com');