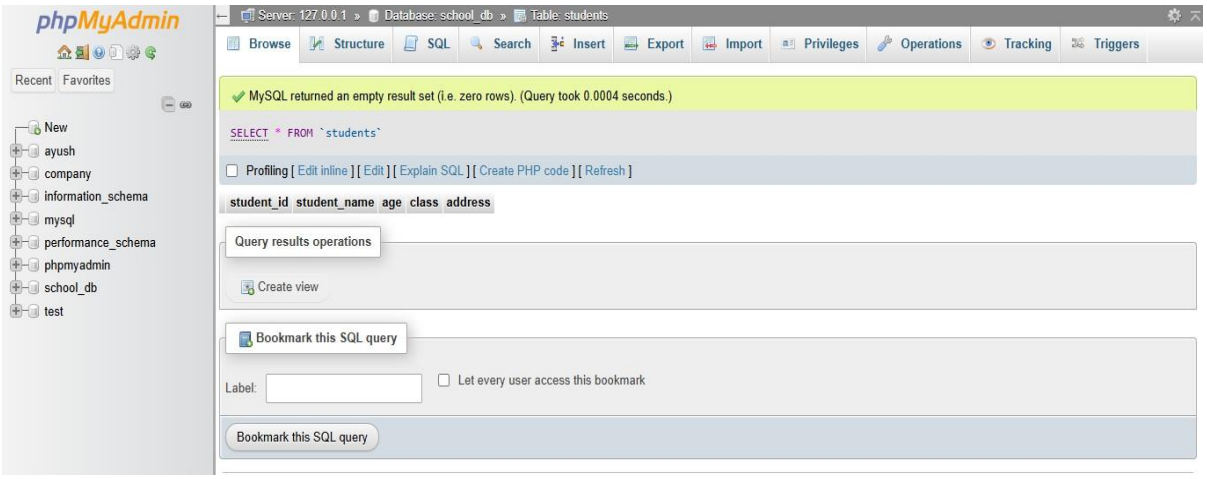
Name : Dhaval Jethva

Module 4 – Introduction to DBMS

1). Introduction to SQL

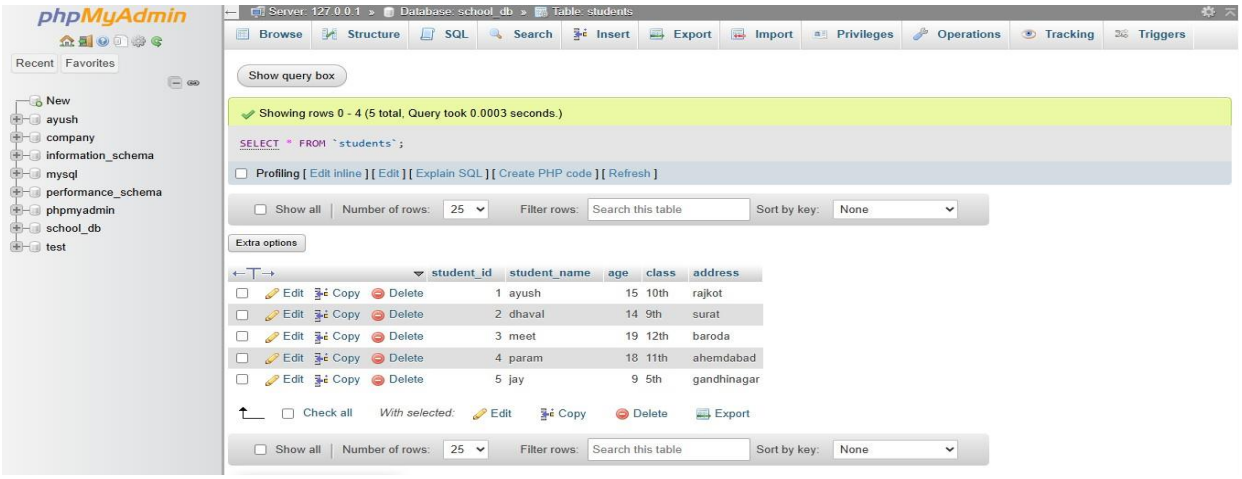
Lab 1 : Create a new database named school_db and a table called students with the following columns: student_id, student_name, age, class, and address.

- CREATE DATABASE school_db
- CREATE TABLE students(student_id int PRIMARY KEY AUTO_INCREMENT,student_name varchar(20),age int,class varchar(20),address varchar(20))



Lab 2 : Insert five records into the students table and retrieve all records using the SELECT statement.

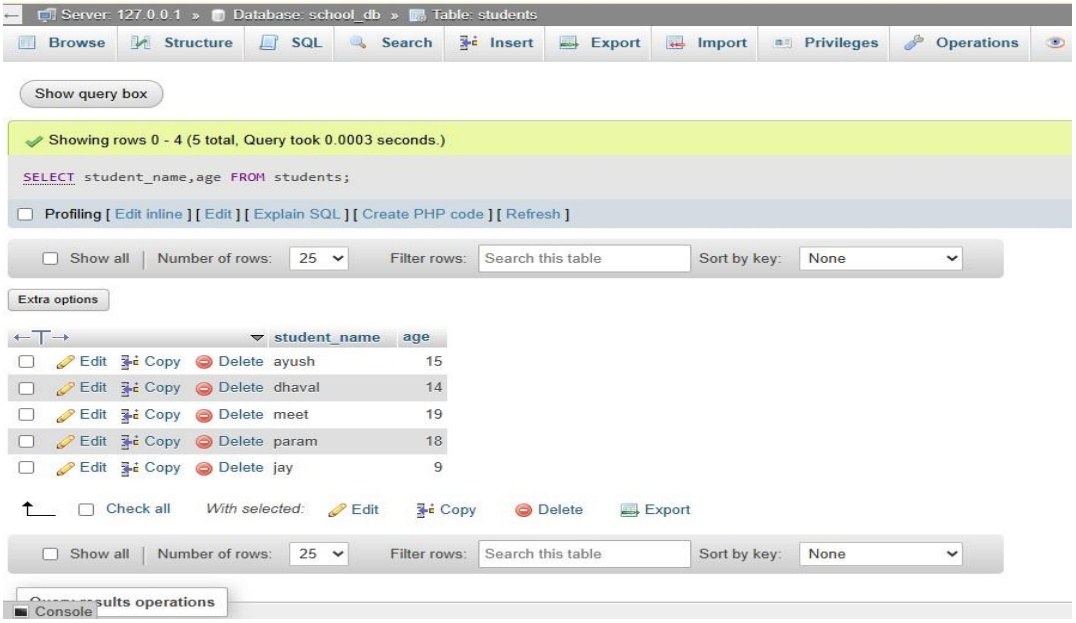
- INSERT INTO students(student_name,age,class,address) VALUES('ayush',15,'10th','rajkot');
- SELECT * FROM `students`



2. SQL Syntax

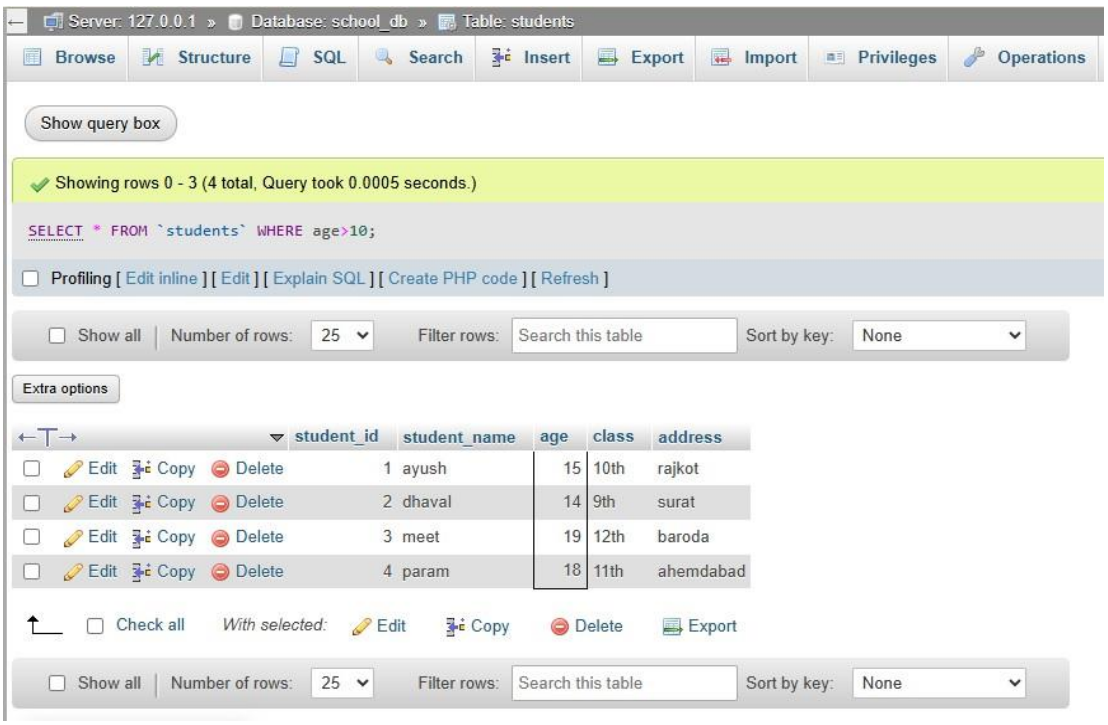
Lab 1 : Write SQL queries to retrieve specific columns (student_name and age) from the students table.

- SELECT student_name,age FROM students



Lab 2 : Write SQL queries to retrieve all students whose age is greater than 10.

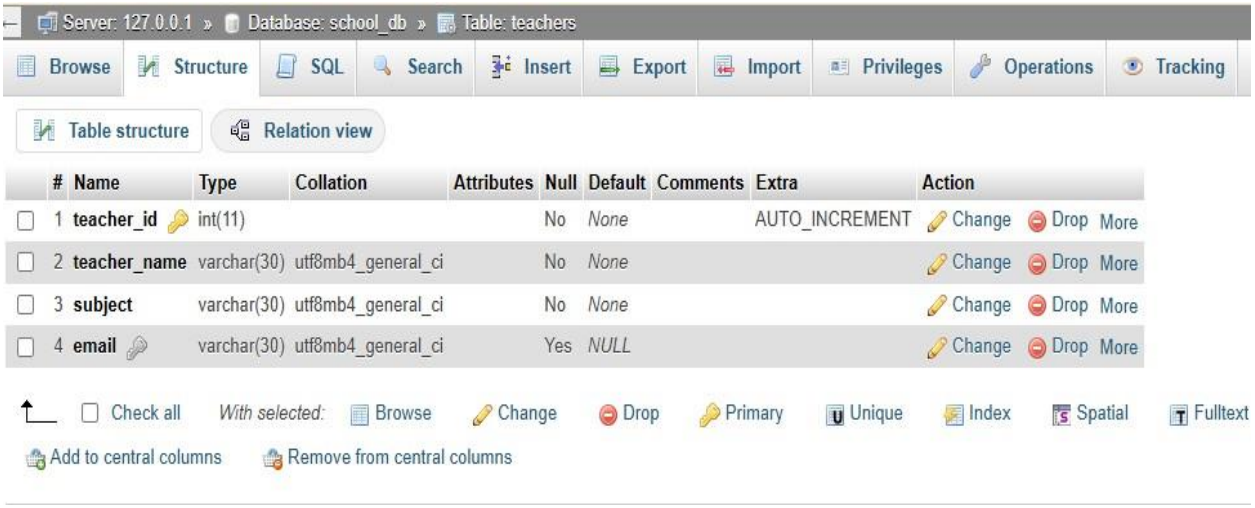
- SELECT * FROM `students` WHERE age>10



3. SQL Constraints

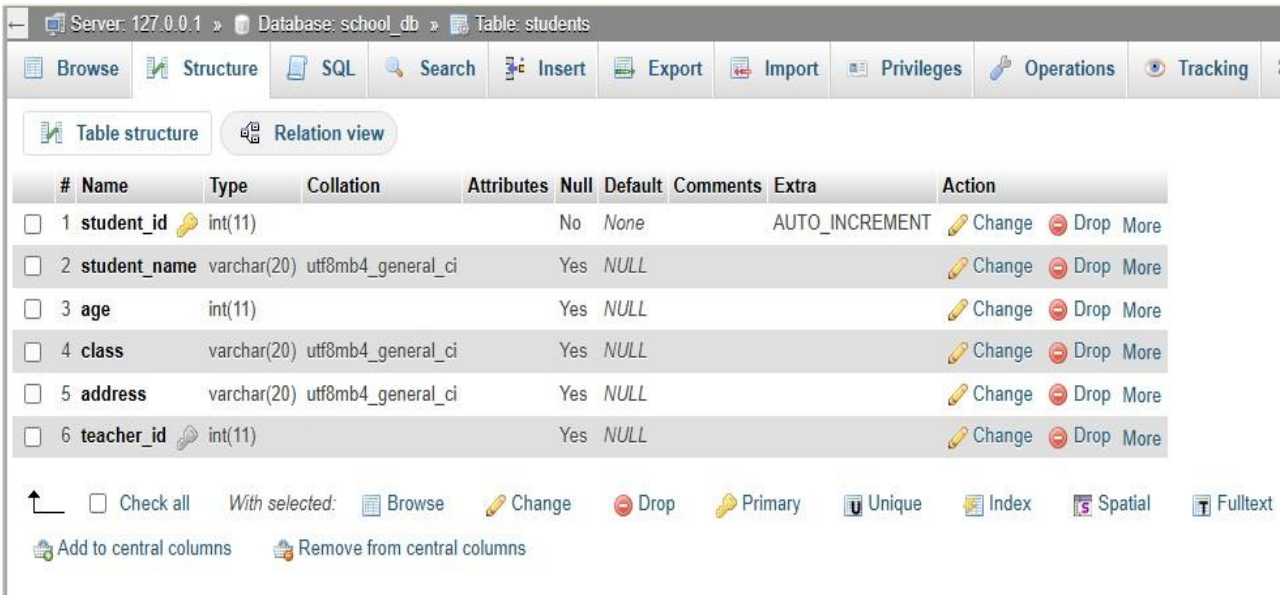
Lab 1 : Create a table teachers with the following columns: teacher_id (Primary Key), teacher_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).

- CREATE TABLE teachers (teacher_id int PRIMARY KEY AUTO_INCREMENT,teacher_name varchar(30) NOT NULL,subject varchar(30) NOT NULL,email varchar(30) UNIQUE)



Lab 2 : Implement a FOREIGN KEY constraint to relate the teacher_id from the teachers table with the students table.

- ALTER TABLE students ADD CONSTRAINT fk_students_teacher FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);

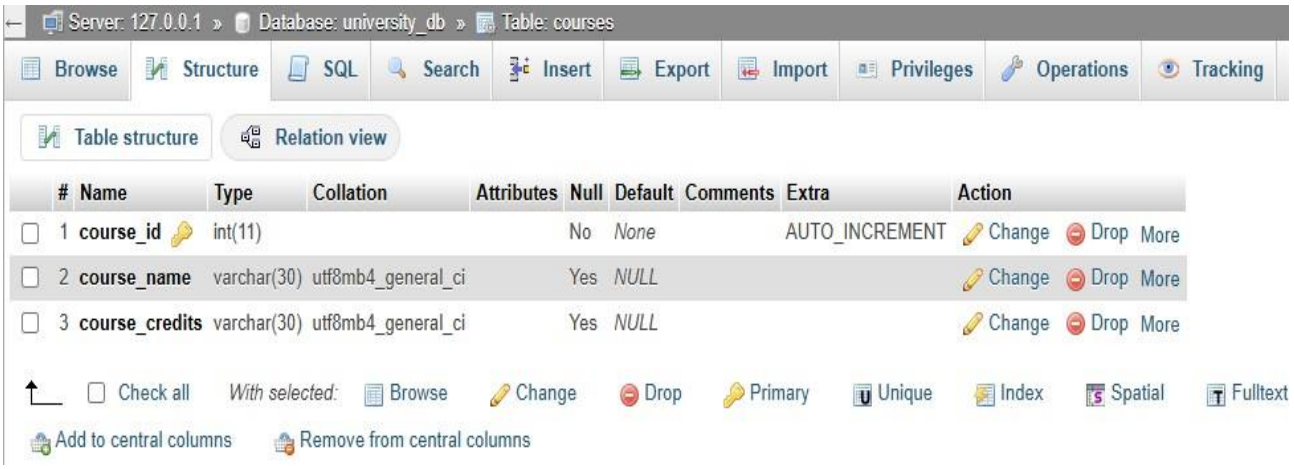


4. Main SQL Commands and Sub-commands (DDL)

Lab 1 : Create a table courses with columns: course_id, course_name, and course_credits. Set the course_id as the primary key.

Lab 2 : Use the CREATE command to create a database university_db.

- CREATE DATABASE university_db
- CREATE TABLE courses(course_id int PRIMARY KEY AUTO_INCREMENT,course_name varchar(30),course_credits varchar(30))



5. ALTER Command

Lab 1 : Modify the courses table by adding a column course_duration using the ALTER command.

Lab 2 : Drop the course_credits column from the courses table.

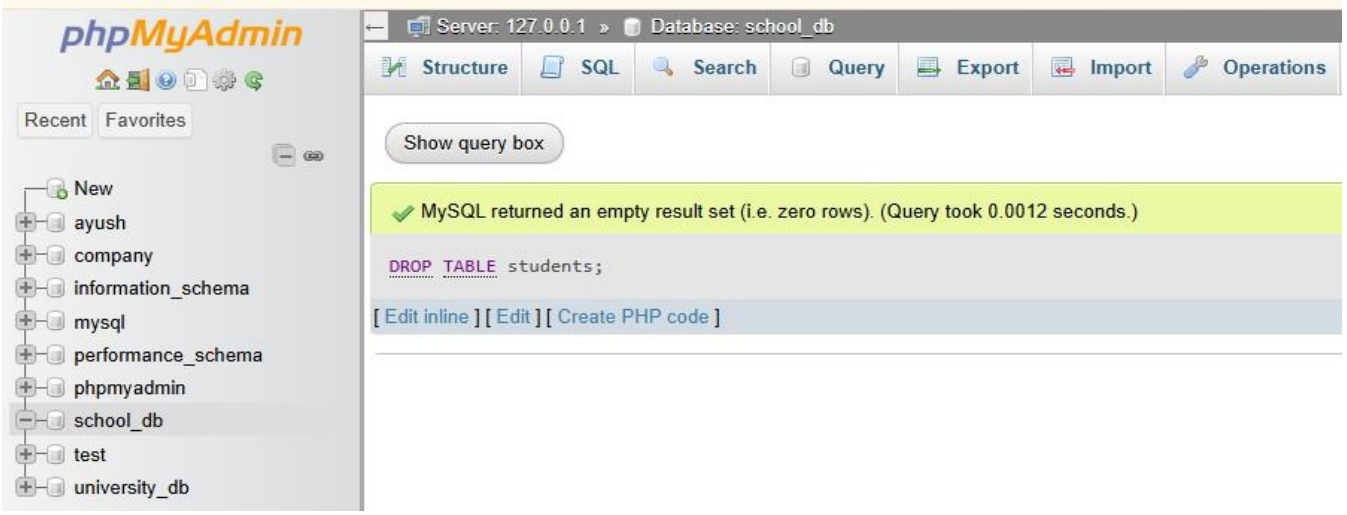
- ALTER TABLE courses ADD course_duration varchar(20)
- ALTER TABLE courses DROP course_credits

6. DROP Command

Lab 1 : Drop the teachers table from the school_db database.

Lab 2 : Drop the students table from the school_db database and verify that the table has been removed.

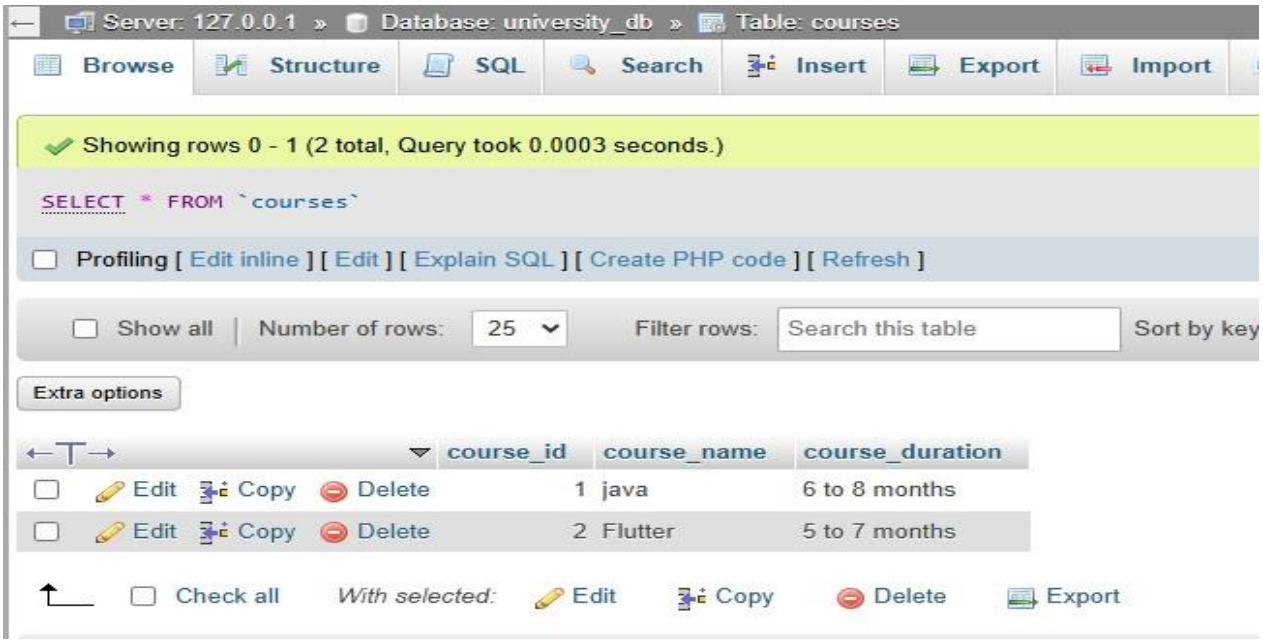
- DROP TABLE teachers
- DROP TABLE students



7. Data Manipulation Language (DML)

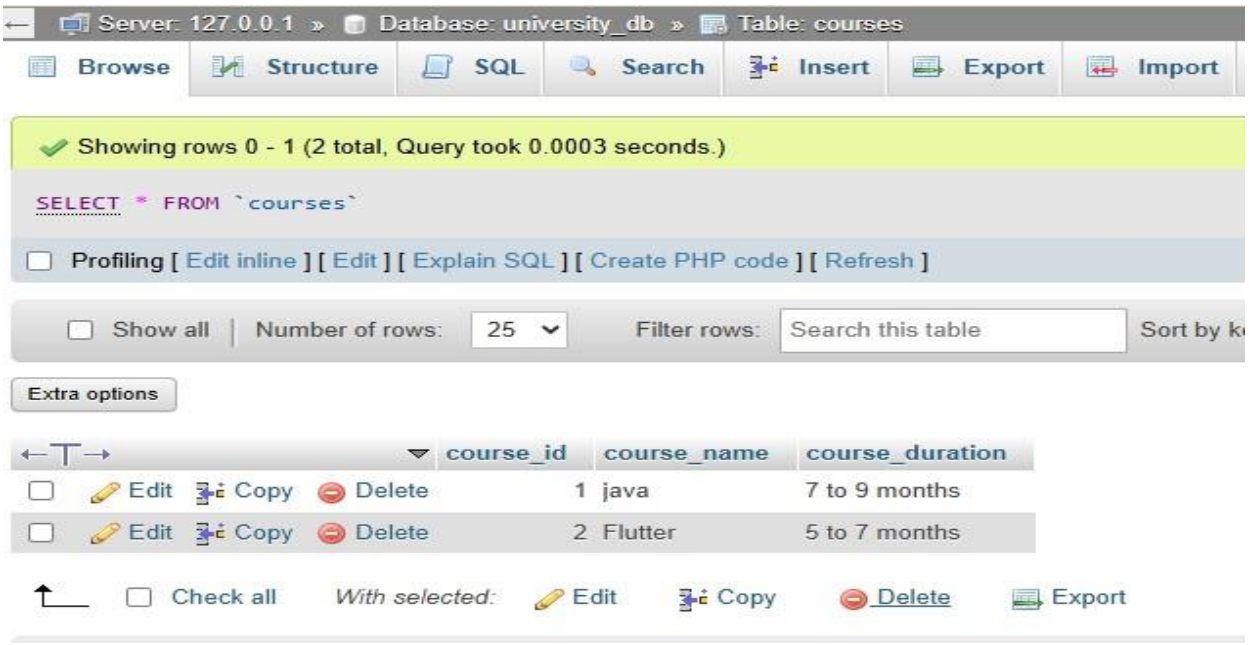
Lab 1 : Insert three records into the courses table using the INSERT command.

- INSERT INTO courses (course_name,course_duration) VALUES ('java','6 to 8 months')



Lab 2 : Update the course duration of a specific course using the UPDATE command.

- UPDATE courses SET course_duration = '7 to 9 months' WHERE course_id = 1



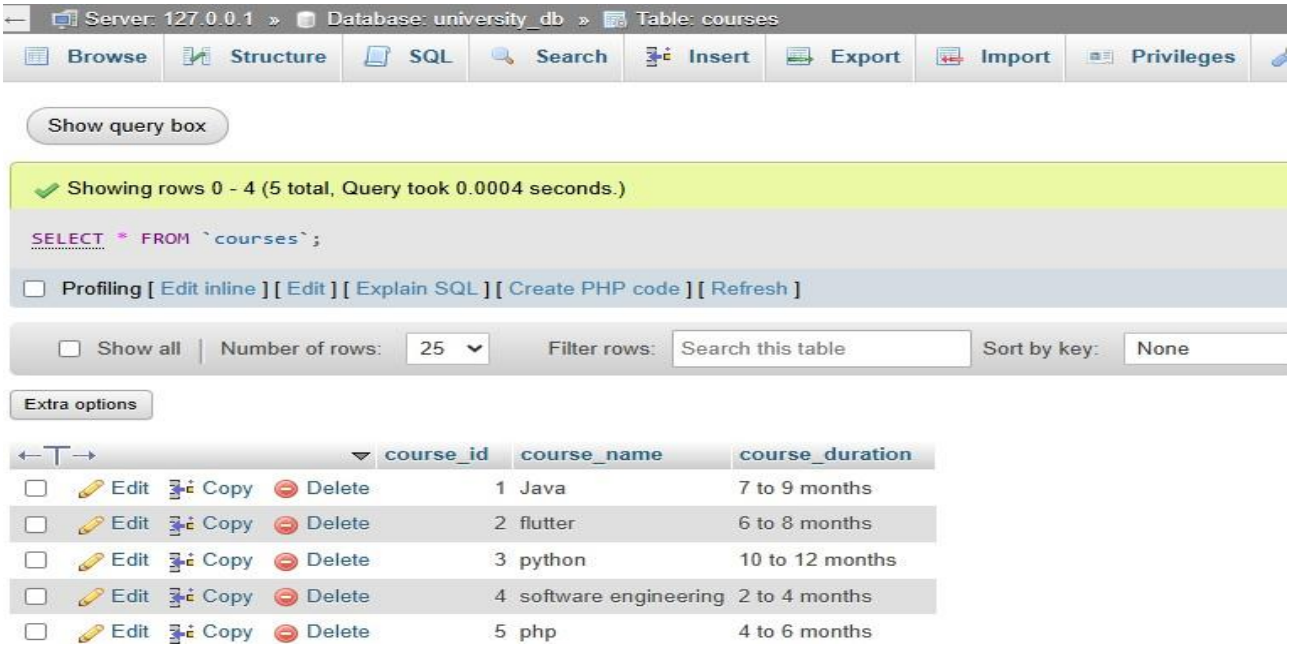
Lab 3 : Delete a course with a specific course_id from the courses table using the DELETE command.

- DELETE FROM courses WHERE course_id = 1

8. Data Query Language (DQL)

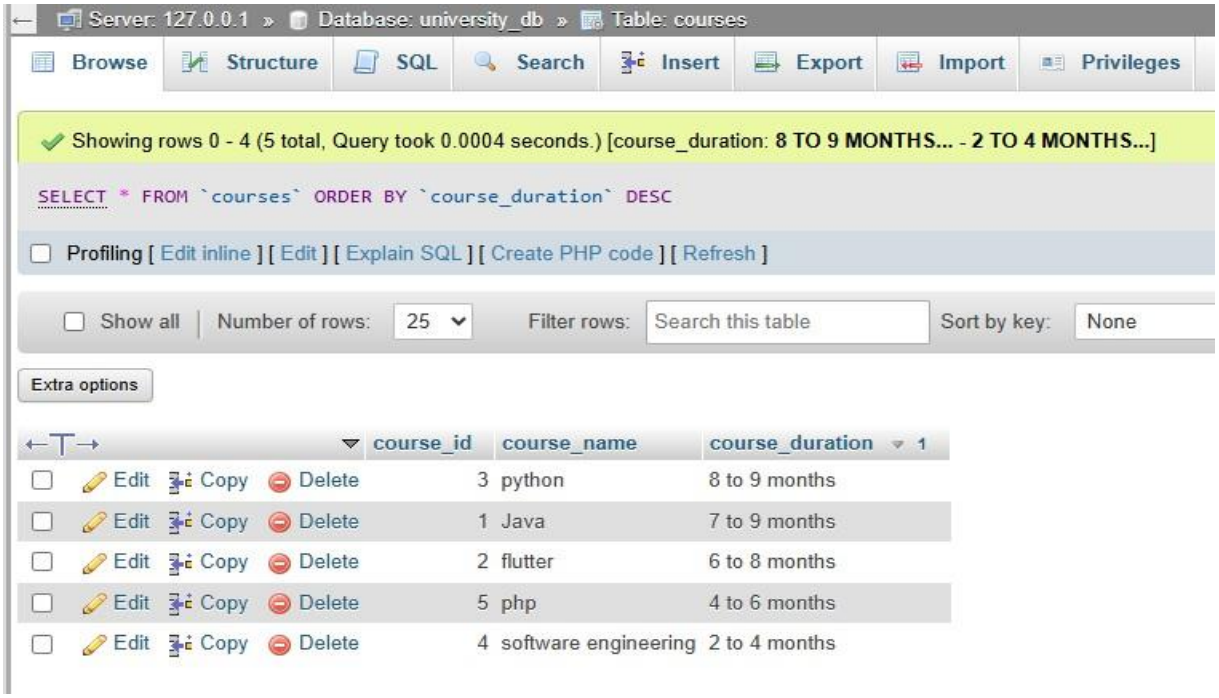
Lab 1 : Retrieve all courses from the courses table using the SELECT statement.

- SELECT * FROM `courses`



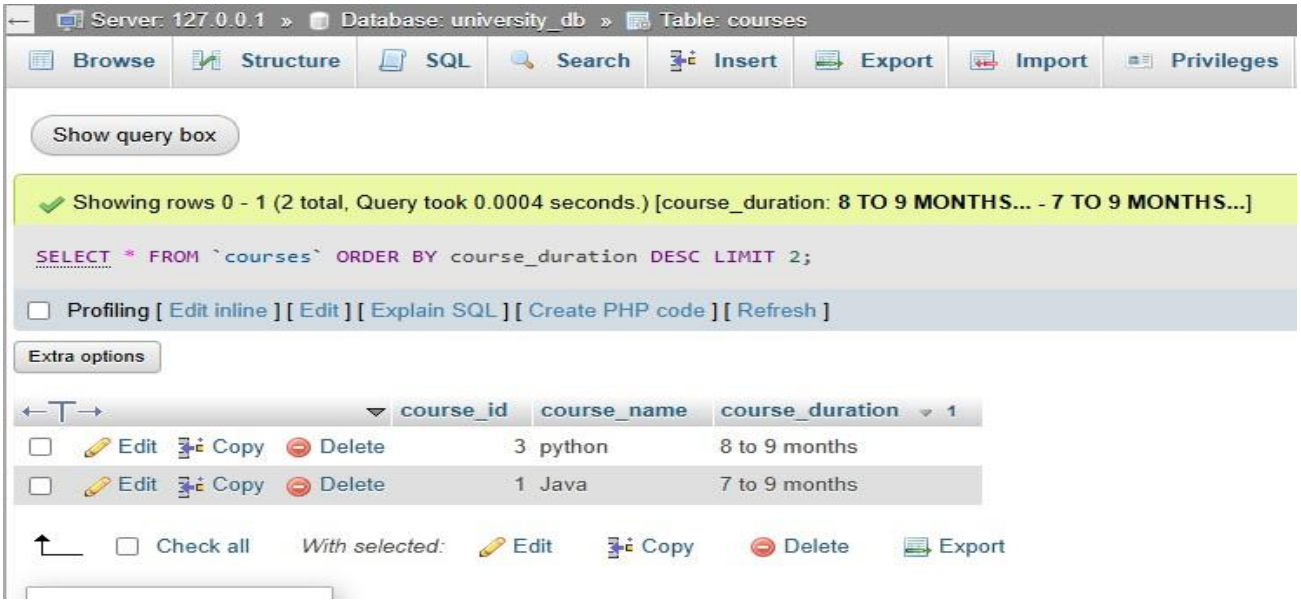
Lab 2 : Sort the courses based on course_duration in descending order using ORDER BY.

- 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.

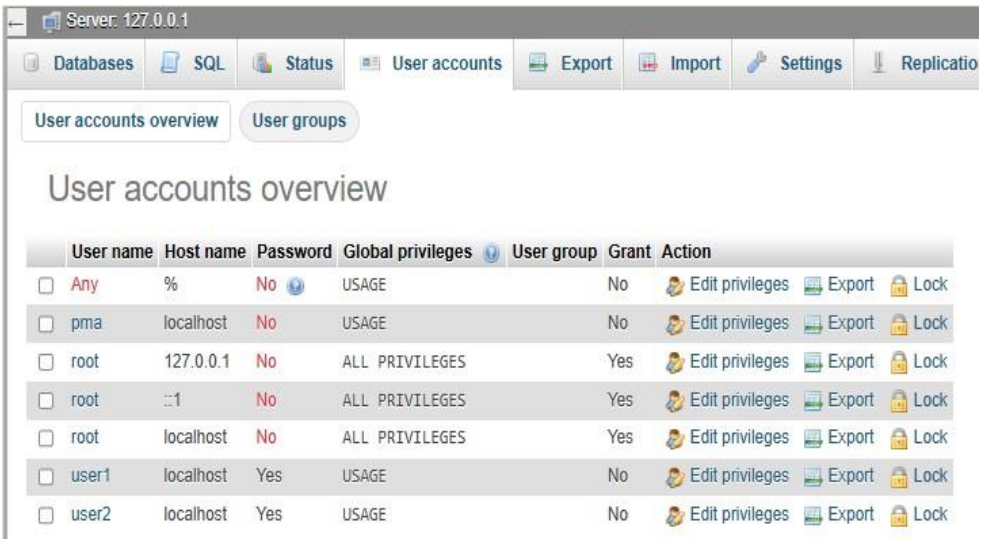
- SELECT * FROM `courses` ORDER BY course_duration DESC LIMIT 2;

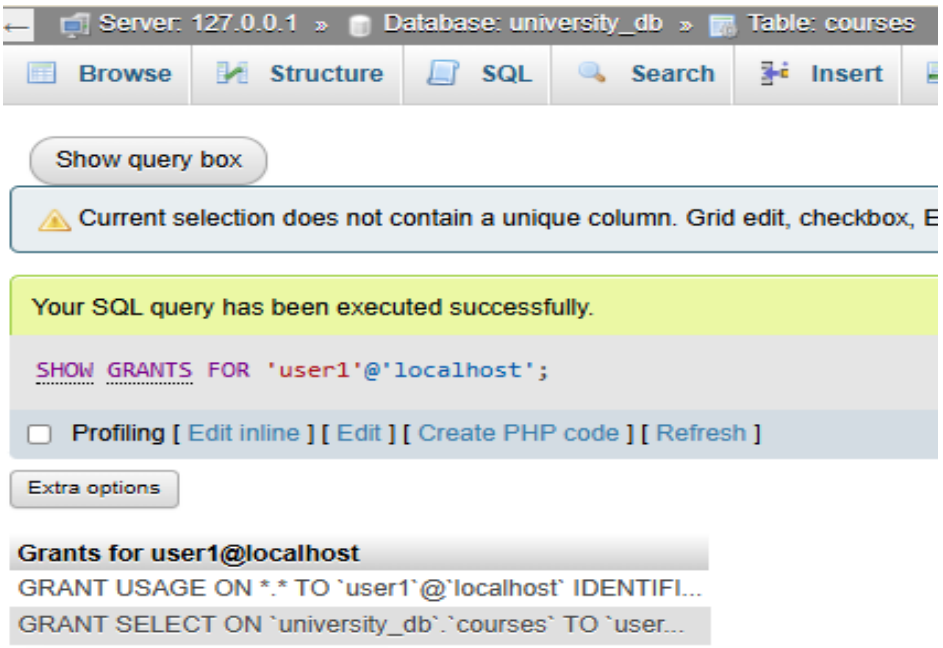


9. Data Control Language (DCL)

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

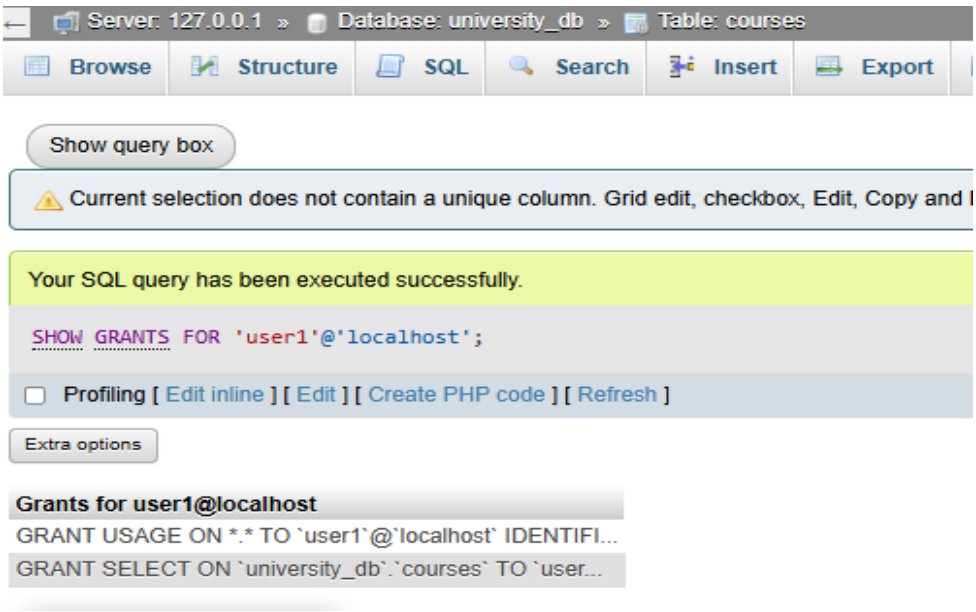
- CREATE USER 'user1'@'localhost' IDENTIFIED BY 'user1pass';
  CREATE USER 'user2'@'localhost' IDENTIFIED BY 'user2pass';
- GRANT SELECT ON university_db.courses TO 'user1'@'localhost';

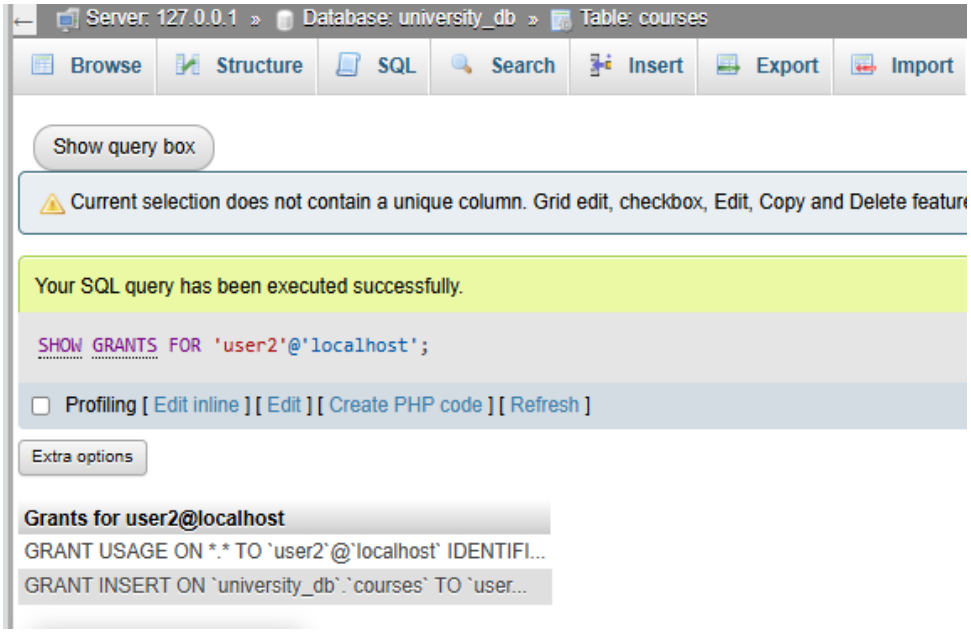




Lab 2 : Revoke the INSERT permission from user1 and give it to user2.

- REVOKE INSERT ON university_db.courses FROM 'user1'@'localhost';
- GRANT INSERT ON university_db.courses TO 'user2'@'localhost';

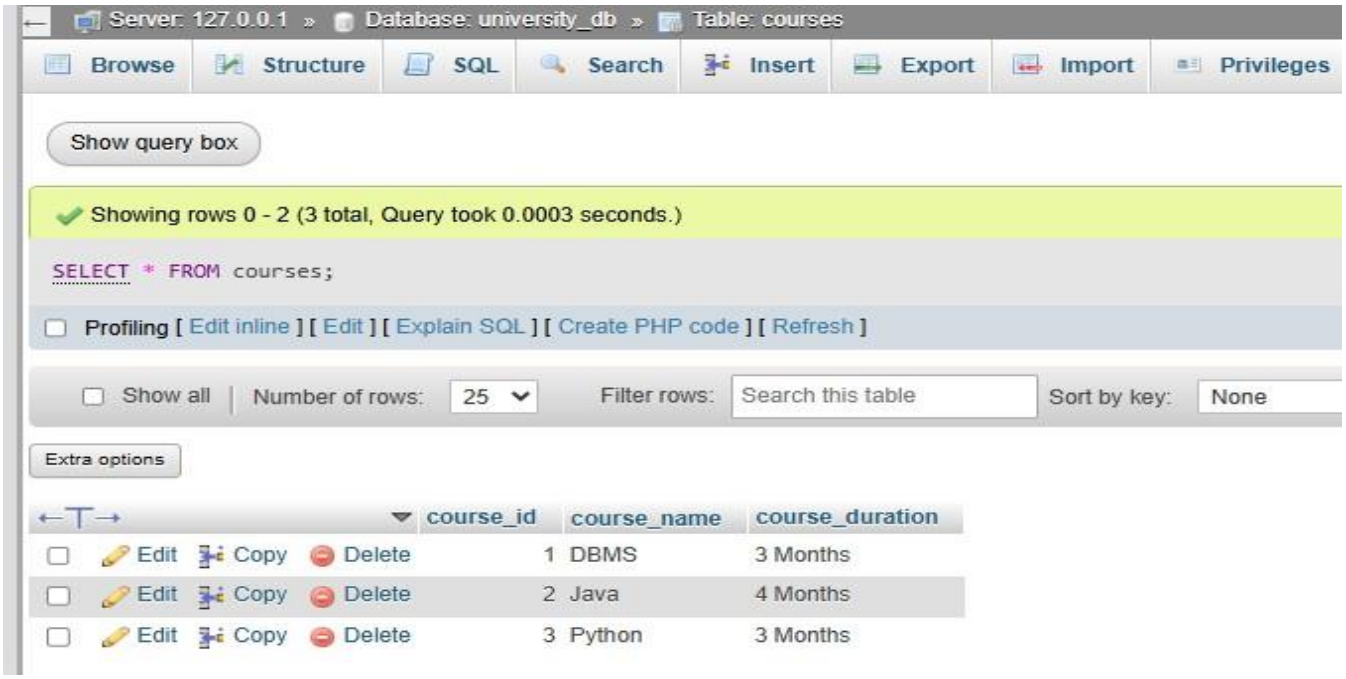




10. Transaction Control Language (TCL)

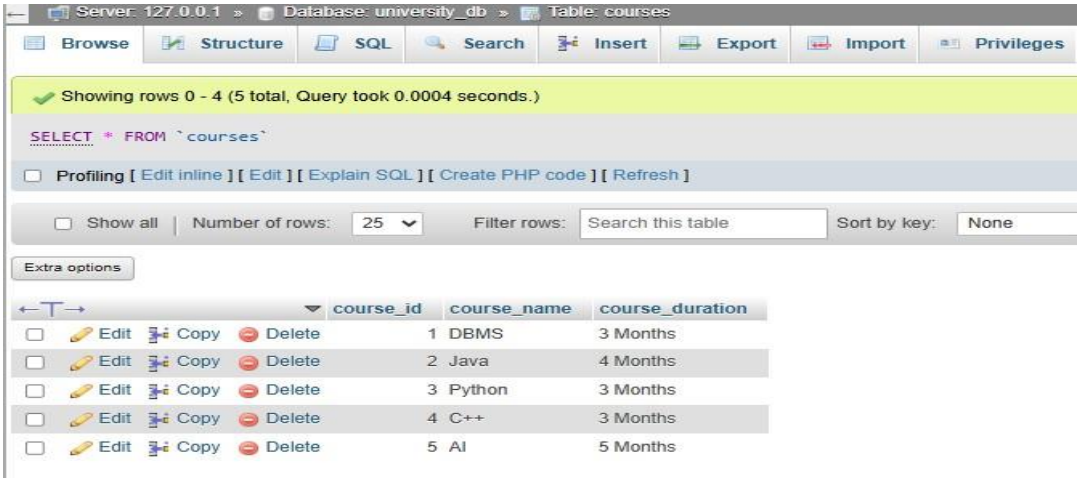
Lab 1 : Insert a few rows into the courses table and use COMMIT to save the changes.

- INSERT INTO courses (course_name, course_duration) VALUES ('DBMS', '3 Months');
  INSERT INTO courses (course_name, course_duration) VALUES ('Java', '4 Months');
  INSERT INTO courses (course_name, course_duration) VALUES ('Python', '3 Months');
  COMMIT;



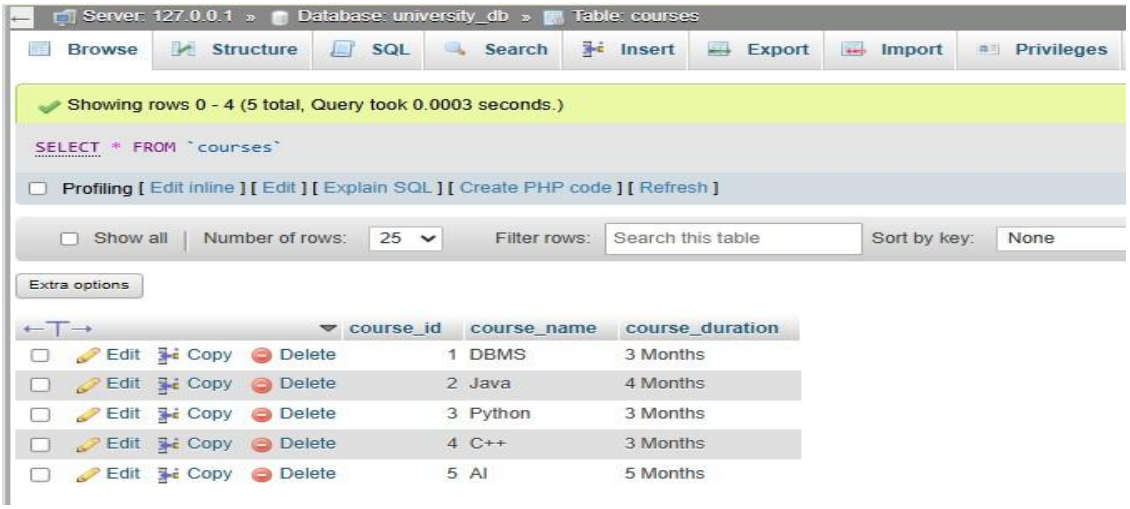
Lab 2 : Insert additional rows, then use ROLLBACK to undo the last insert operation.

- SET autocommit = 0;
  INSERT INTO courses (course_id, course_name, course_duration) VALUES (6, 'ML', '4 Months');
  INSERT INTO courses (course_id, course_name, course_duration) VALUES (7, 'Data Science', '6 Months');
  ROLLBACK;
  SET autocommit = 1;



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

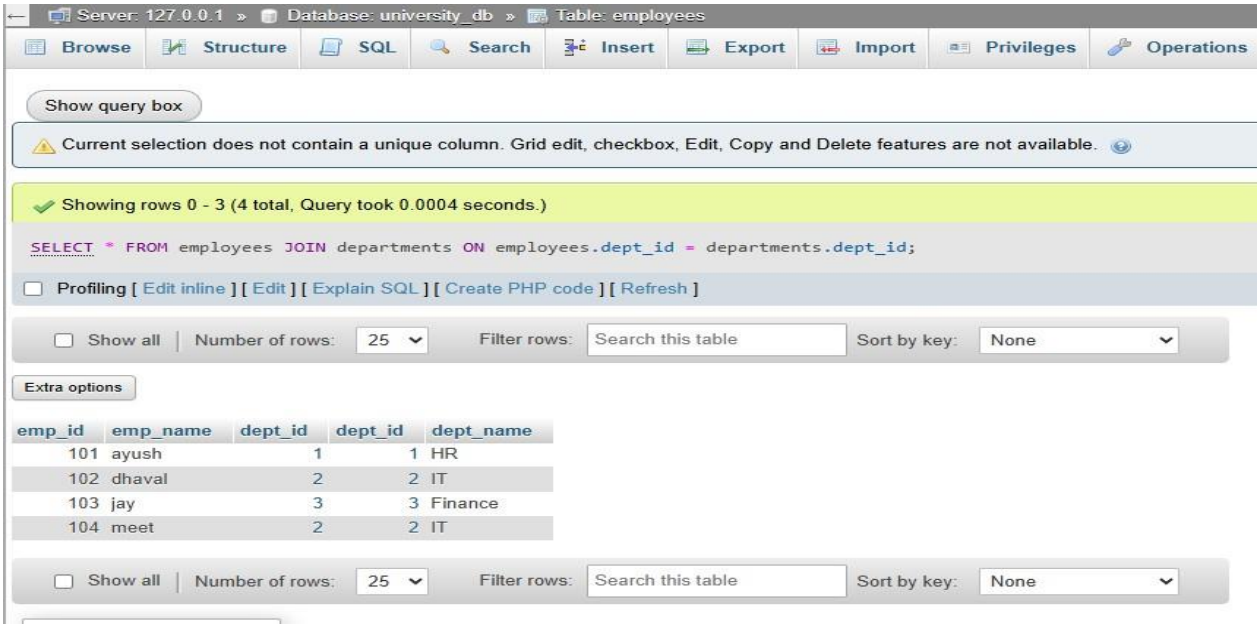
- START TRANSACTION;
  SAVEPOINT before_update;
  UPDATE courses SET course_duration = '7 Months' WHERE course_id=5;
  ROLLBACK TO before_update;
  COMMIT;



11. SQL Joins

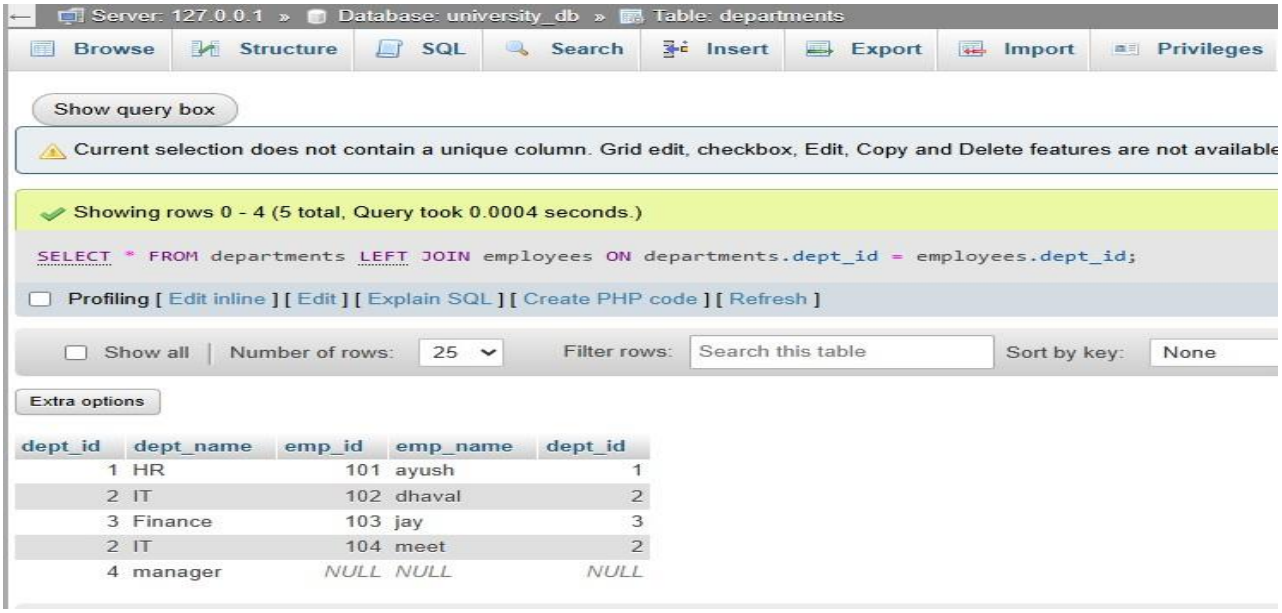
Lab 1 : Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.

- SELECT * FROM employees JOIN departments ON employees.dept_id = departments.dept_id;



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

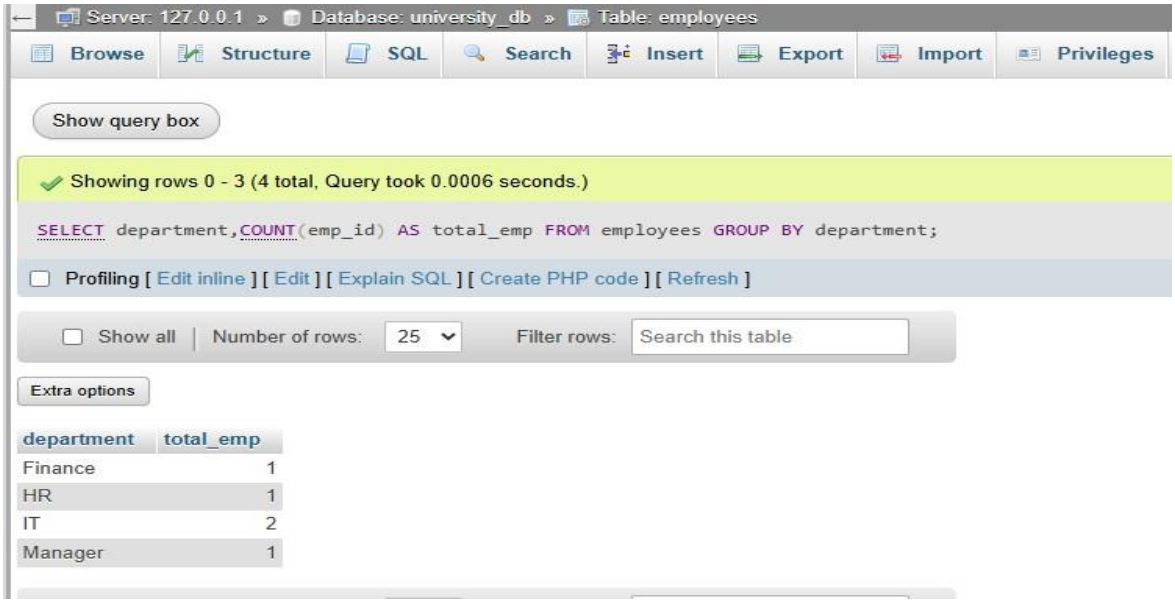
- SELECT * FROM departments LEFT JOIN employees ON departments.dept_id = employees.dept_id;



12. SQL Group By

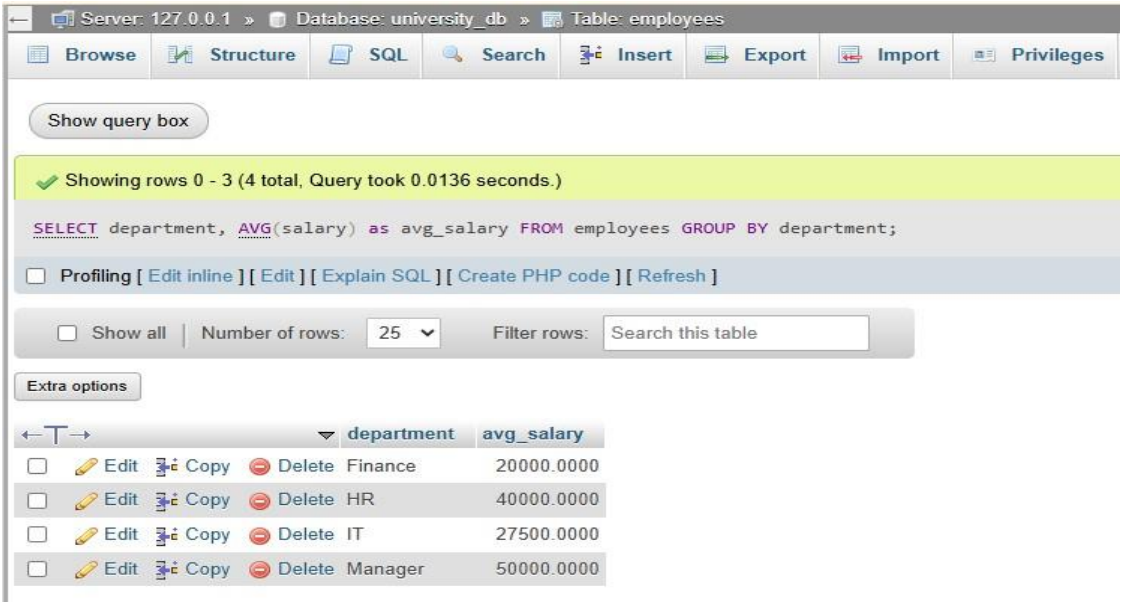
Lab 1 : Group employees by department and count the number of employees in each department using GROUP BY.

- SELECT department,COUNT(emp_id) AS total_emp FROM employees GROUP BY department



Lab 2 : Use the AVG aggregate function to find the average salary of employees in each department.

- SELECT department, AVG(salary) as avg_salary FROM employees GROUP BY department



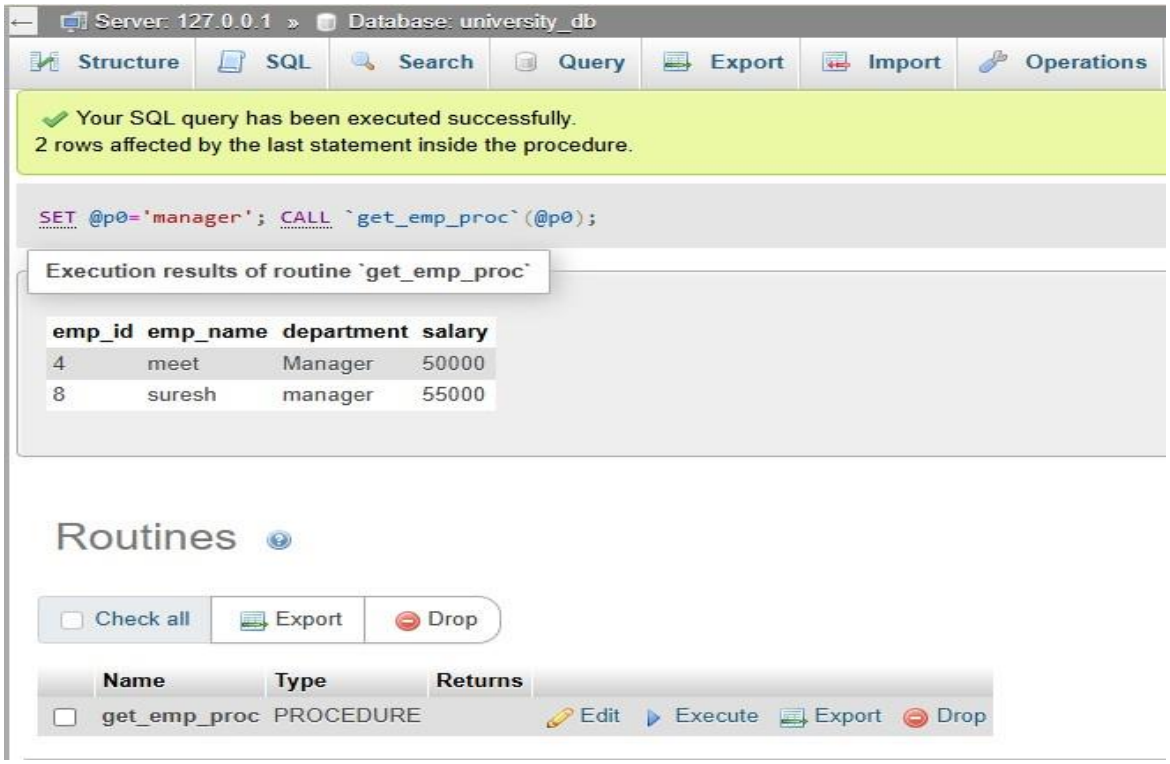
13. SQL Stored Procedure

Lab 1 : Write a stored procedure to retrieve all employees from the employees table based on department.

- DELIMITER //

  CREATE PROCEDURE get_emp_proc (IN dept_name varchar(20))
  BEGIN
  SELECT * FROM employees WHERE department = dept_name;
  END //

  DELIMITER ;

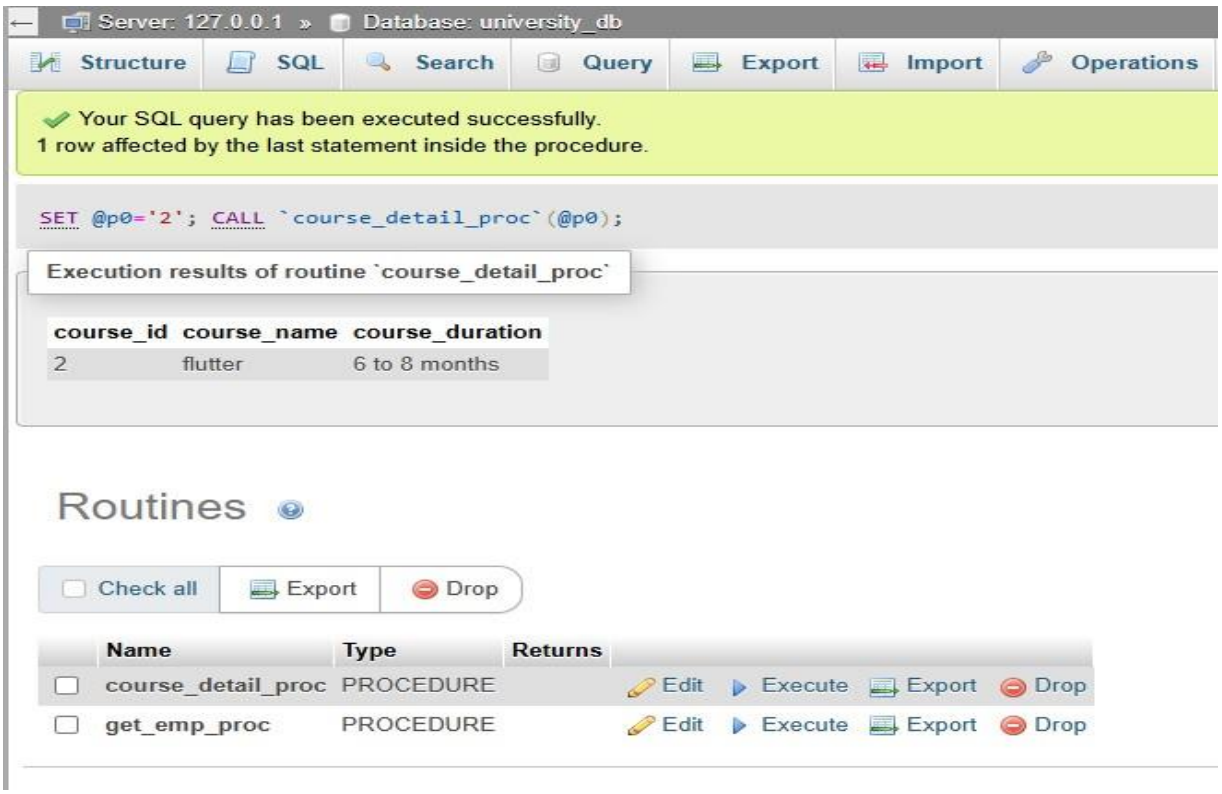


Lab 2 : Write a stored procedure that accepts course_id as input and returns the course details.

- DELIMITER //

  CREATE PROCEDURE course_detail_proc (IN c_id int)
  BEGIN
  SELECT * FROM courses WHERE course_id = c_id;
  END //

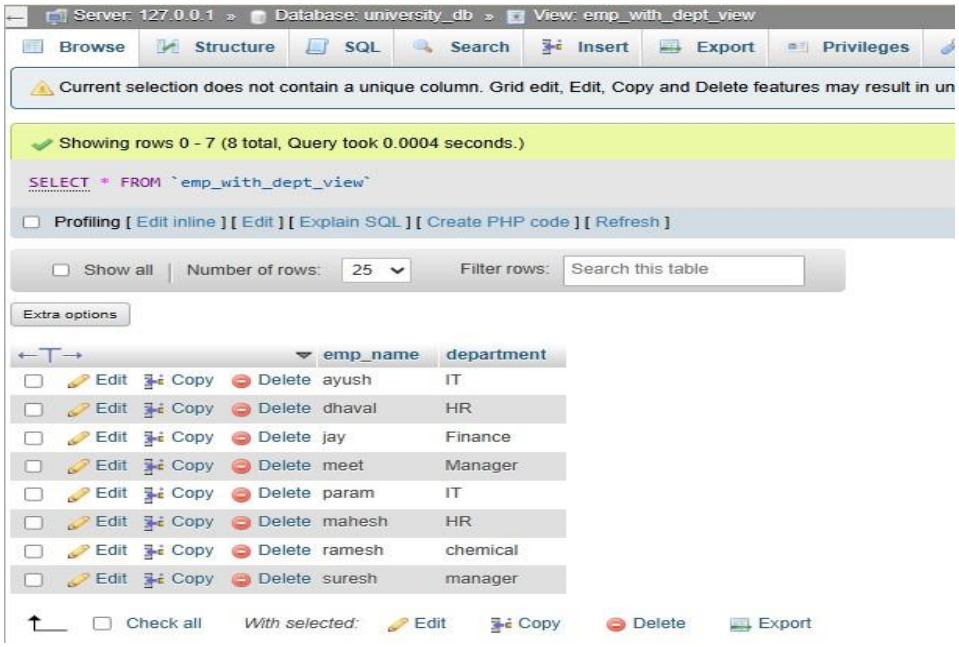
  DELIMITER ;



14. SQL View

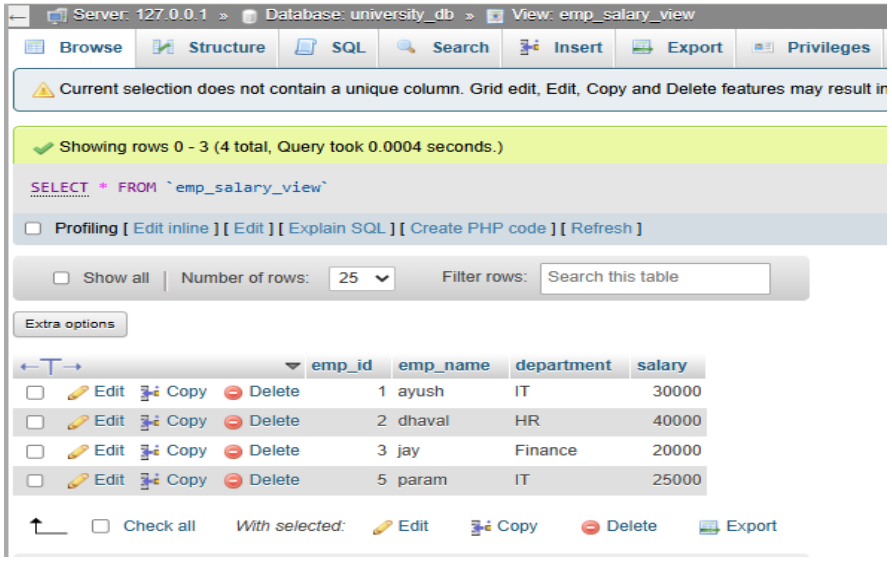
Lab 1 : Create a view to show all employees along with their department names.

- CREATE VIEW emp_with_dept_view AS SELECT emp_name,department FROM employees



Lab 2 : Modify the view to exclude employees whose salaries are below $50,000.

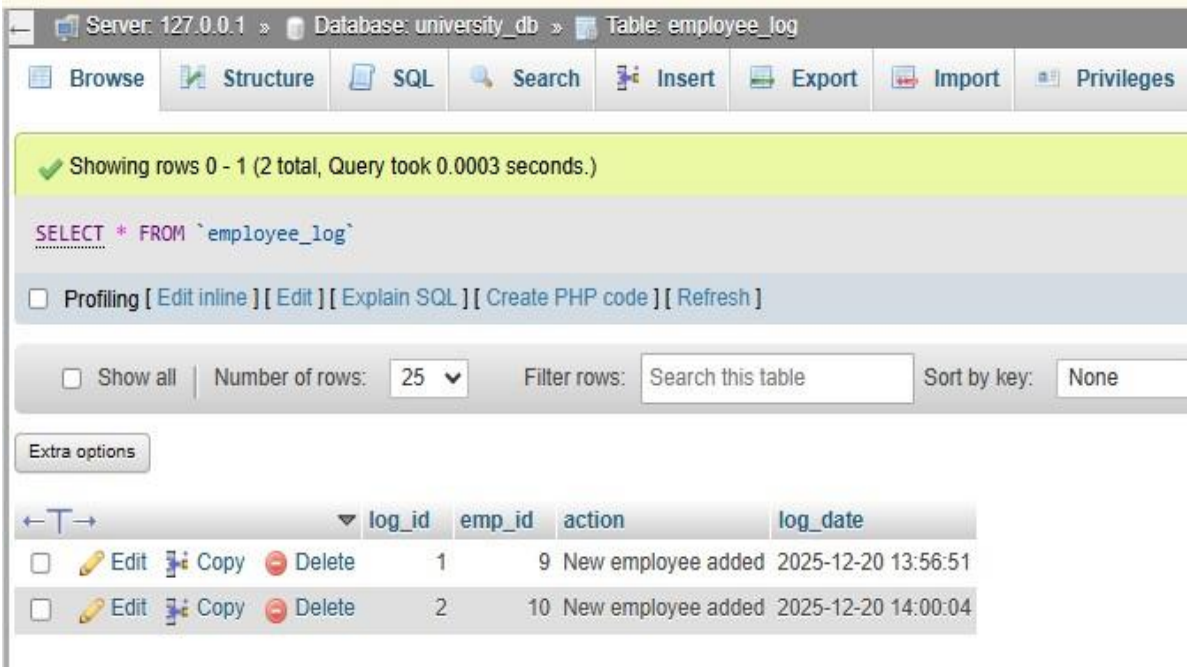
- CREATE VIEW emp_salary_view AS SELECT emp_id,emp_name,department,salary FROM employees WHERE salary<=50000



15. SQL Triggers

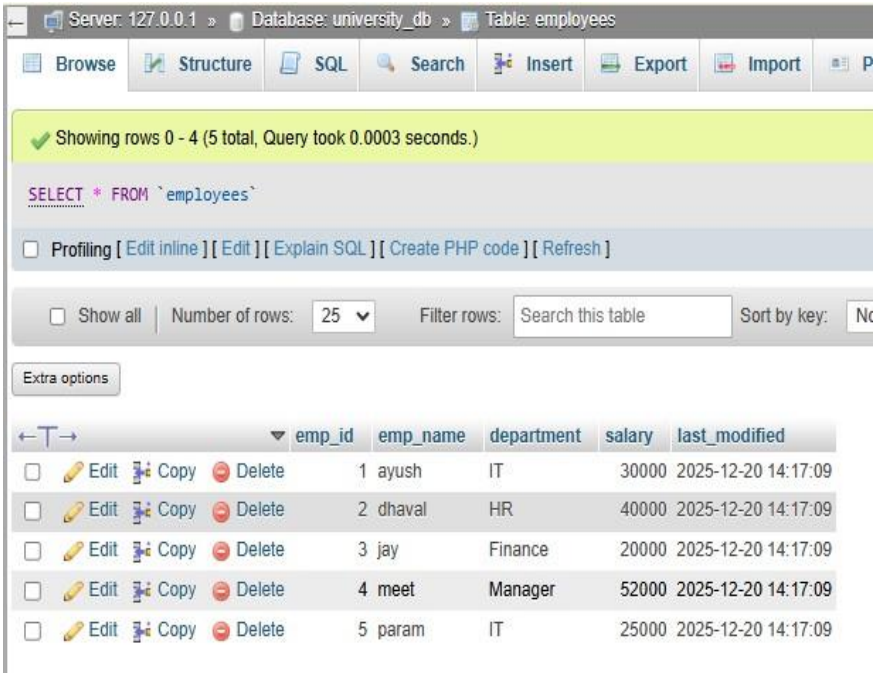
Lab 1 : Create a trigger to automatically log changes to the employees table when a new employee is added.

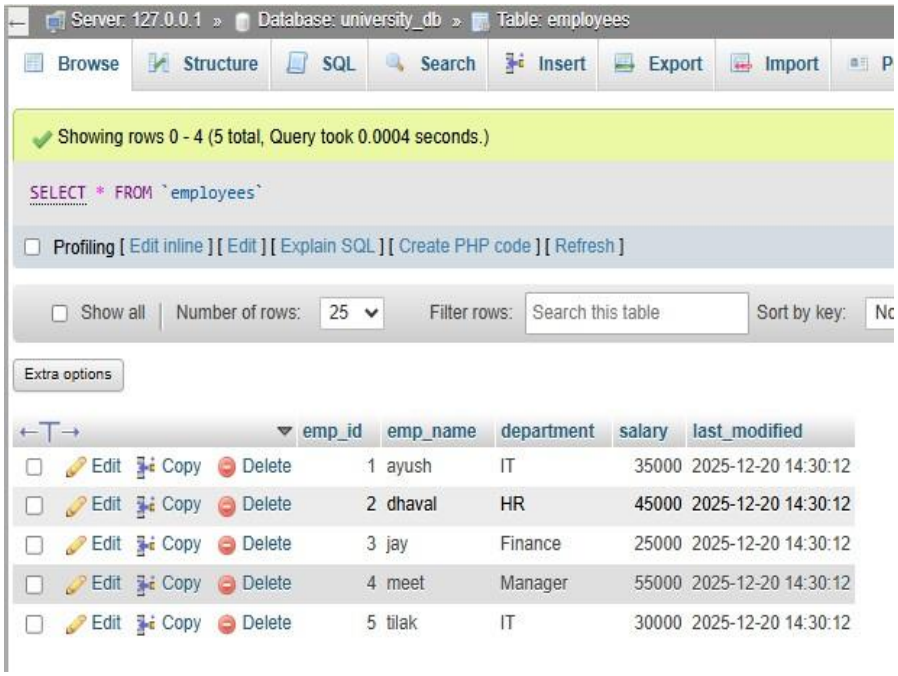
- DELIMITER $$
  CREATE TRIGGER after_emp_insert AFTER INSERT ON employees FOR EACH ROW
  BEGIN
  INSERT INTO employee_log (emp_id, action) VALUES (NEW.emp_id, 'New employee added');
  END $$
  DELIMITER ;



Lab 2 : Create a trigger to update the last_modified timestamp whenever an employee record is updated.

- CREATE TRIGGER before_emp_update BEFORE UPDATE ON employees FOR EACH ROW
  BEGIN
  SET NEW.last_modified = NOW();
  END;

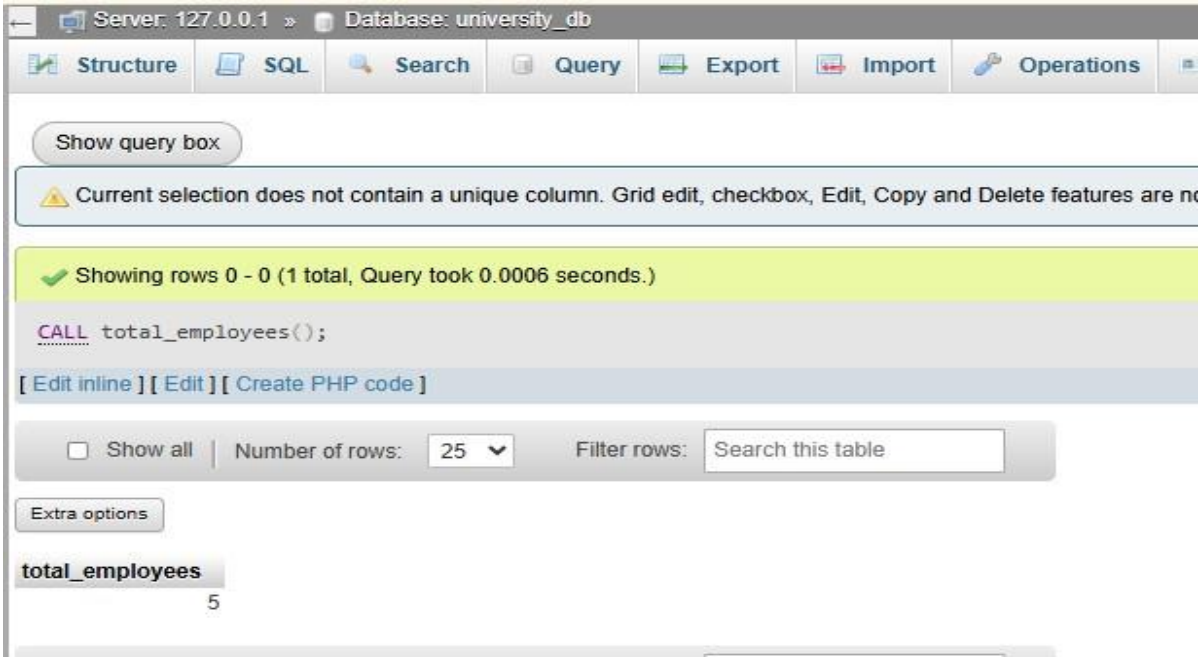




16. Introduction to PL/SQL

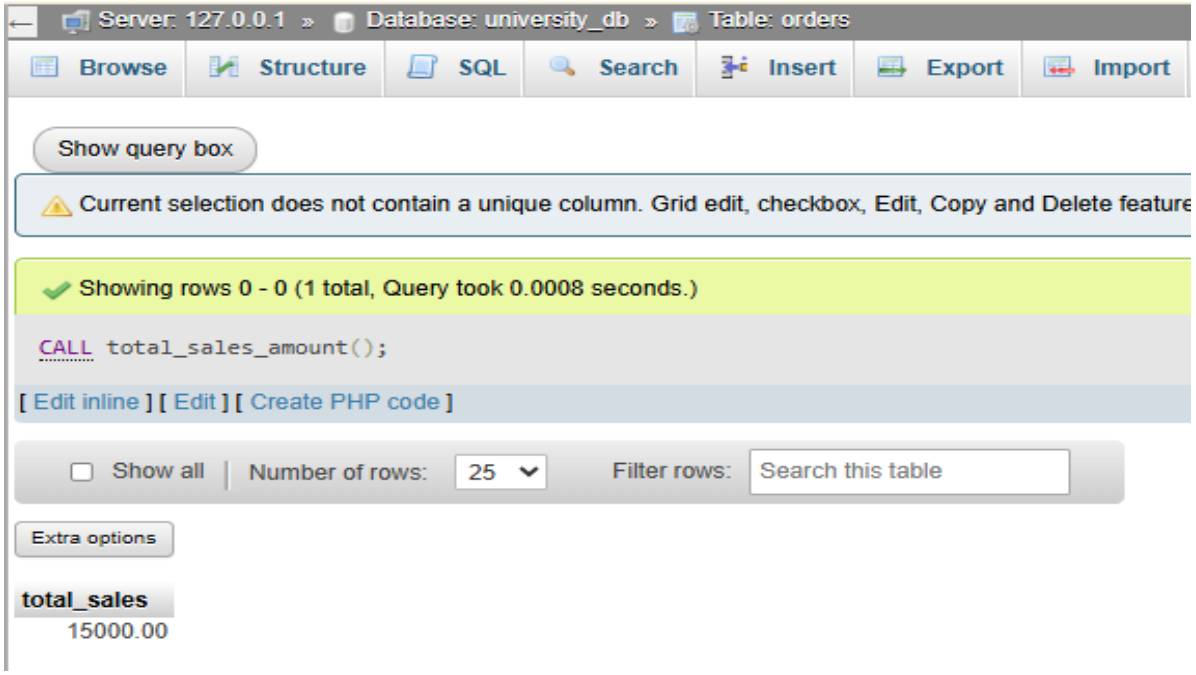
Lab 1 : Write a PL/SQL block to print the total number of employees from the employees table.

- DELIMITER $$
  CREATE PROCEDURE total_employees()
  BEGIN
   SELECT COUNT(*) AS total_employees
   FROM employees;
  END $$
  DELIMITER ;
  CALL total_employees();



Lab 2 : Create a PL/SQL block that calculates the total sales from an orders table.

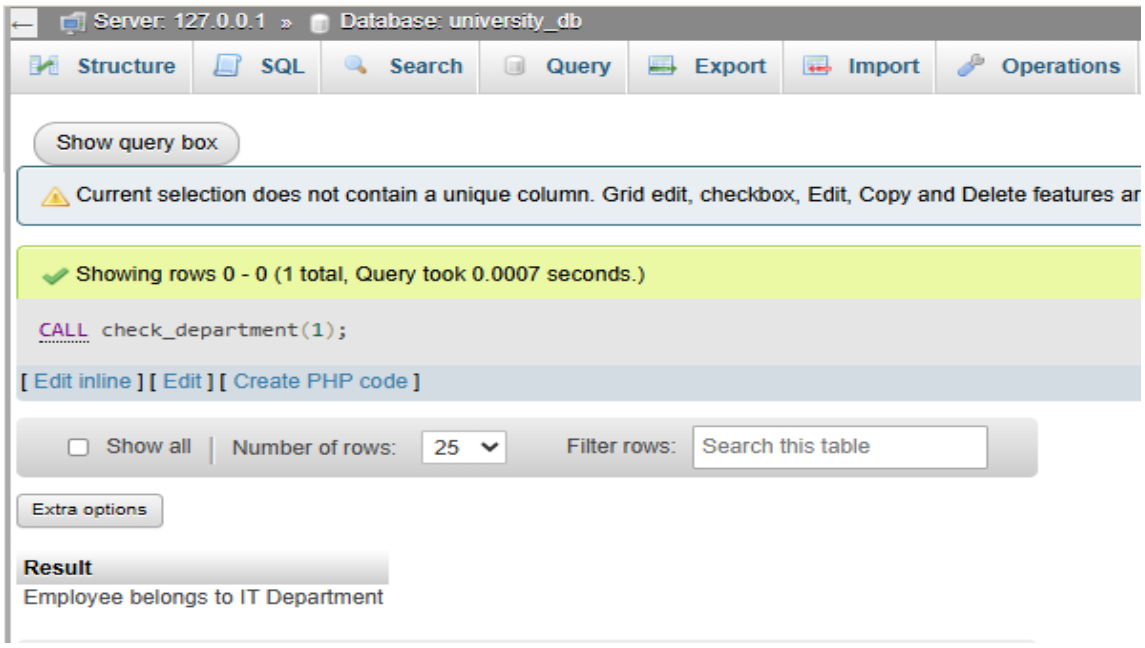
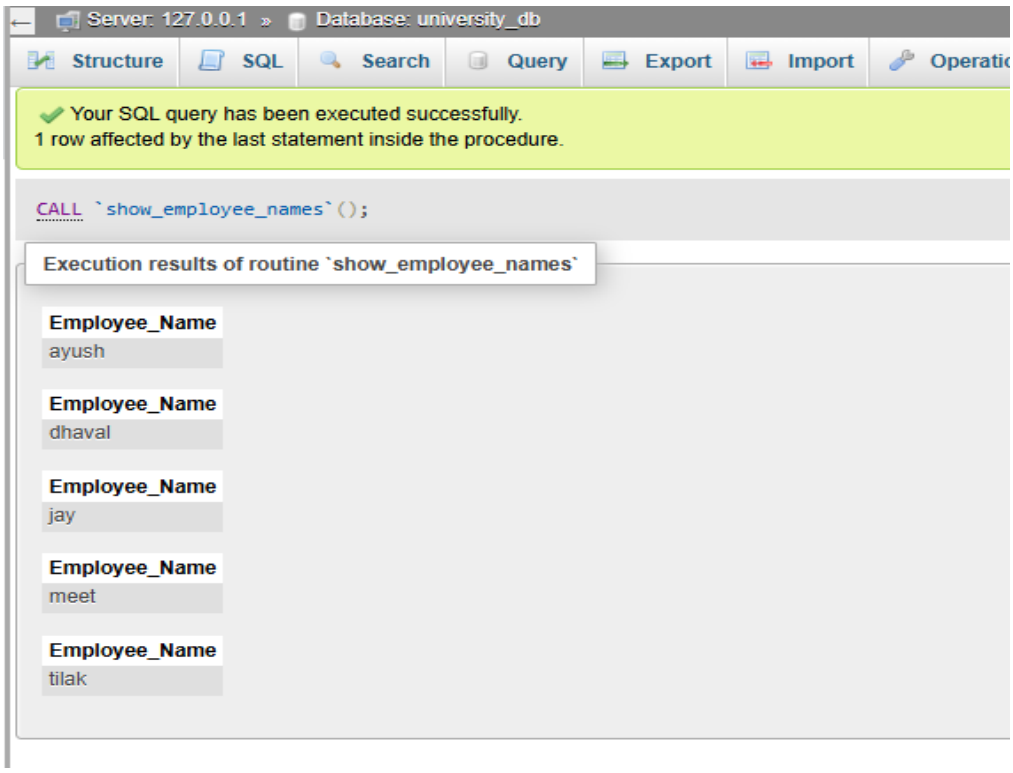
- DELIMITER $$
  CREATE PROCEDURE total_sales_amount()
  BEGIN
  DECLARE total_sales DECIMAL(10,2);
  SELECT SUM(order_amount) INTO total_sales
  FROM orders;
  SELECT total_sales AS total_sales;
  END $$
  DELIMITER ;
  CALL total_sales_amount();



17. PL/SQL Control Structures

Lab 1 : Write a PL/SQL block using an IF-THEN condition to check the department of an employee.

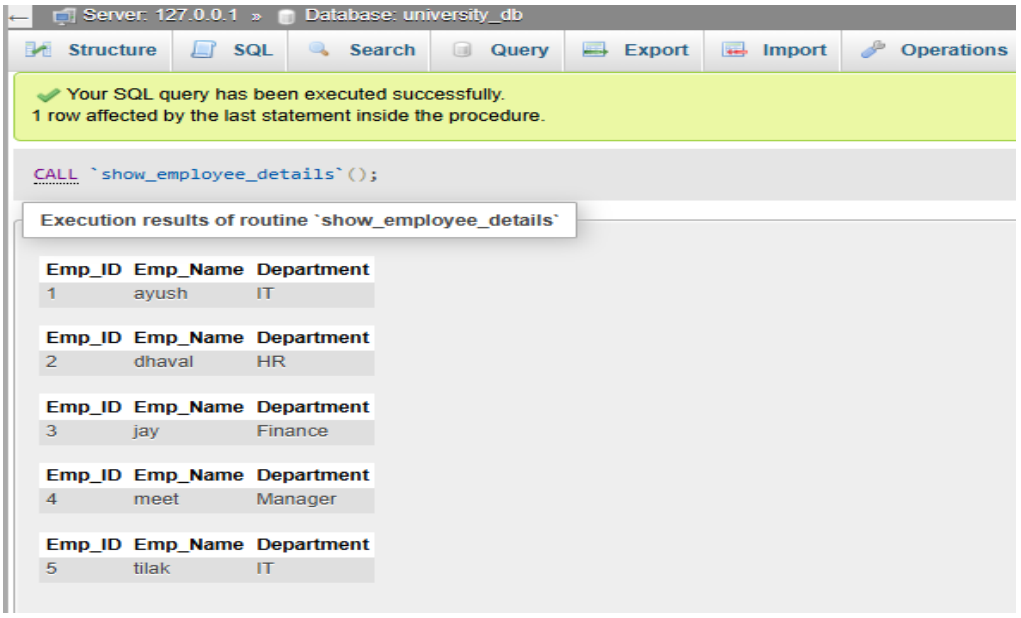
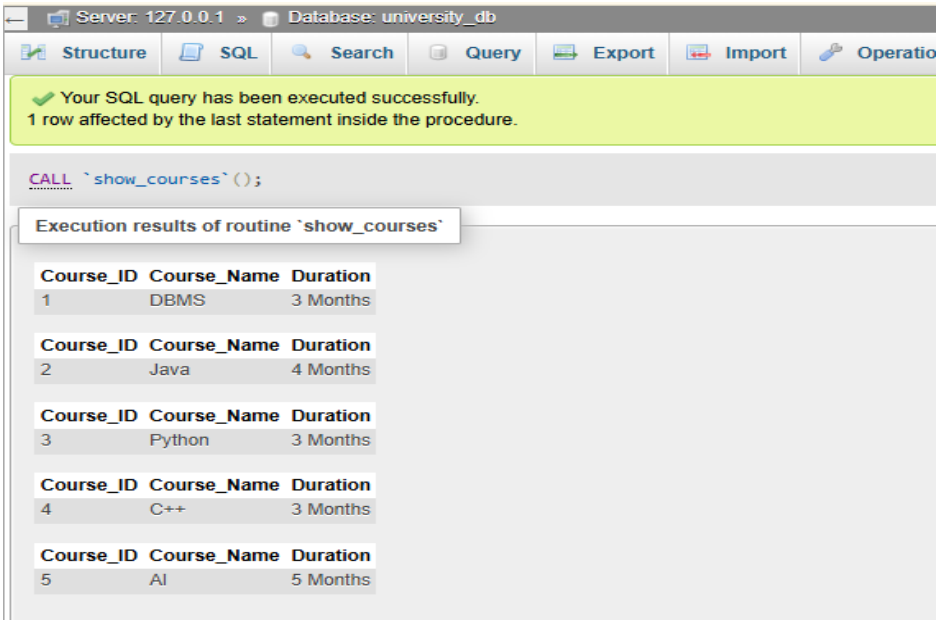
- ```
  DELIMITER $$
  CREATE PROCEDURE check_department(IN p_emp_id INT)
  BEGIN
   DECLARE v_department VARCHAR(50);
   SELECT department
  INTO v_department
   FROM employees
   WHERE emp_id = p_emp_id;
   IF v_department = 'IT' THEN
   SELECT 'Employee belongs to IT Department' AS Result;
  ELSE
  SELECT 'Employee does not belong to IT Department' AS Result;
   END IF;
  END $$
  DELIMITER ;
  CALL check_department(2);
  ```


  ```

Lab 2 : Use a FOR LOOP to iterate through employee records and display their names.

- ```
  DELIMITER $$
  CREATE PROCEDURE show_employee_names()
  BEGIN
   DECLARE done INT DEFAULT 0;
  DECLARE v_name VARCHAR(100);
   DECLARE emp_cursor CURSOR FOR
   SELECT emp_name FROM employees;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
  OPEN emp_cursor;
  read_loop: LOOP
   FETCH emp_cursor INTO v_name;
  IF done = 1 THEN
  LEAVE read_loop;
   END IF;
   SELECT v_name AS Employee_Name;
  END LOOP;
   CLOSE emp_cursor;
  END$$
  DELIMITER ;
  ```


  ```

18. SQL Cursors

Lab 1 : Write a PL/SQL block using an explicit cursor to retrieve and display employee details.

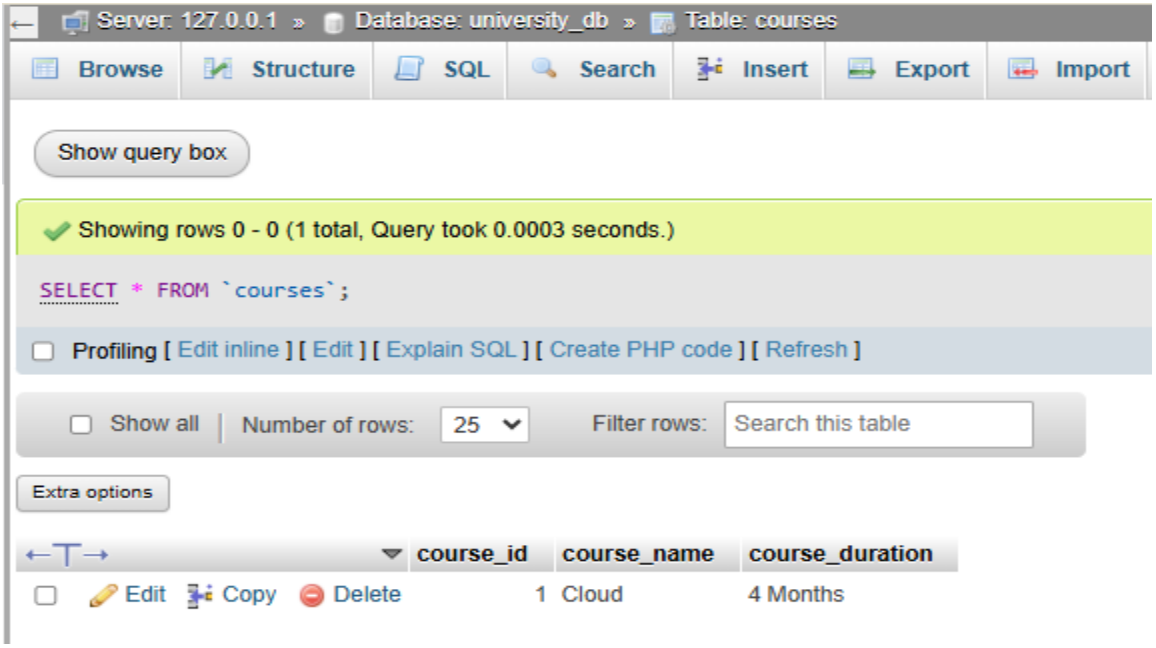
- ```
  DELIMITER $$
  CREATE PROCEDURE show_employee_details()
  BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_id INT;
    DECLARE v_name VARCHAR(100);
    DECLARE v_dept VARCHAR(50);
    DECLARE emp_cursor CURSOR FOR
      SELECT emp_id, emp_name, department FROM employees;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    OPEN emp_cursor;
    emp_loop: LOOP
      FETCH emp_cursor INTO v_id, v_name, v_dept;
      IF done = 1 THEN
        LEAVE emp_loop;
      END IF;
      SELECT v_id AS Emp_ID,
          v_name AS Emp_Name,
          v_dept AS Department;
    END LOOP;
    CLOSE emp_cursor;
  END$$
  DELIMITER ;
  ```


  ```

Lab 2 : Create a cursor to retrieve all courses and display them one by one.

- ```
  DELIMITER $$
  CREATE PROCEDURE show_courses()
  BEGIN
    DECLARE done INT DEFAULT 0;
     DECLARE v_id INT;
     DECLARE v_name VARCHAR(100);
     DECLARE v_duration VARCHAR(50);
     DECLARE course_cursor CURSOR FOR
        SELECT course_id, course_name, course_duration FROM courses;
     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
     OPEN course_cursor;
     course_loop: LOOP
        FETCH course_cursor INTO v_id, v_name, v_duration;
        IF done = 1 THEN
           LEAVE course_loop;
        END IF;
        SELECT v_id AS Course_ID, v_name AS Course_Name,  v_duration
  AS Duration;
     END LOOP;
     CLOSE course_cursor;
  END$$
  DELIMITER ;
  ```


  ```

19. Rollback and Commit Savepoint

Lab 1 : Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.

- START TRANSACTION;
  INSERT INTO courses (course_name, course_duration)
  VALUES ('Cloud', '4 Months');
  SAVEPOINT sp1;
  INSERT INTO courses (course_name, course_duration)
  VALUES ('DevOps', '5 Months');
  ROLLBACK TO sp1;
  COMMIT;



Lab 2 : Commit part of a transaction after using a savepoint and then rollback the remaining changes.

- START TRANSACTION;
  INSERT INTO courses (course_name, course_duration)
  VALUES ('Java Advanced', '5 Months');
  SAVEPOINT sp2;
  INSERT INTO courses (course_name, course_duration)
  VALUES ('Spring Boot', '4 Months');
  COMMIT;
  START TRANSACTION;
  INSERT INTO courses (course_name, course_duration)
  VALUES ('Microservices', '6 Months');
  ROLLBACK;