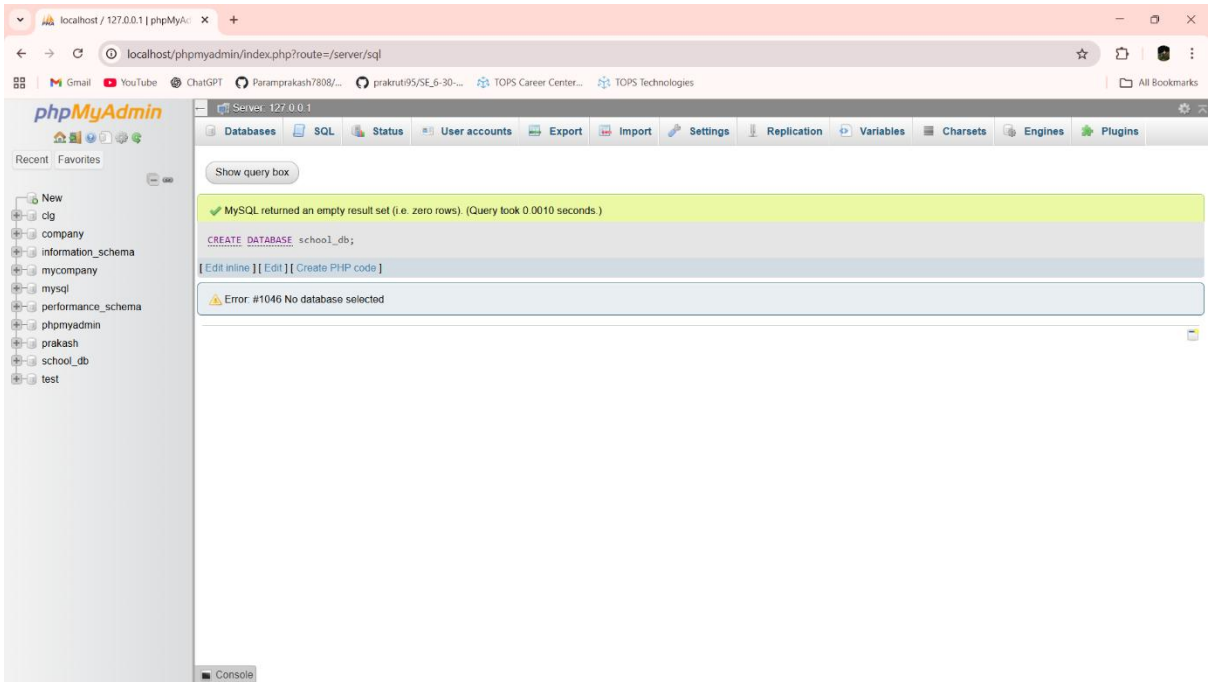
Name: Paramprakash Makwana

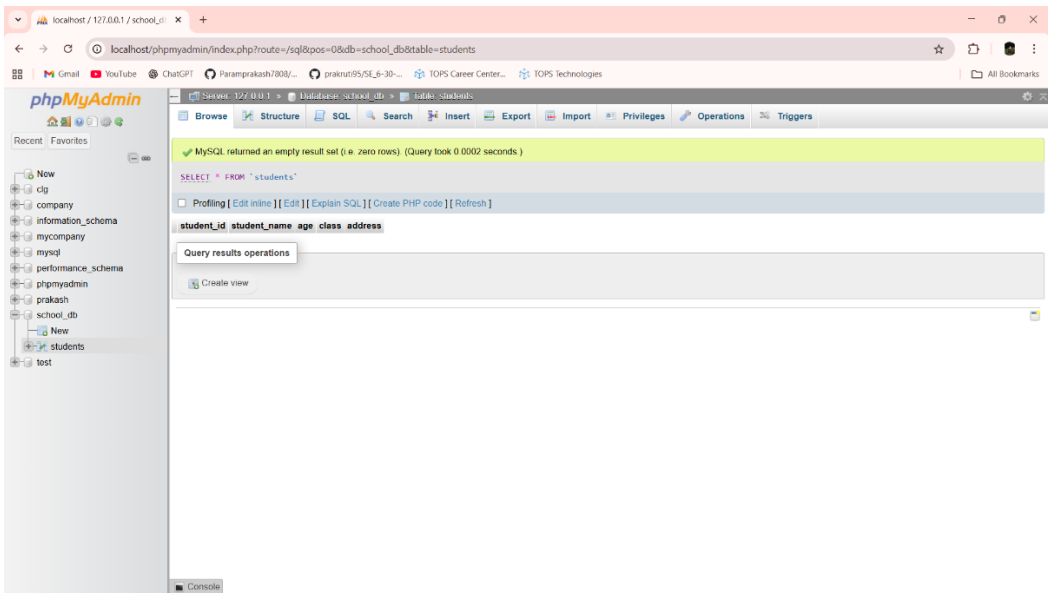
Module 4 – Introduction to DBMS(Lab)

Lab 1.1: Create a new database named school_db and a table called students with the following columns: student_id, student_name, age, class, and address.

Ans:- CREATE DATABASE school_db



CREATE TABLE students (student_id int PRIMARY KEY AUTO_INCREMENT,student_name varchar(30),age int,class varchar(30),address varchar(50))



Lab 1.2: Insert five records into the students table and retrieve all records using the SELECT statement.

Ans:- INSERT INTO students (student_id,student_name,age,class,address) VALUES (101,'Prakash',21,'8TD2','Morbi Road Bedi Village Rajkot')

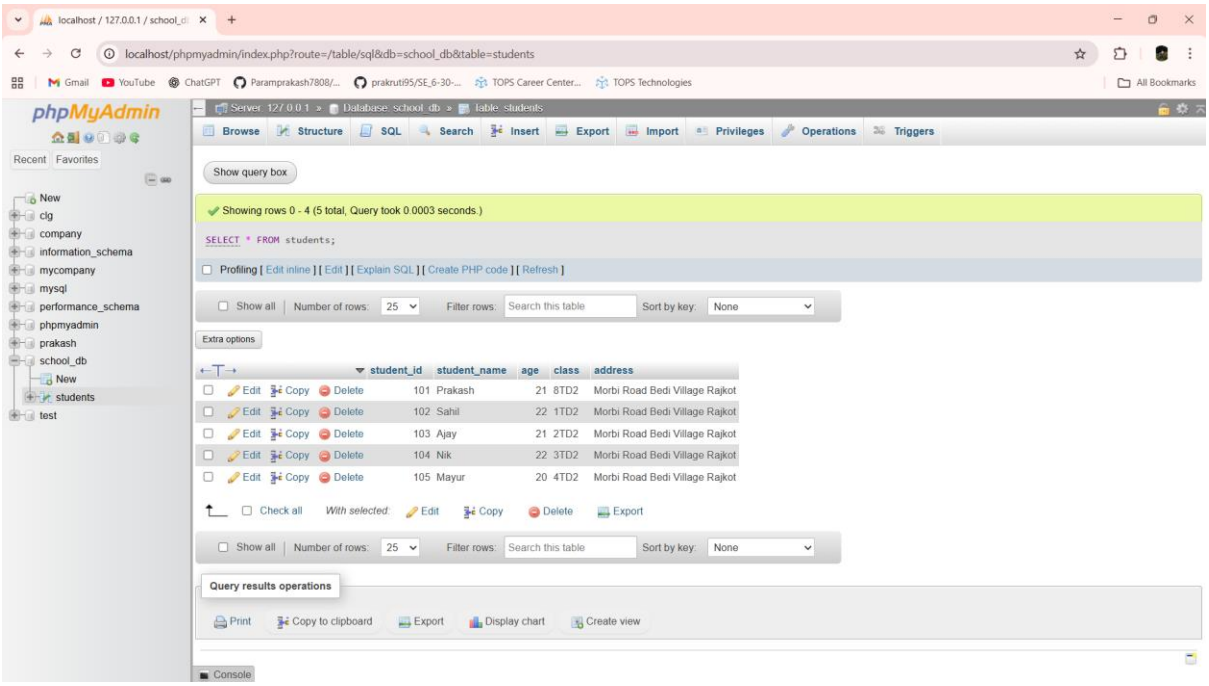
INSERT INTO students (student_id,student_name,age,class,address) VALUES (102,'Sahil',22,'1TD2','Morbi Road Bedi Village Rajkot')

INSERT INTO students (student_id,student_name,age,class,address) VALUES (103,'Ajay',21,'2TD2','Morbi Road Bedi Village Rajkot')

INSERT INTO students (student_id,student_name,age,class,address) VALUES (104,'Nik',22,'3TD2','Morbi Road Bedi Village Rajkot')

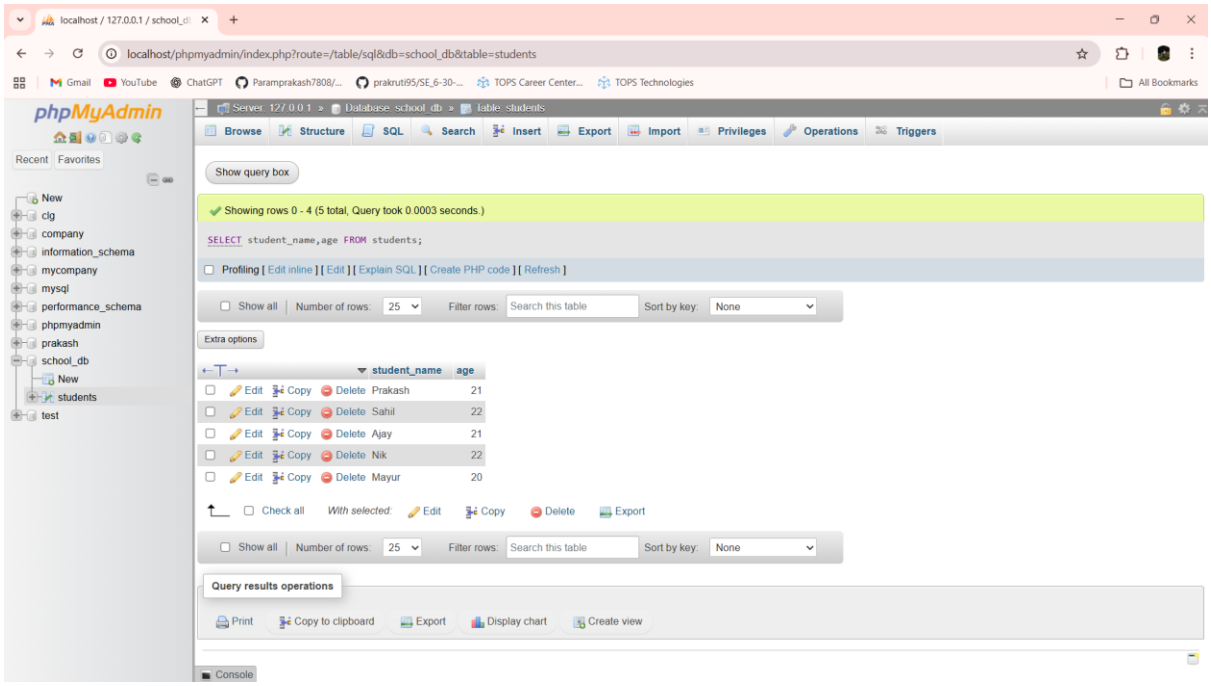
INSERT INTO students (student_id,student_name,age,class,address) VALUES (105,'Mayur',20,'4TD2','Morbi Road Bedi Village Rajkot')

SELECT * FROM students



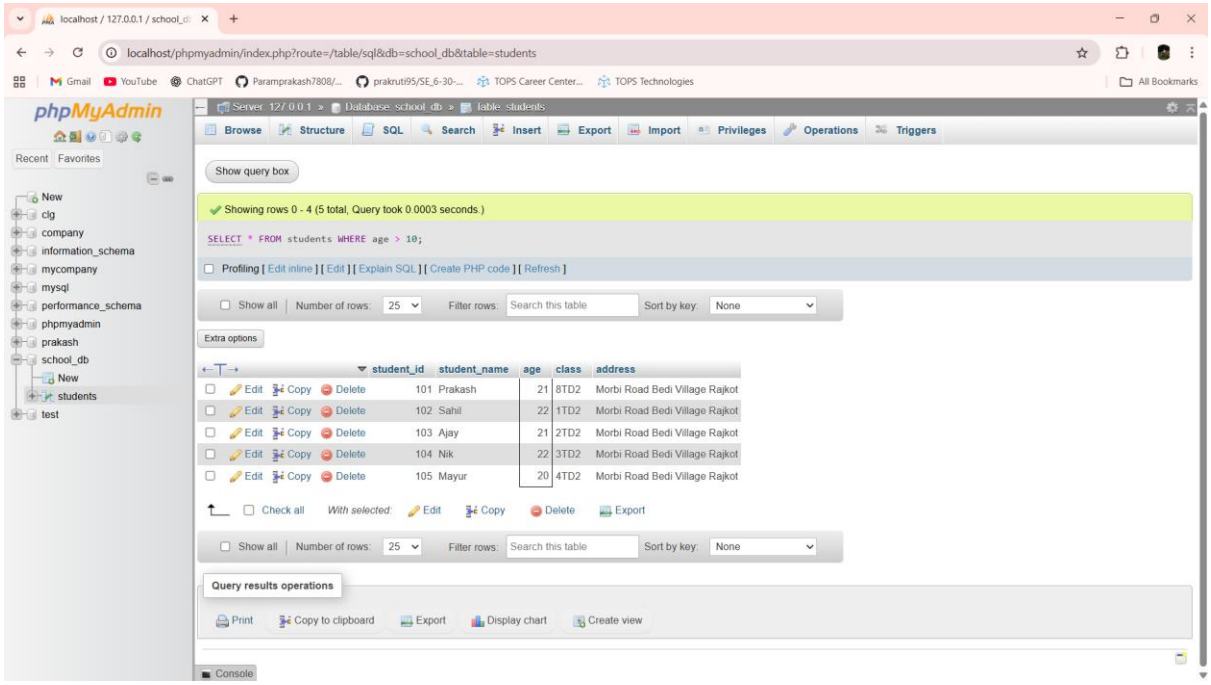
Lab 2.1: Write SQL queries to retrieve specific columns (student_name and age) from the students table.

Ans:- SELECT student_name,age FROM students



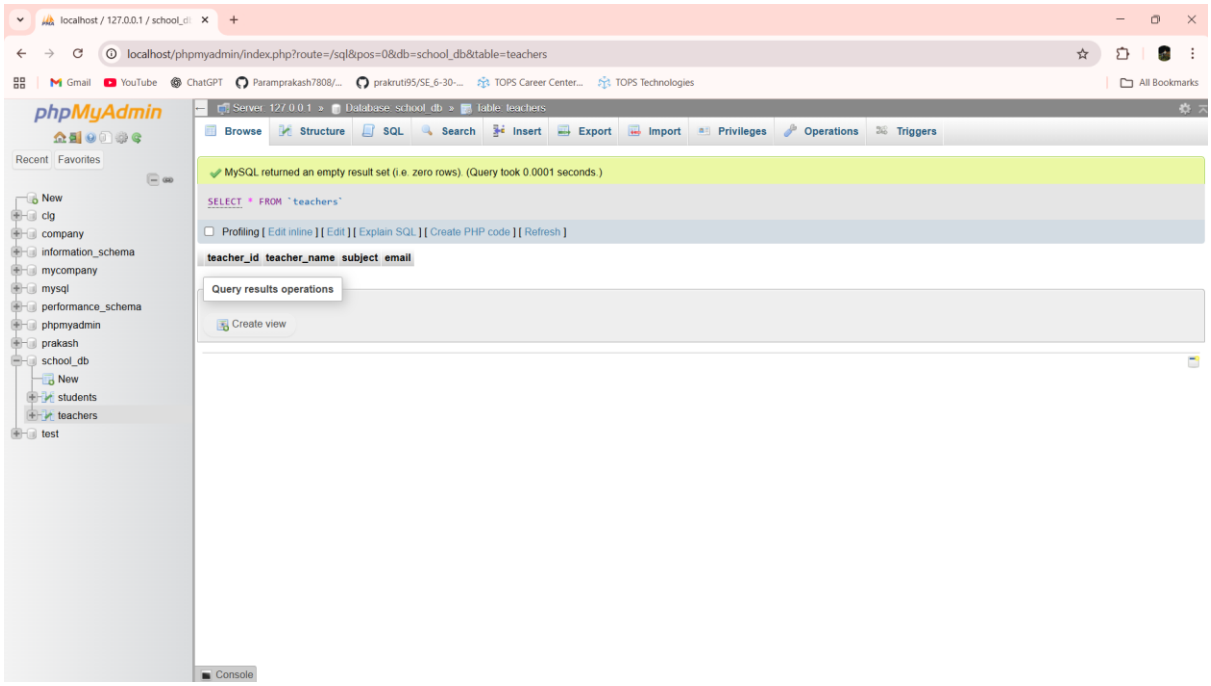
Lab 2.2: Write SQL queries to retrieve all students whose age is greater than 10.

Ans:- SELECT * FROM students WHERE age > 10



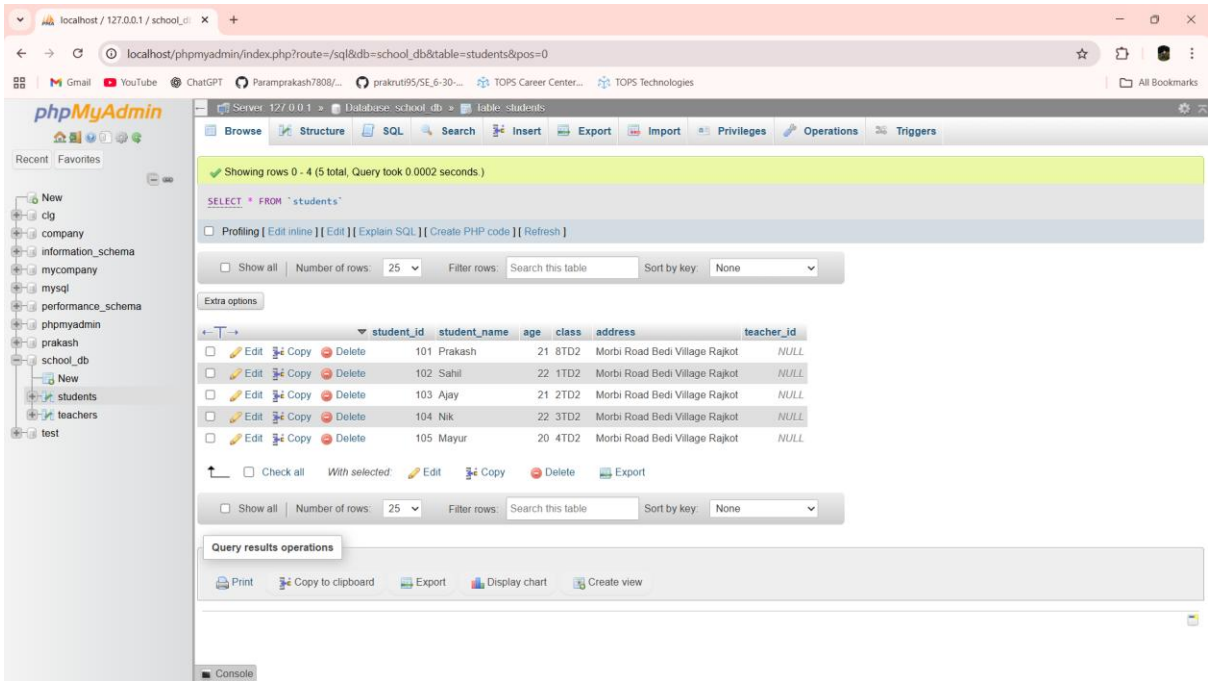
Lab 3.1: Create a table teachers with the following columns: teacher_id (Primary Key), teacher_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).

Ans:- CREATE TABLE teachers(teacher_id int Primary Key AUTO_INCREMENT,teacher_name varchar(30), subject varchar(30),email varchar(30))



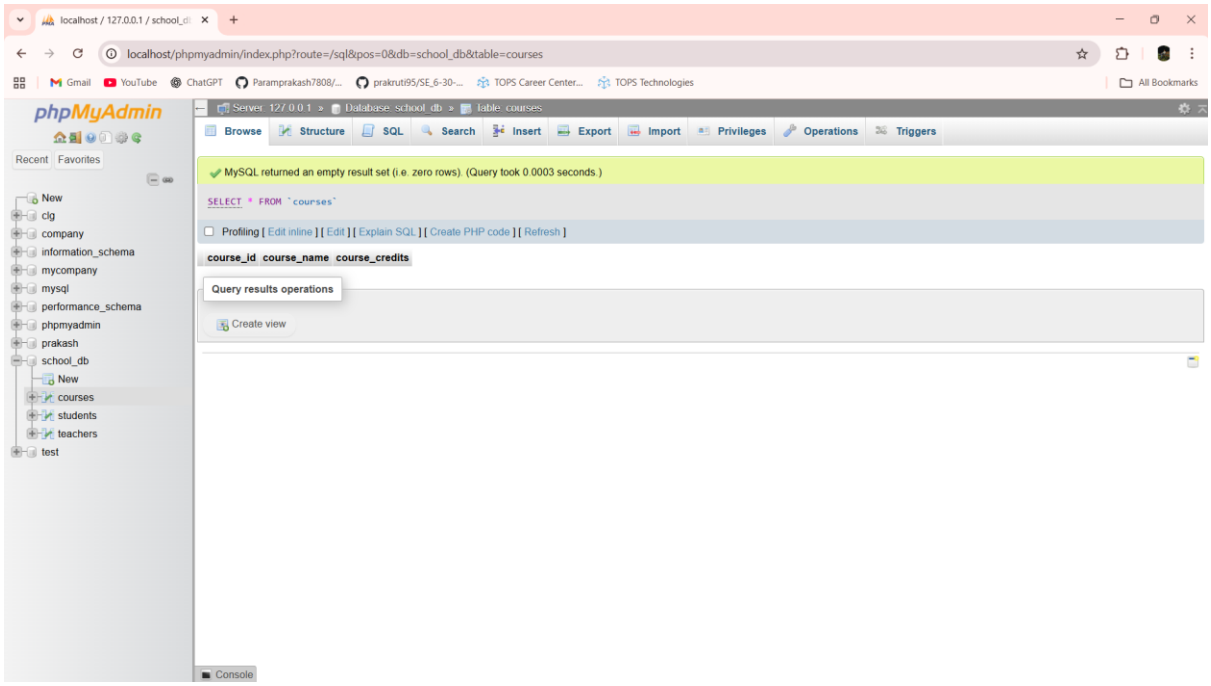
Lab 3.2: Implement a FOREIGN KEY constraint to relate the teacher_id from the teachers table with the students table.

Ans:- ALTER TABLE students ADD CONSTRAINT fk_teacher FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)



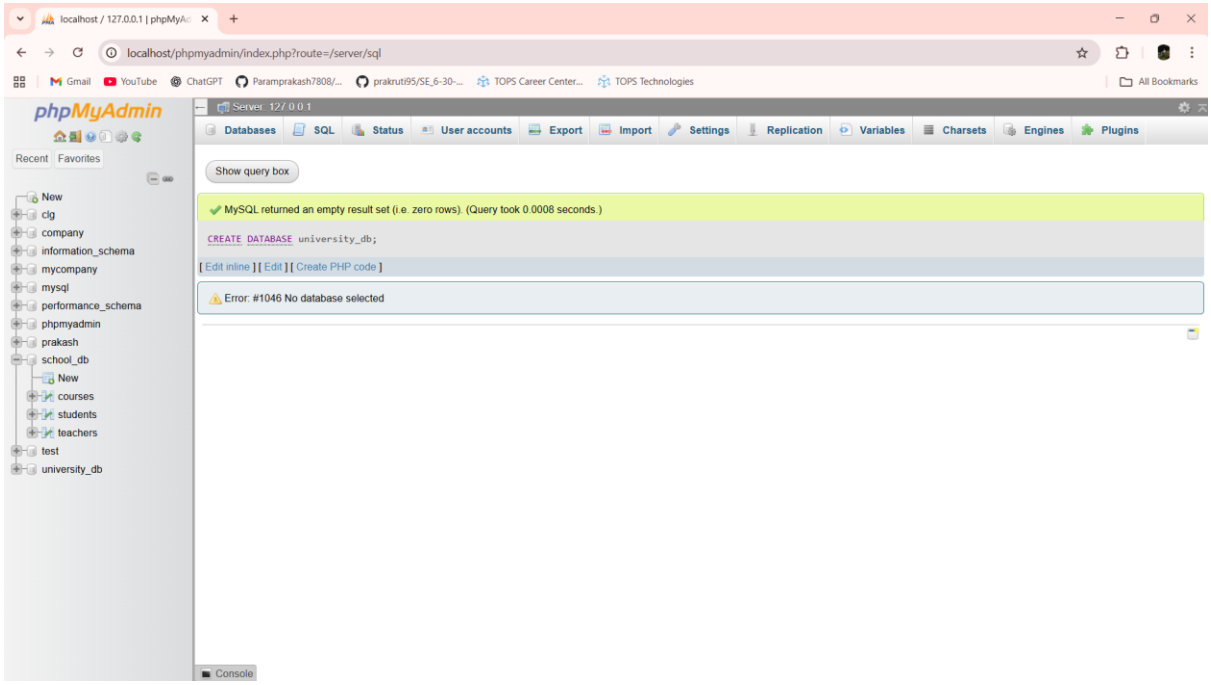
Lab 4.1: Create a table courses with columns: course_id, course_name, and course_credits. Set the course_id as the primary key.

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



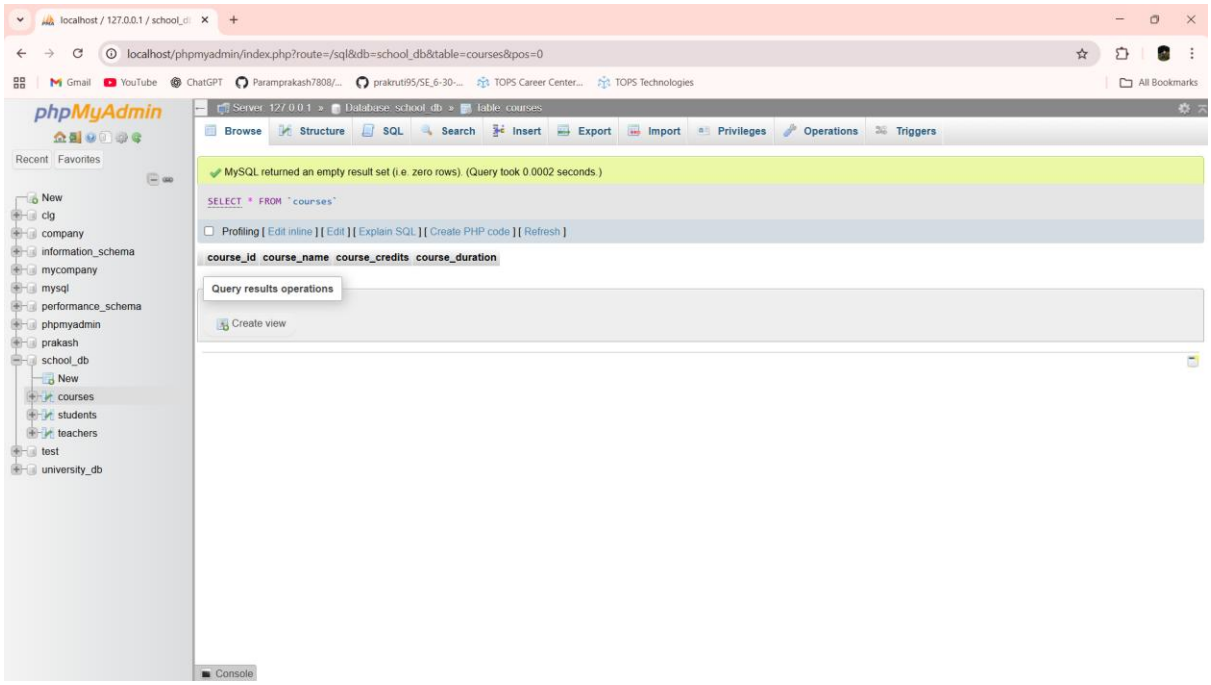
Lab 4.2: Use the CREATE command to create a database university_db.

Ans:- CREATE DATABASE university_db



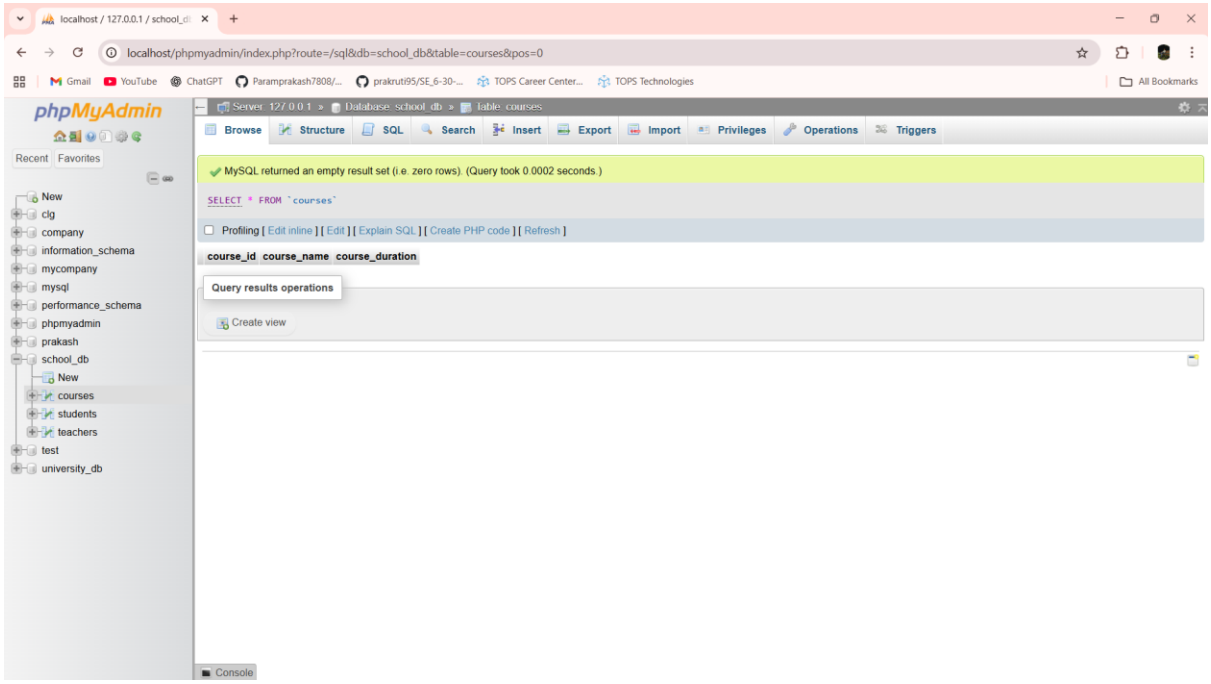
Lab 5.1: Modify the courses table by adding a column course_duration using the ALTER command.

Ans:- ALTER TABLE courses ADD course_duration varchar(30)



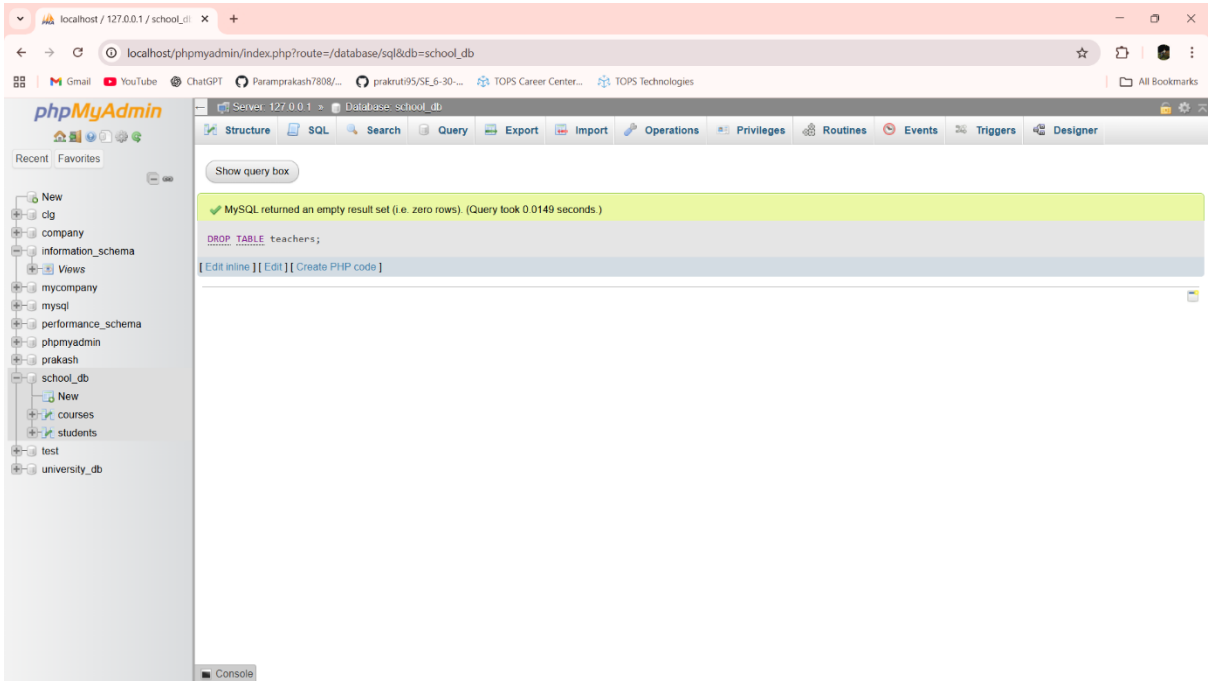
Lab 5.2: Drop the course_credits column from the courses table.

Ans:- ALTER TABLE courses DROP COLUMN course_credits



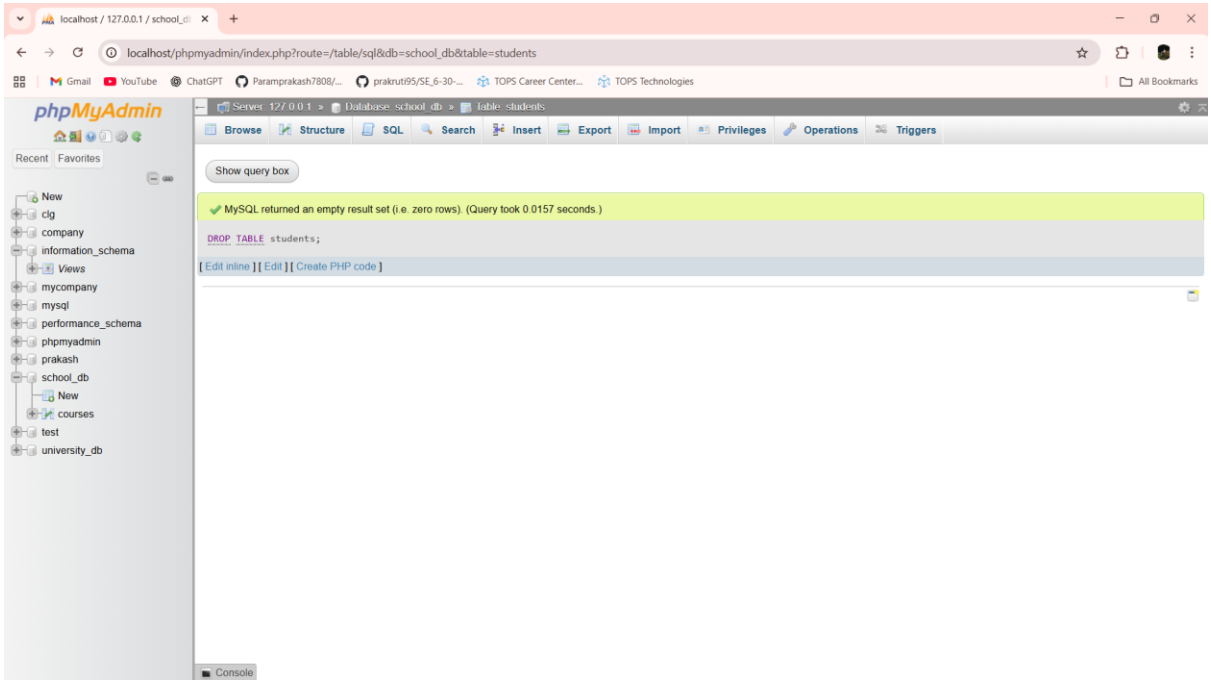
Lab 6.1: Drop the teachers table from the school_db database.

Ans:- DROP TABLE teachers



Lab 6.2: Drop the students table from the school_db database and verify that the table has been removed.

Ans:- DROP TABLE students

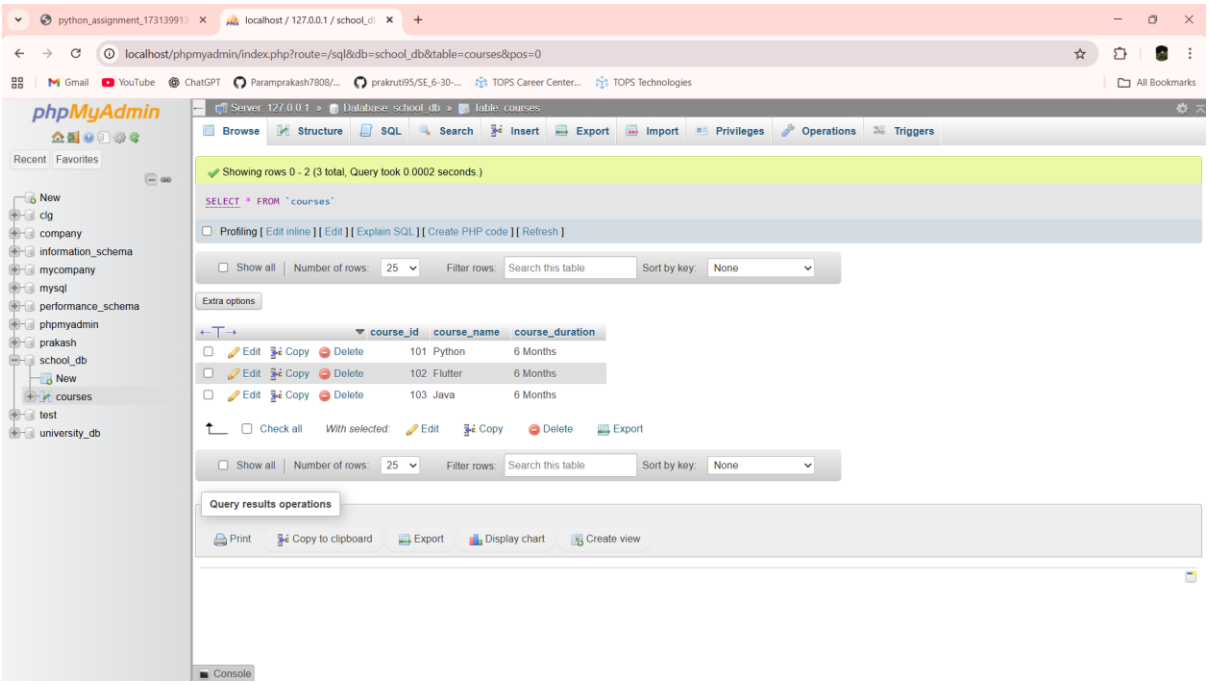


Lab 7.1: Insert three records into the courses table using the INSERT command.

Ans:- INSERT INTO courses (course_id,course_name,course_duration) VALUES (101,'Python','6 Months')

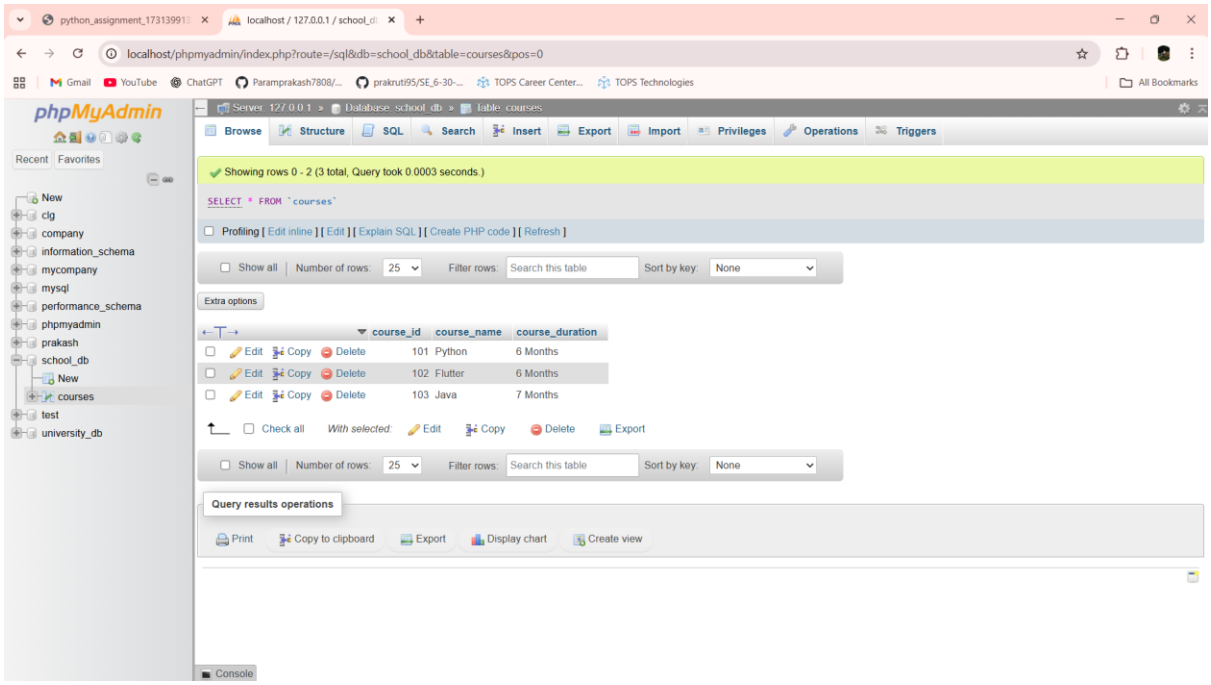
INSERT INTO courses (course_id,course_name,course_duration) VALUES (102,'Flutter','6 Months')

INSERT INTO courses (course_id,course_name,course_duration) VALUES (103,'Java','6 Months')



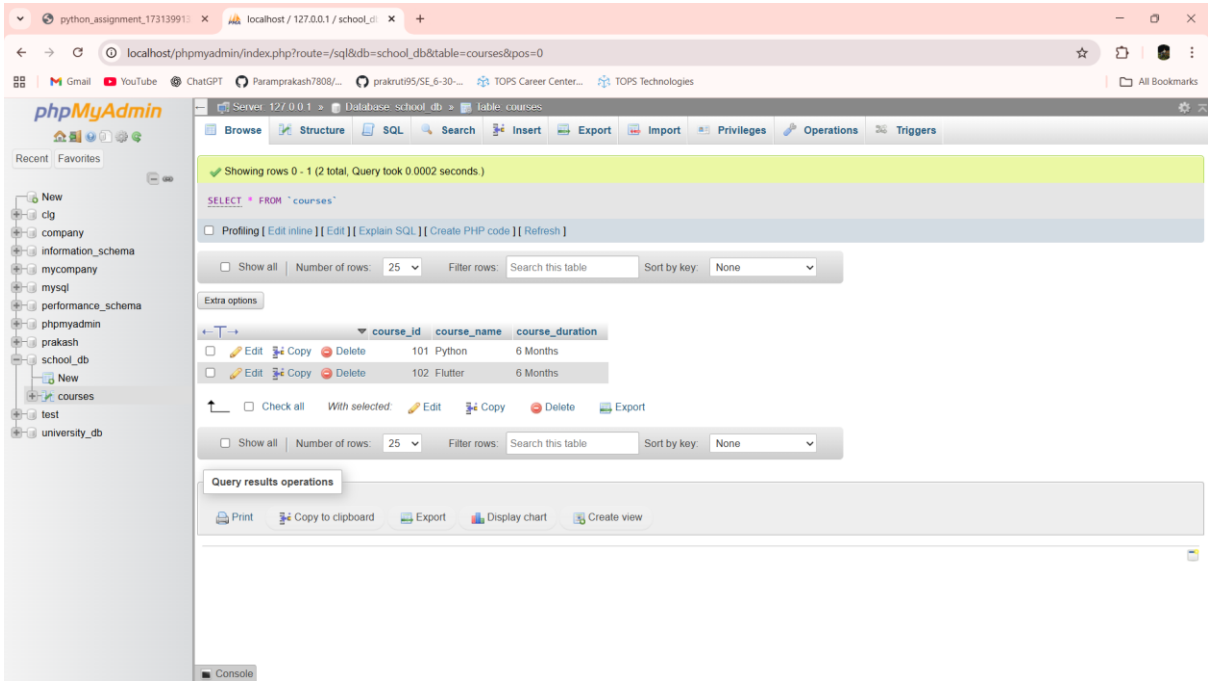
Lab 7.2: Update the course duration of a specific course using the UPDATE command.

Ans:- UPDATE courses SET course_duration = '7 Months' WHERE course_id = 103



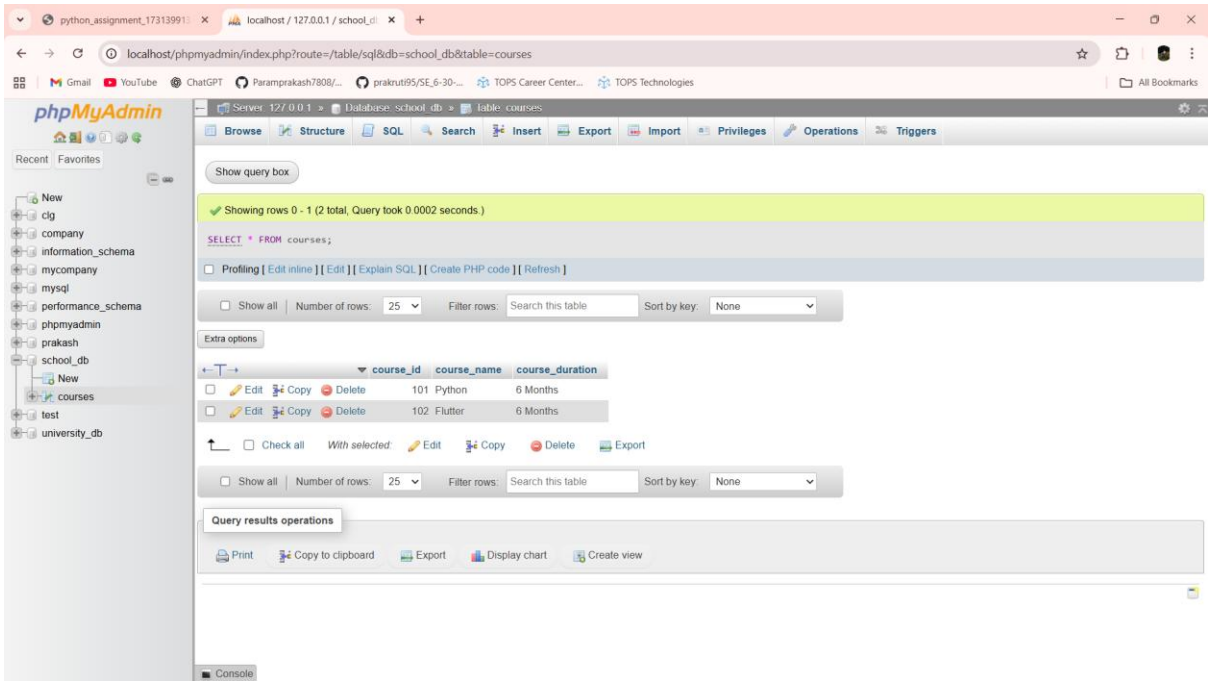
Lab 7.3: Delete a course with a specific course_id from the courses table using the DELETE command.

Ans:- DELETE FROM courses WHERE course_id = 103



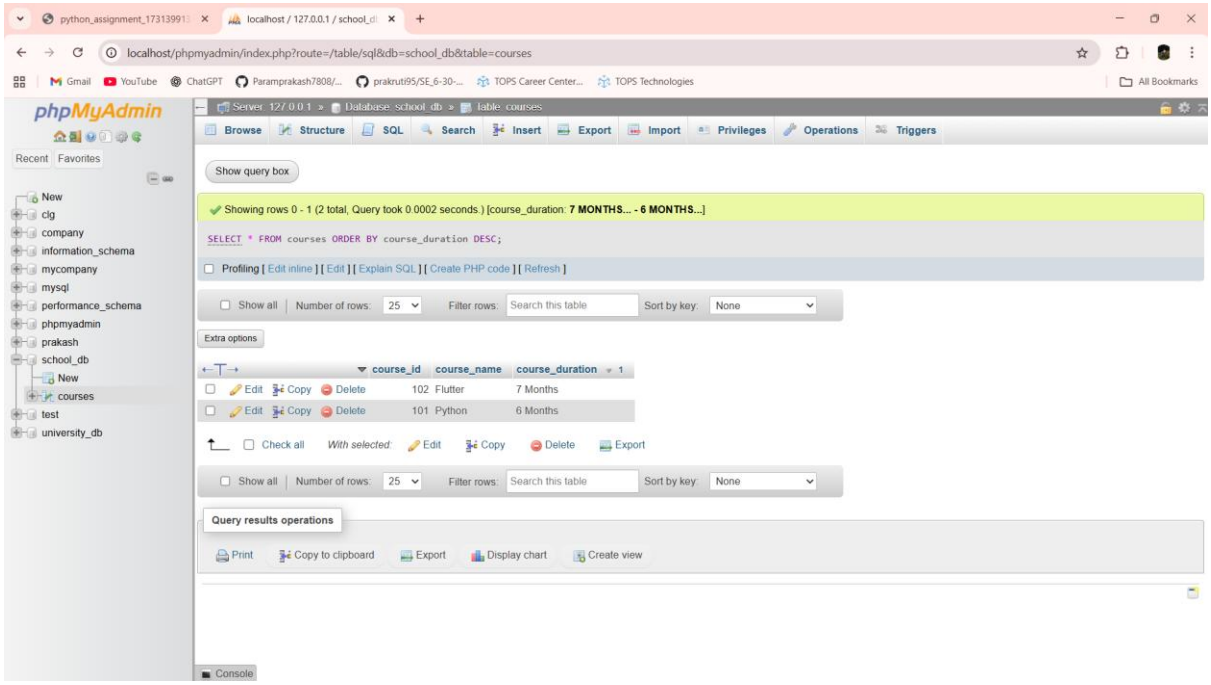
Lab 8.1: Retrieve all courses from the courses table using the SELECT statement.

Ans:- SELECT * FROM courses



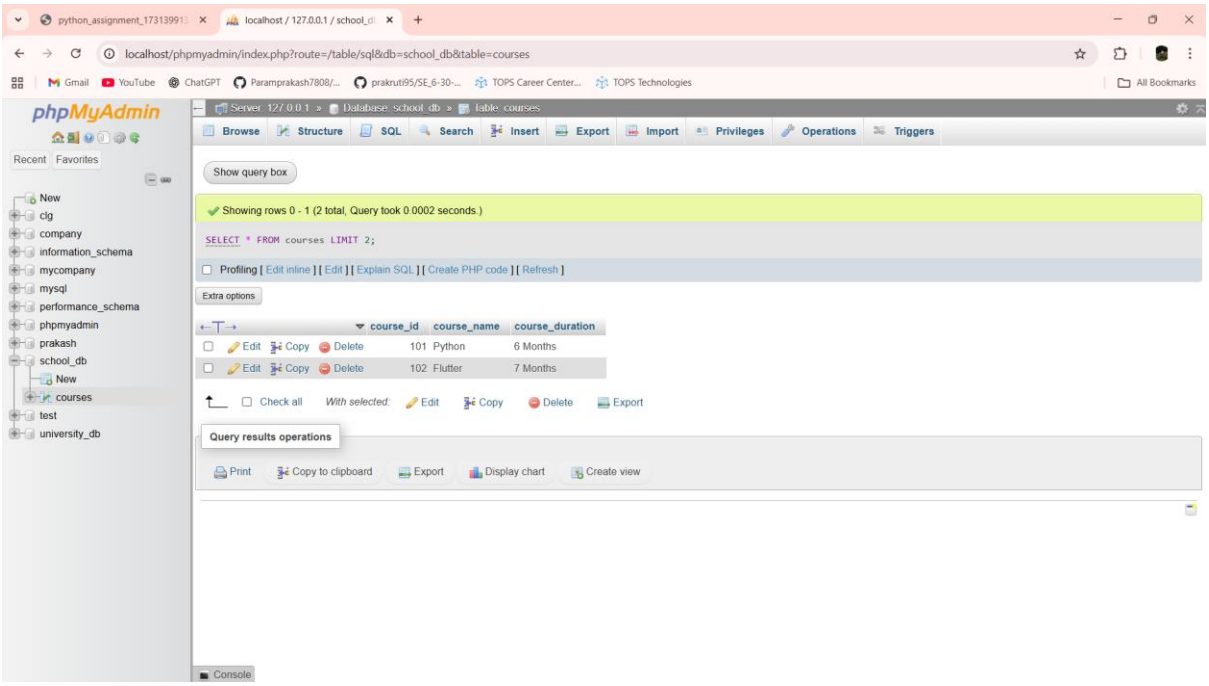
Lab 8.2: Sort the courses based on course_duration in descending order using ORDER BY.

Ans:- SELECT * FROM courses ORDER BY course_duration DESC



Lab 8.3: Limit the results of the SELECT query to show only the top two courses using LIMIT.

Ans:- SELECT * FROM courses LIMIT 2

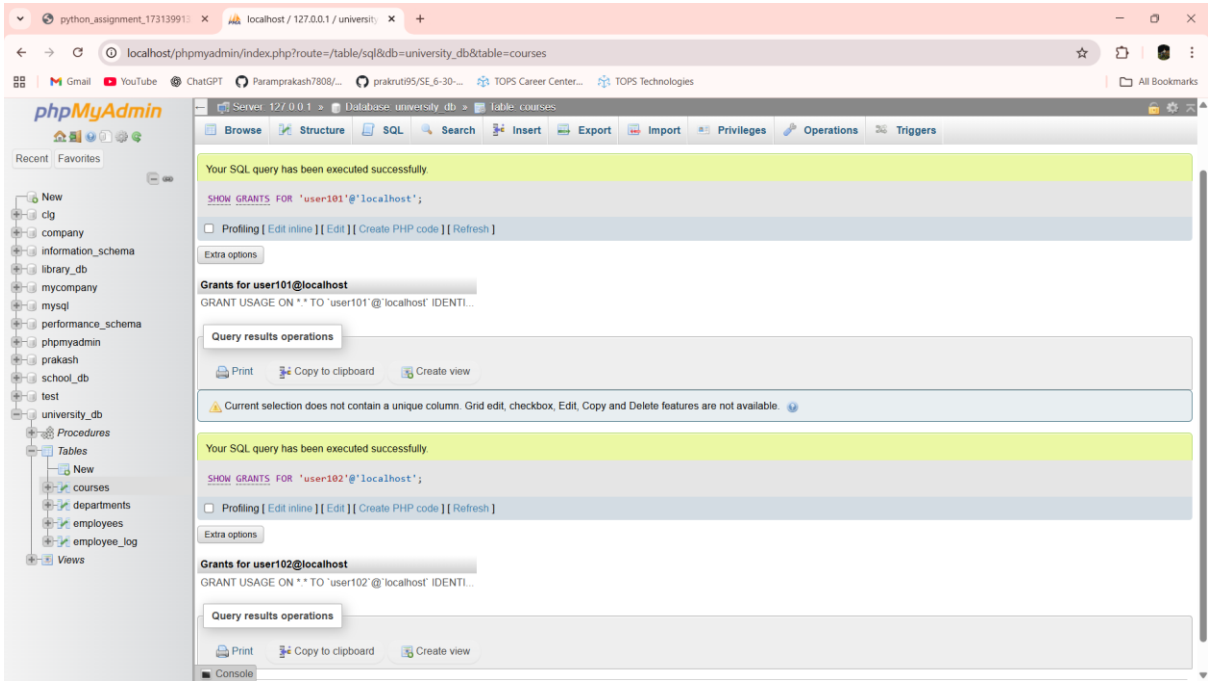


Lab 9.1: Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.

Ans:- CREATE USER 'user101'@'localhost' IDENTIFIED BY 'password1';

CREATE USER 'user102'@'localhost' IDENTIFIED BY 'password2';

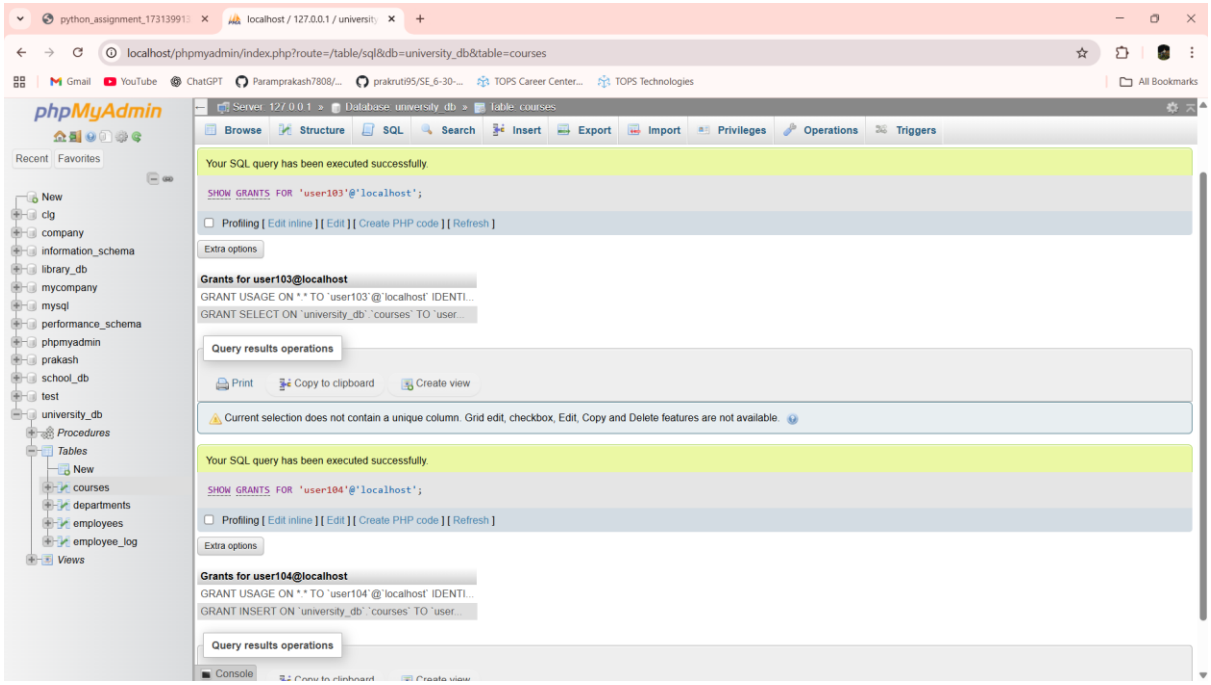
GRANT SELECT ON university_db.courses TO 'user101'@'localhost';



Lab 9.2: Revoke the INSERT permission from user1 and give it to user2.

Ans:- REVOKE INSERT ON university_db.courses FROM 'user101'@'localhost';

GRANT INSERT ON university_db.courses TO 'user102'@'localhost';



Lab 10.1: Insert a few rows into the courses table and use COMMIT to save the changes.

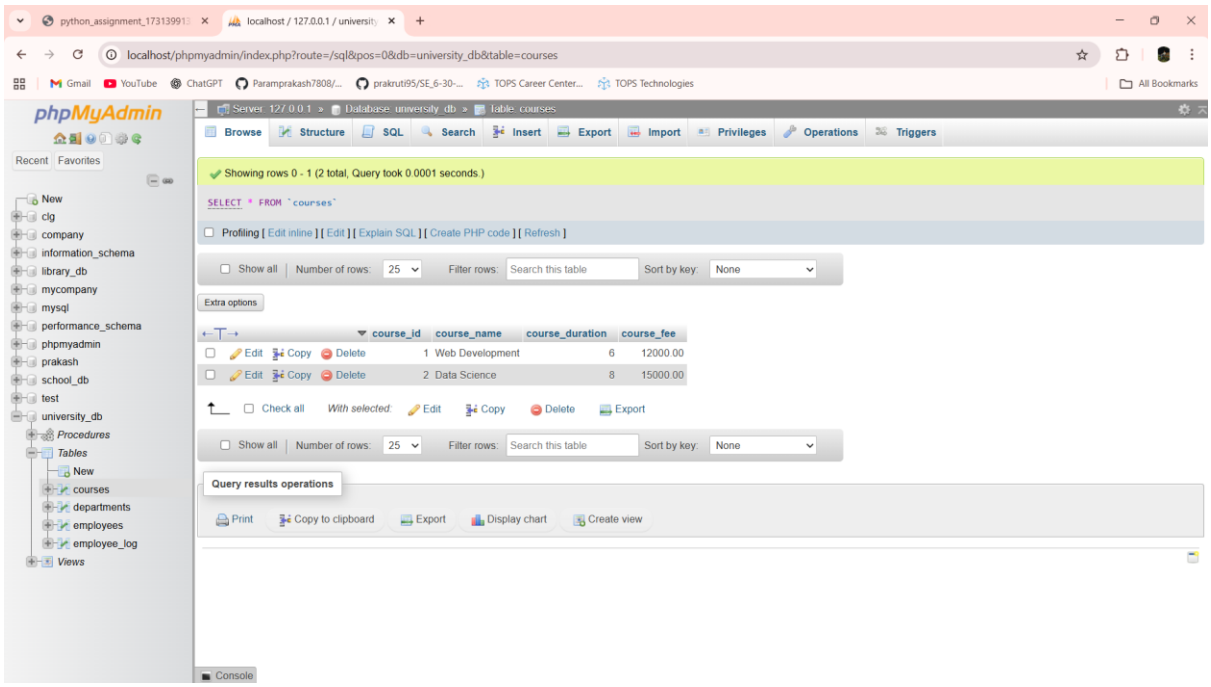
Ans:- START TRANSACTION;

INSERT INTO courses VALUES

(1, 'Web Development', 6, 12000),

(2, 'Data Science', 8, 15000);

COMMIT;



Lab 10.2: Insert additional rows, then use ROLLBACK to undo the last insert operation.

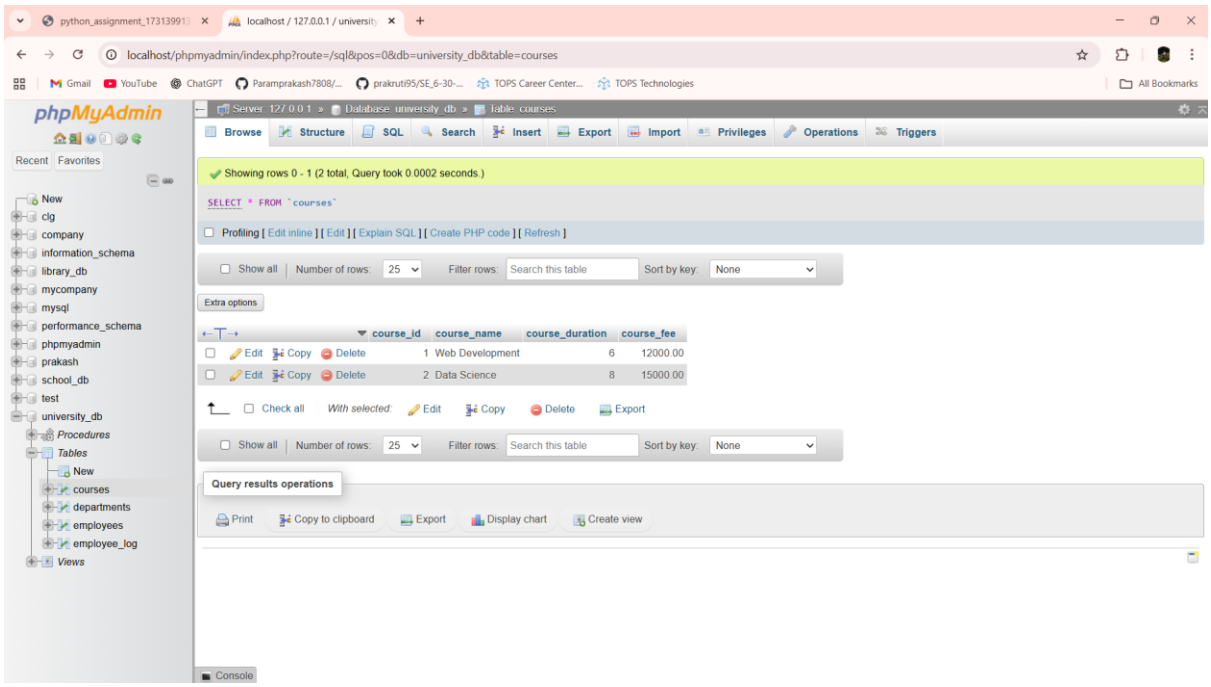
Ans:- START TRANSACTION;

INSERT INTO courses VALUES

(3, 'Python', 4, 8000),

(4, 'Java', 5, 10000);

ROLLBACK;



Lab 10.3: Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.

Ans:- START TRANSACTION;

SAVEPOINT before_update;

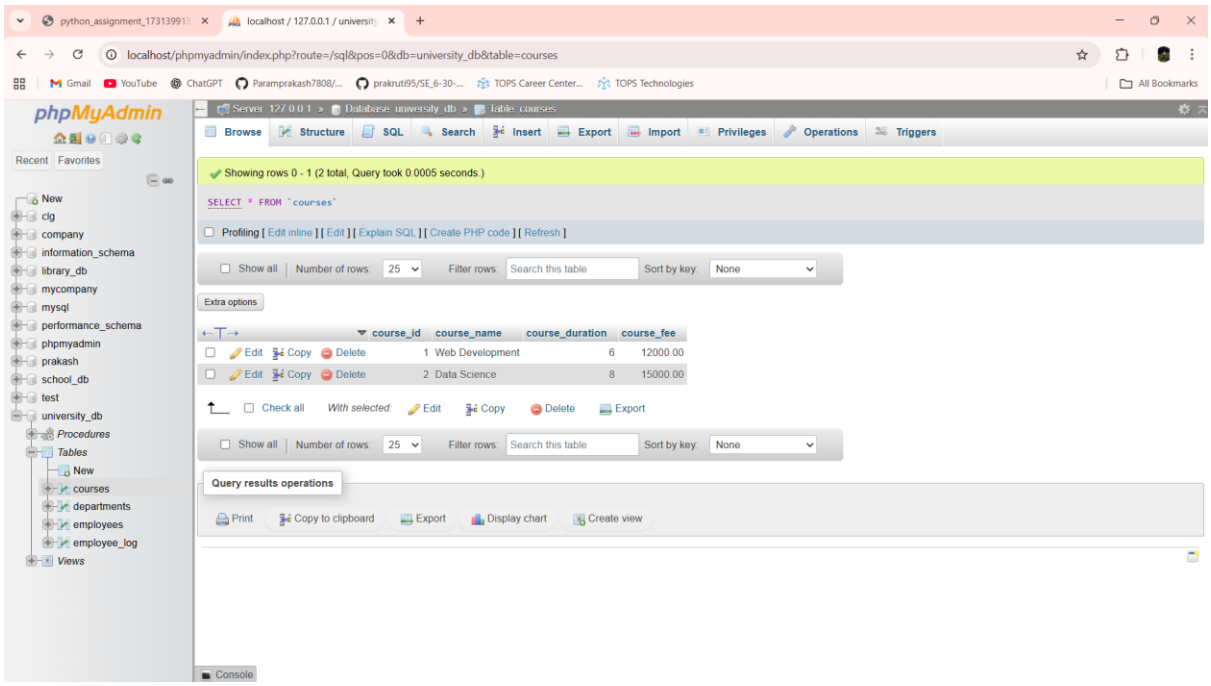
UPDATE courses

SET course_fee = 13000

WHERE course_id = 1;

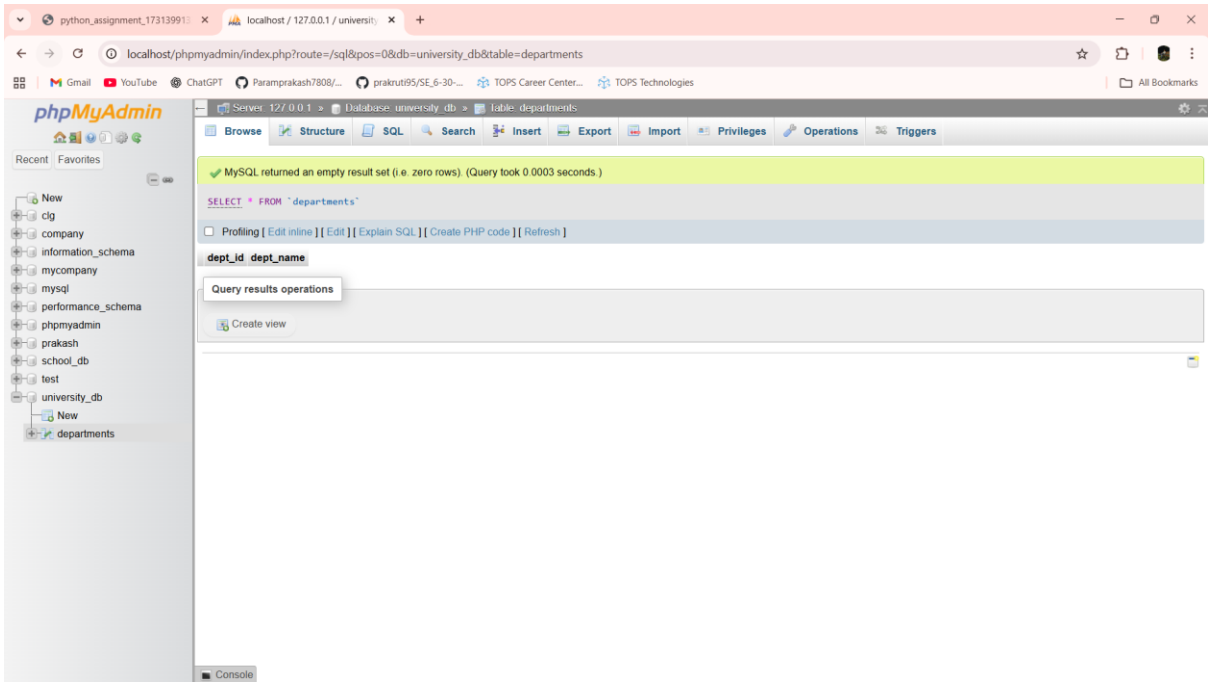
ROLLBACK TO before_update;

COMMIT;

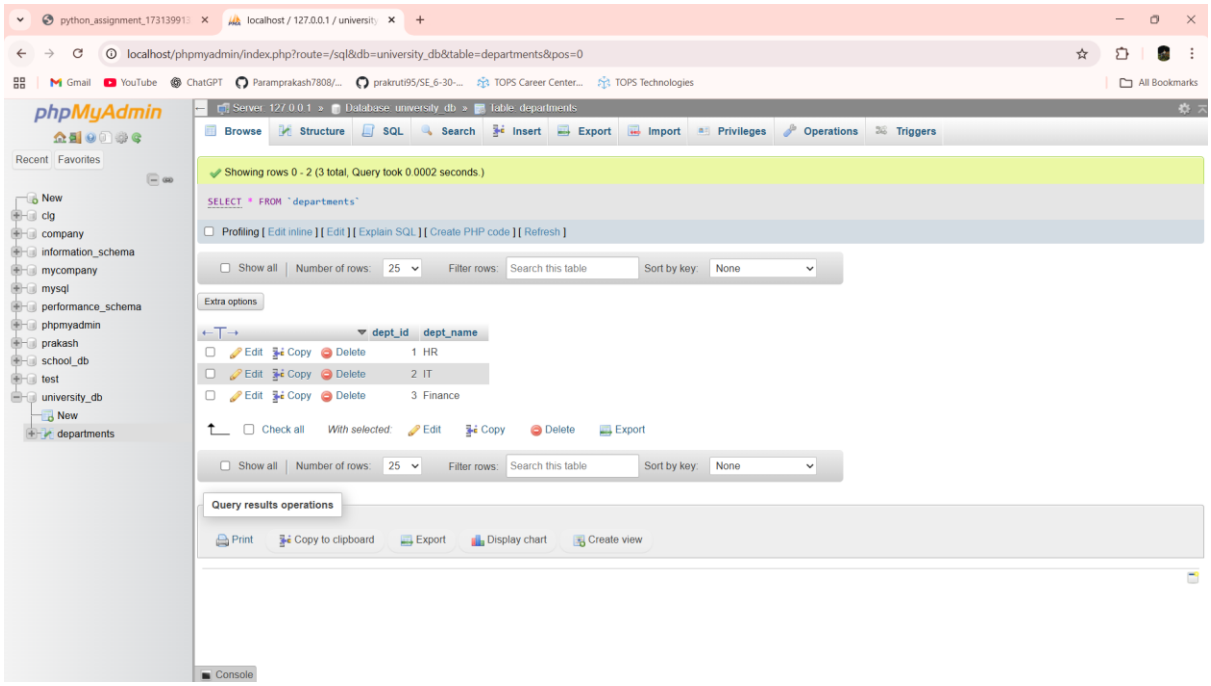


Lab 11.1: Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.

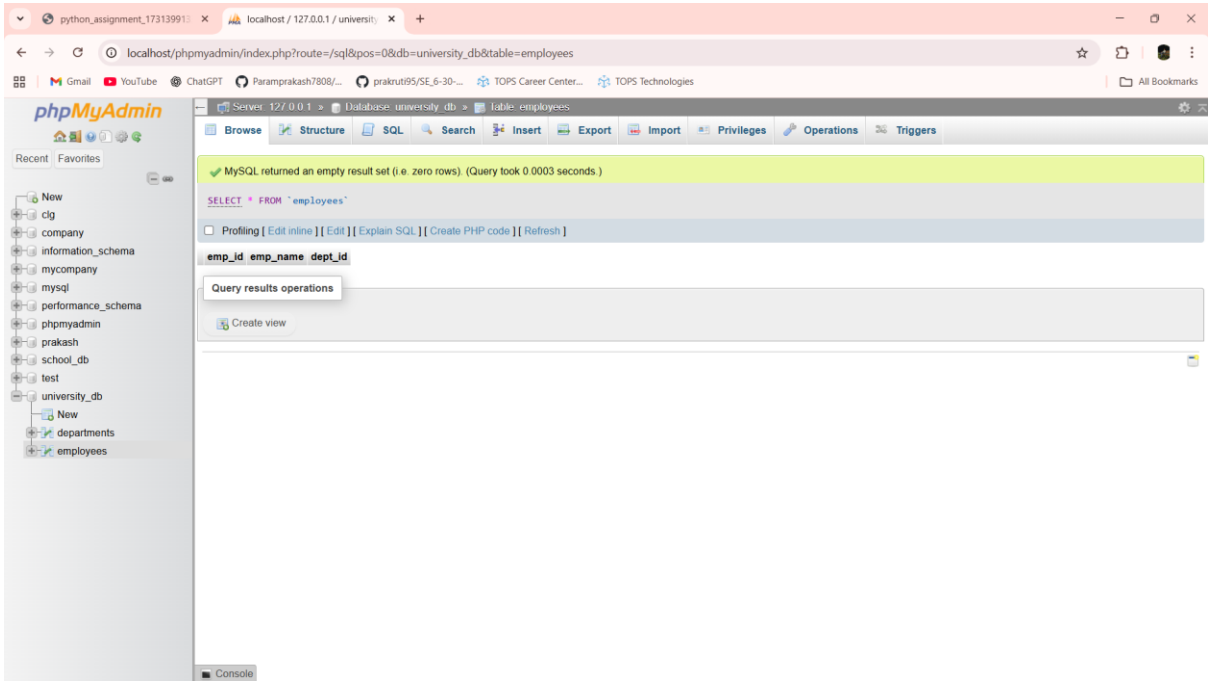
Ans:- CREATE TABLE departments (dept_id INT PRIMARY KEY,dept_name VARCHAR(30))



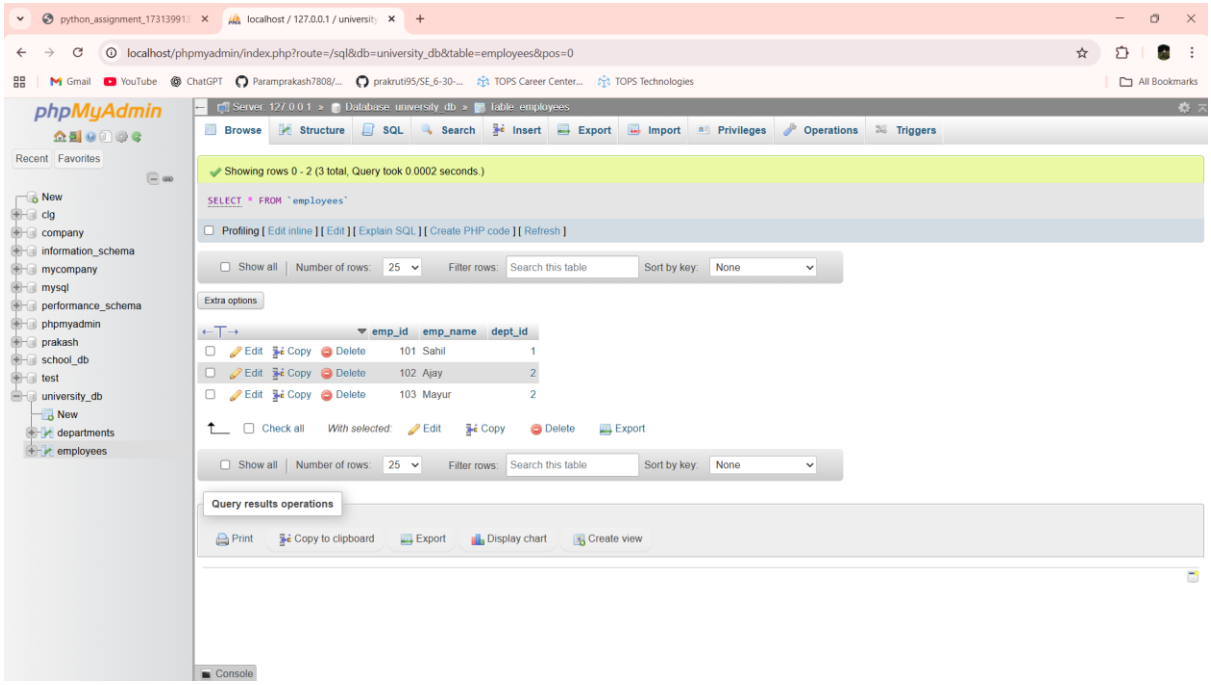
INSERT INTO departments VALUES(1, 'HR'),(2, 'IT'),(3, 'Finance')



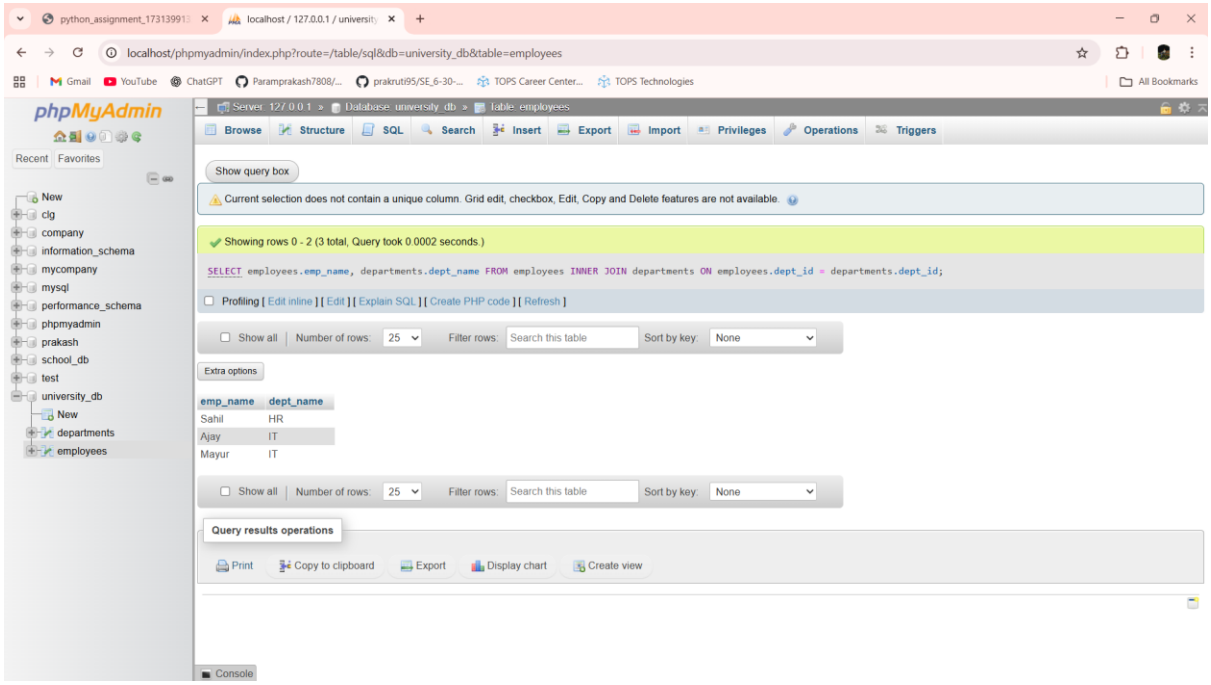
CREATE TABLE employees (emp_id INT PRIMARY KEY,emp_name VARCHAR(50),dept_id INT,FOREIGN KEY (dept_id) REFERENCES departments(dept_id))



INSERT INTO employees VALUES(101, 'Sahil', 1),(102, 'Ajay', 2),(103, 'Mayur', 2)

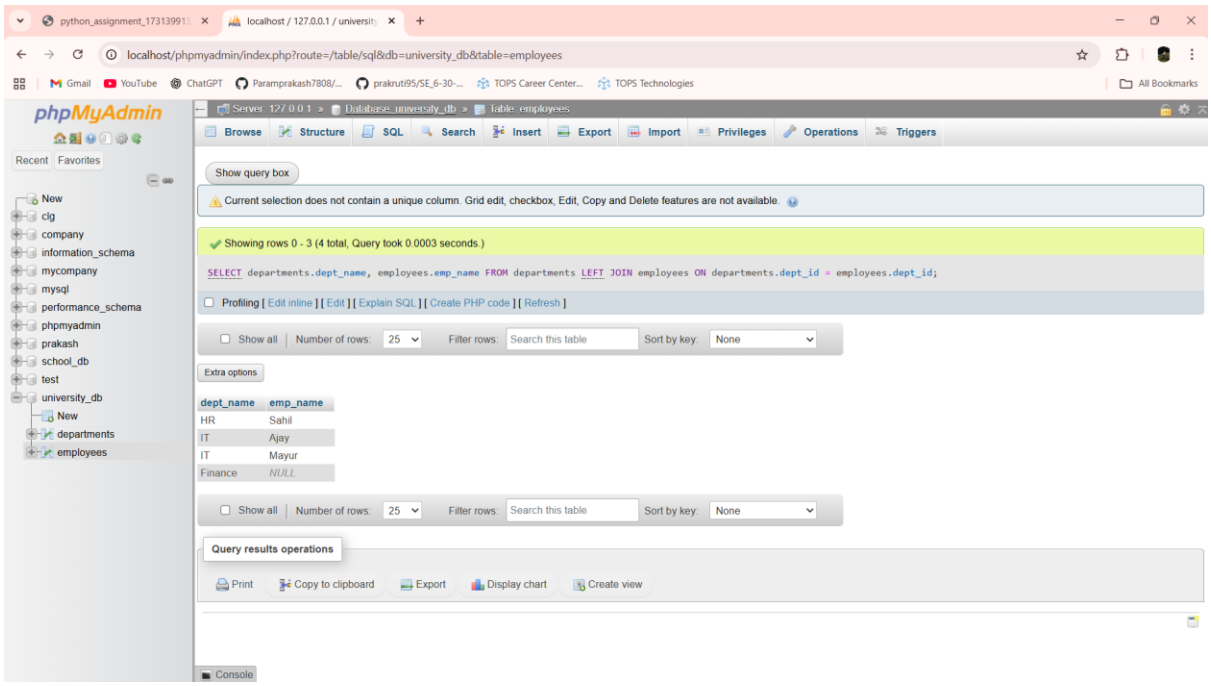


SELECT employees.emp_name, departments.dept_name FROM employees INNER JOIN departments ON employees.dept_id = departments.dept_id



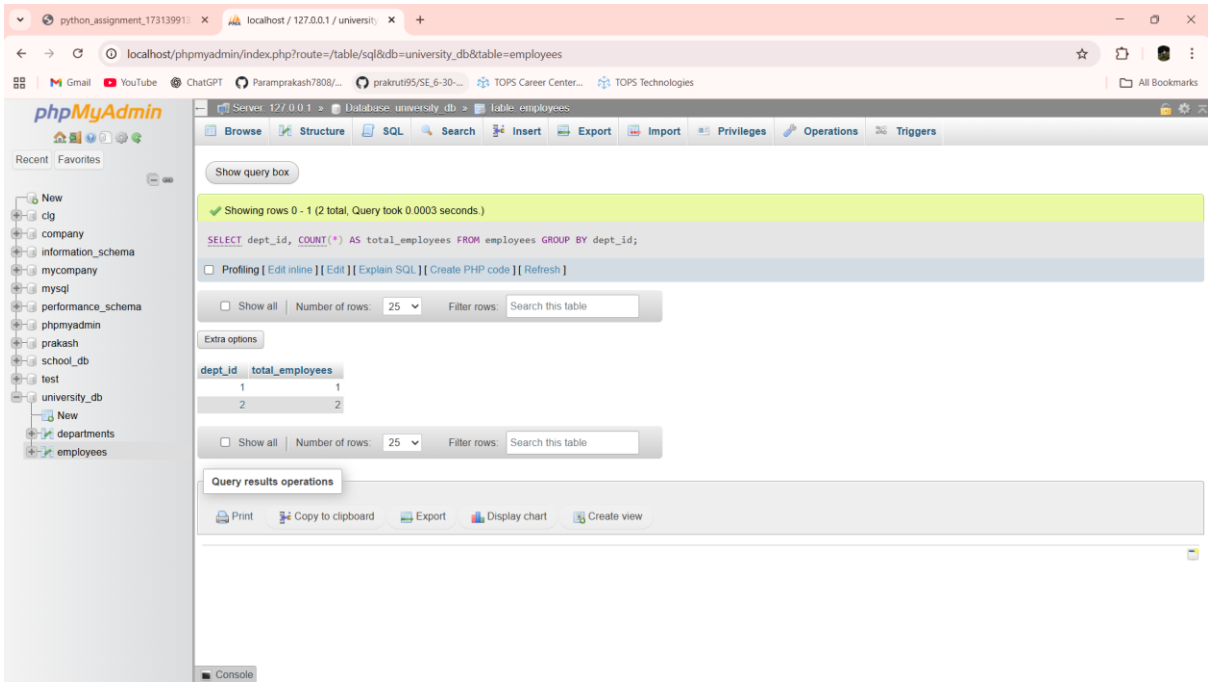
Lab 11.2: Use a LEFT JOIN to show all departments, even those without employees.

Ans:- SELECT departments.dept_name, employees.emp_name FROM departments LEFT JOIN employees ON departments.dept_id = employees.dept_id



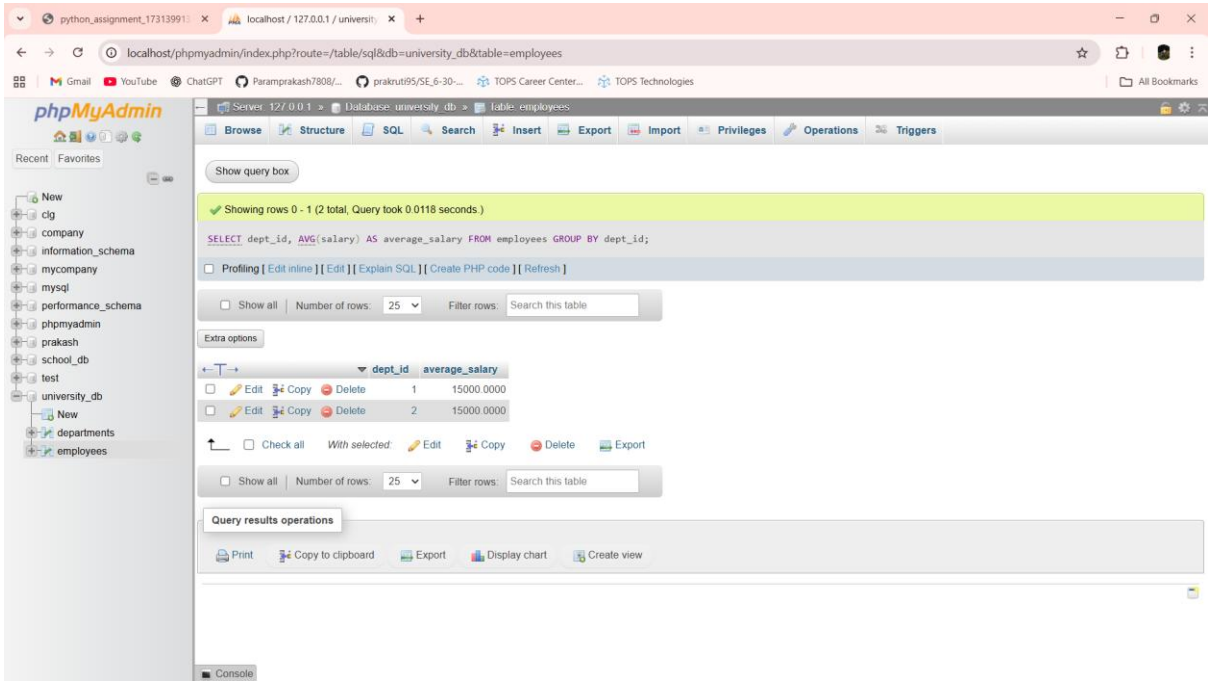
Lab 12.1: Group employees by department and count the number of employees in each department using GROUP BY.

Ans:- SELECT dept_id, COUNT(*) AS total_employees FROM employees GROUP BY dept_id



Lab 12.2: Use the AVG aggregate function to find the average salary of employees in each department.

Ans:- SELECT dept_id, AVG(salary) AS average_salary FROM employees GROUP BY dept_id



Lab 13.1: Write a stored procedure to retrieve all employees from the employees table based on department.

Ans:- DELIMITER $$

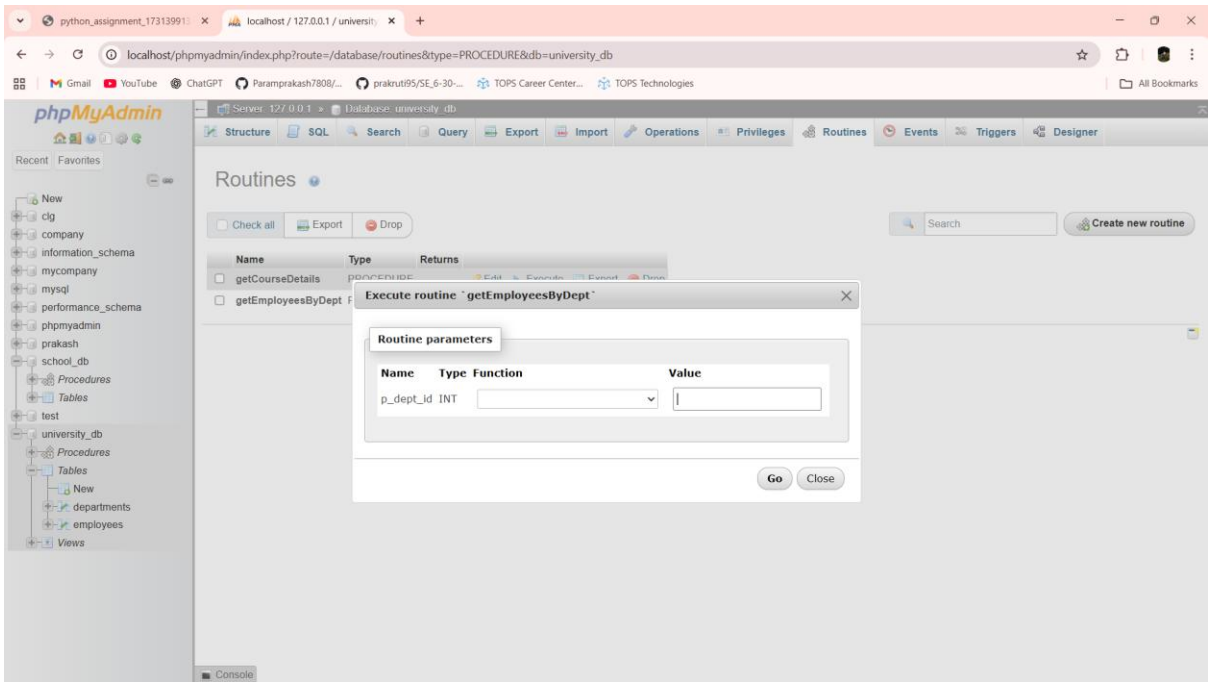
CREATE PROCEDURE getEmployeesByDept(IN p_dept_id INT)

BEGIN

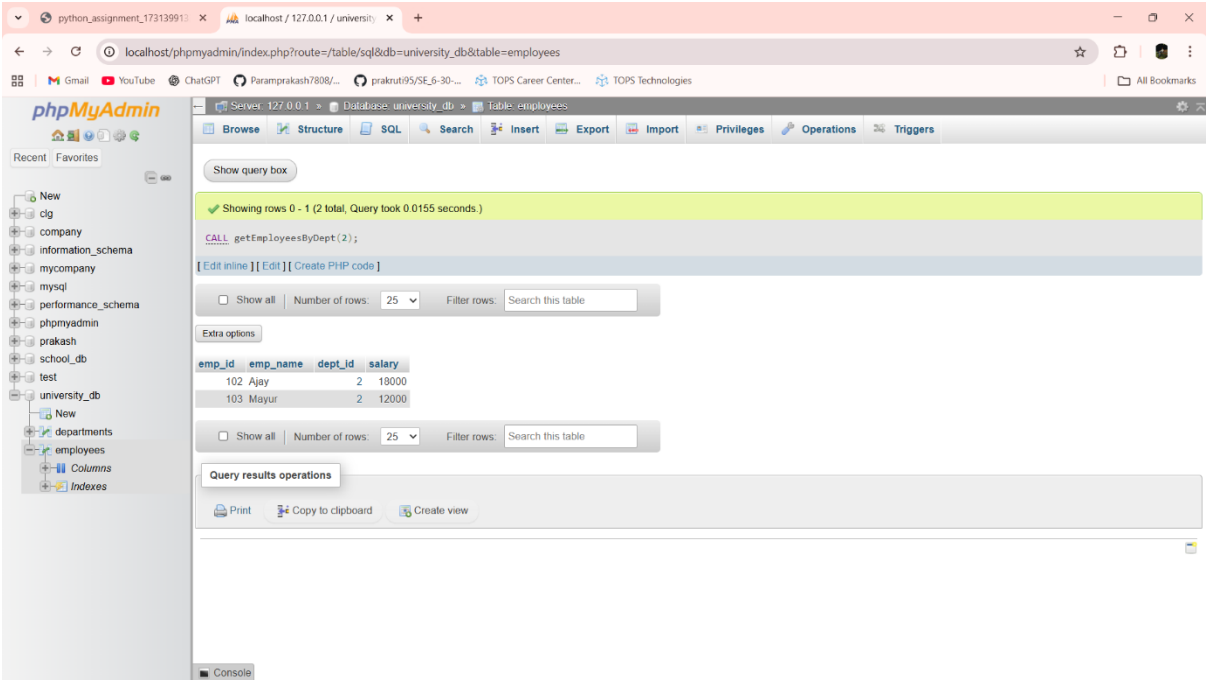
  SELECT * FROM employees WHERE dept_id = p_dept_id;

END $$

DELIMITER ;



CALL getEmployeesByDept(2);



Lab 13.2: Write a stored procedure that accepts course_id as input and returns the course details.

Ans:- DELIMITER $$

CREATE PROCEDURE getCourseDetails(IN p_course_id INT)

BEGIN

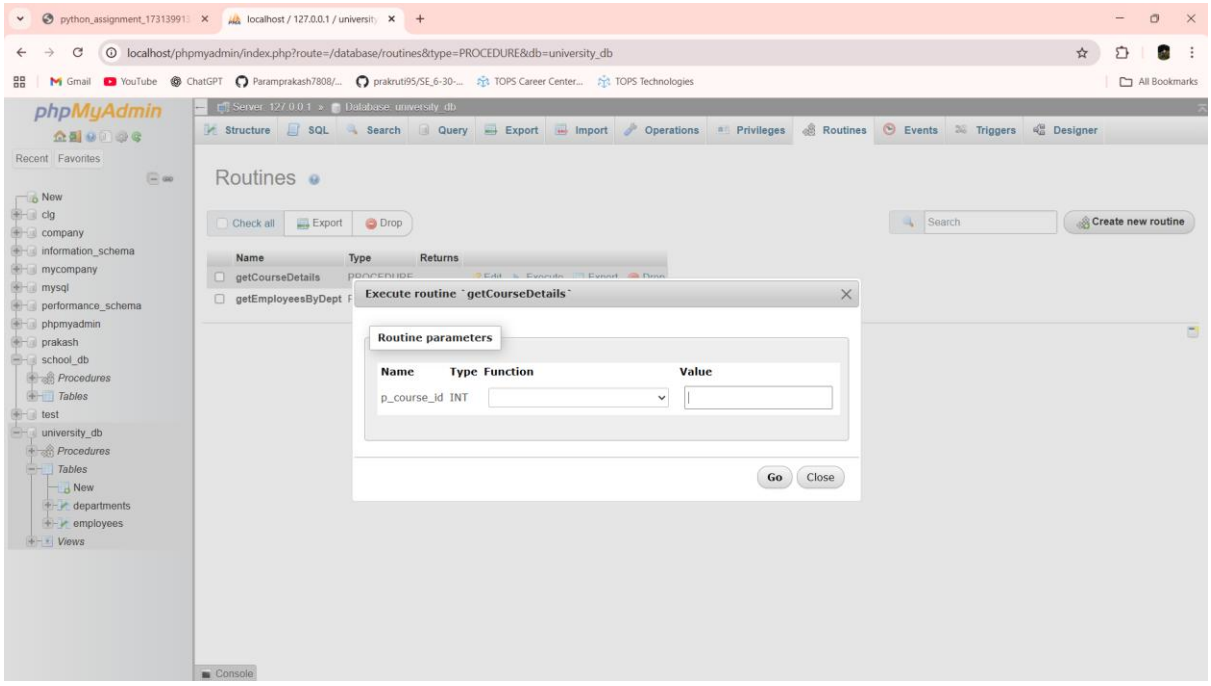
  SELECT *

  FROM courses

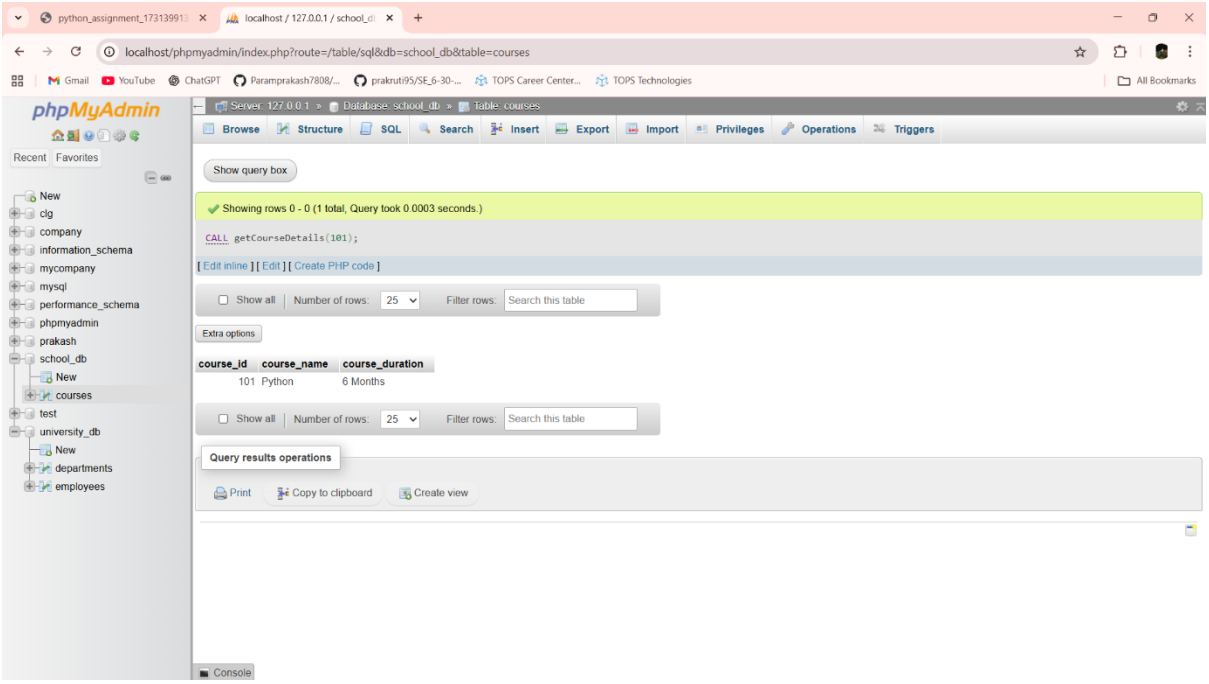
  WHERE course_id = p_course_id;

END $$

DELIMITER ;



CALL getCourseDetails(101)



Lab 14.1: Create a view to show all employees along with their department names.

Ans:- CREATE VIEW emp_dept_view AS

SELECT

e.emp_id,

e.emp_name,

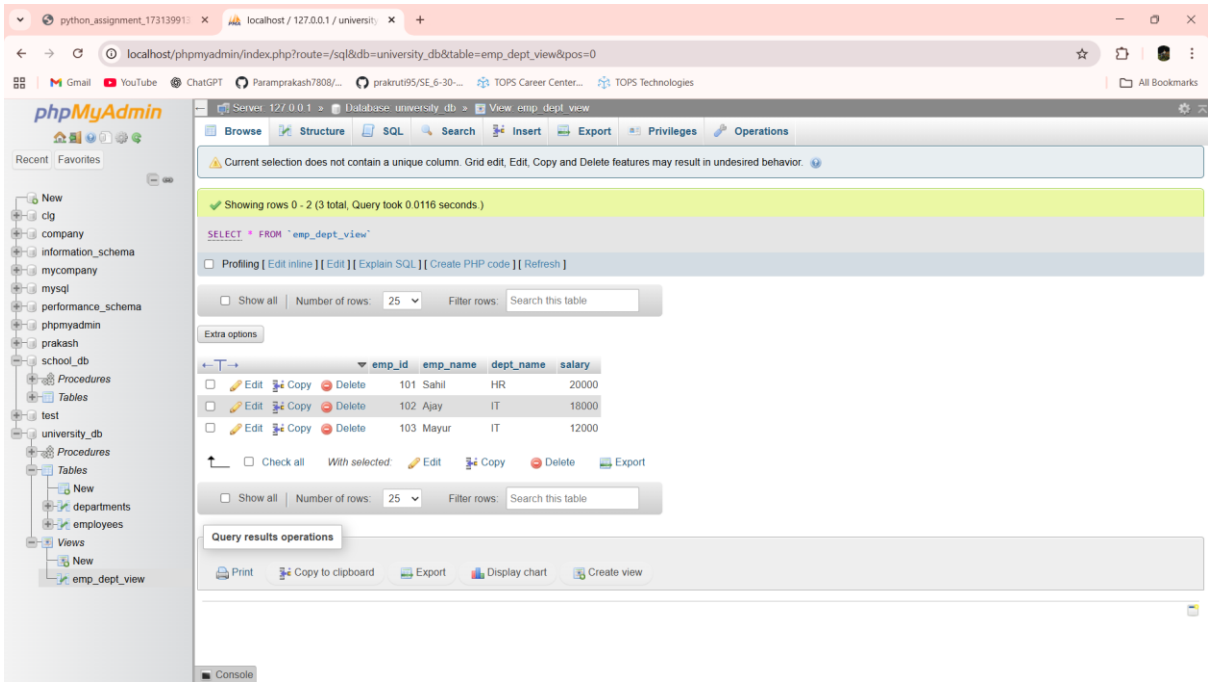
d.dept_name,

e.salary

FROM employees e

INNER JOIN departments d

ON e.dept_id = d.dept_id;



Lab 14.2: Modify the view to exclude employees whose salaries are below $50,000.

Ans:- CREATE OR REPLACE VIEW emp_dept_view AS

SELECT

e.emp_id,

e.emp_name,

d.dept_name,

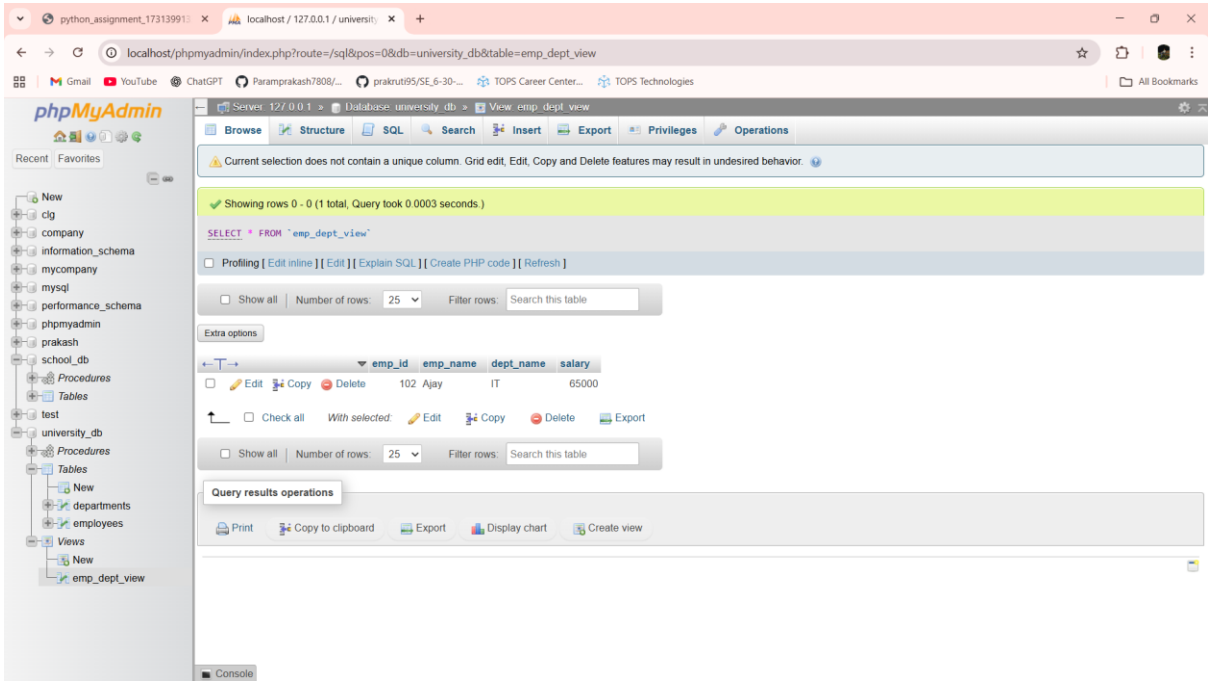
e.salary

FROM employees e

INNER JOIN departments d

ON e.dept_id = d.dept_id

WHERE e.salary >= 50000;



Lab 15.1: Create a trigger to automatically log changes to the employees table when a new employee is added.

Ans:- CREATE TABLE employee_log (

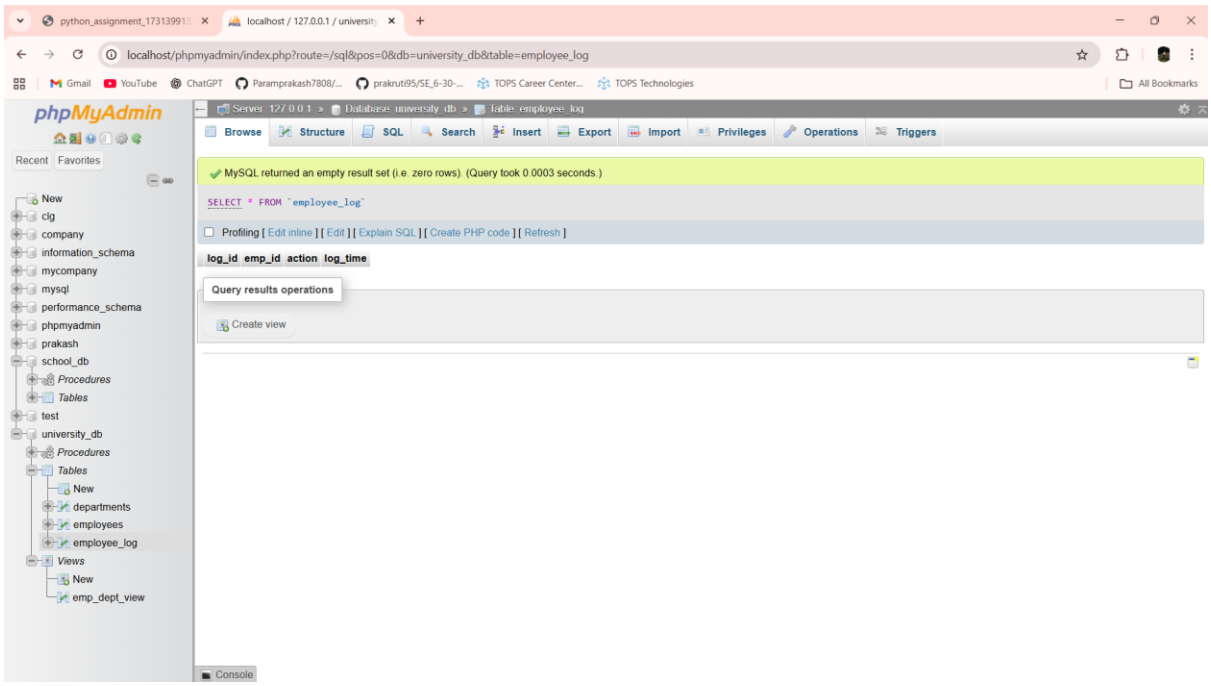
log_id INT AUTO_INCREMENT PRIMARY KEY,

emp_id INT,

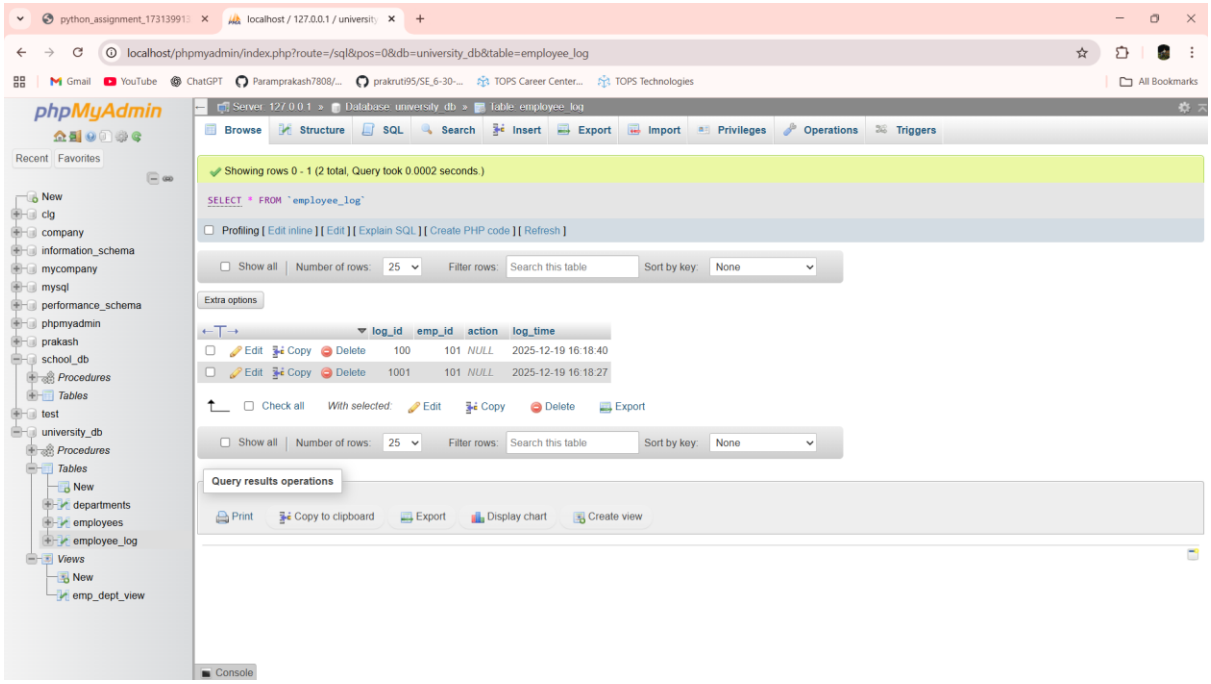
action VARCHAR(50),

log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);



```
DELIMITER $$

CREATE TRIGGER after_employee_insert

AFTER INSERT ON employees

FOR EACH ROW

BEGIN

  INSERT INTO employee_log (emp_id, action)

  VALUES (NEW.emp_id, 'INSERT');

END $$

DELIMITER ;
```



Lab 15.2: Create a trigger to update the last_modified timestamp whenever an employee record is updated.

Ans:- DELIMITER $$

CREATE TRIGGER before_employee_update

BEFORE UPDATE ON employees

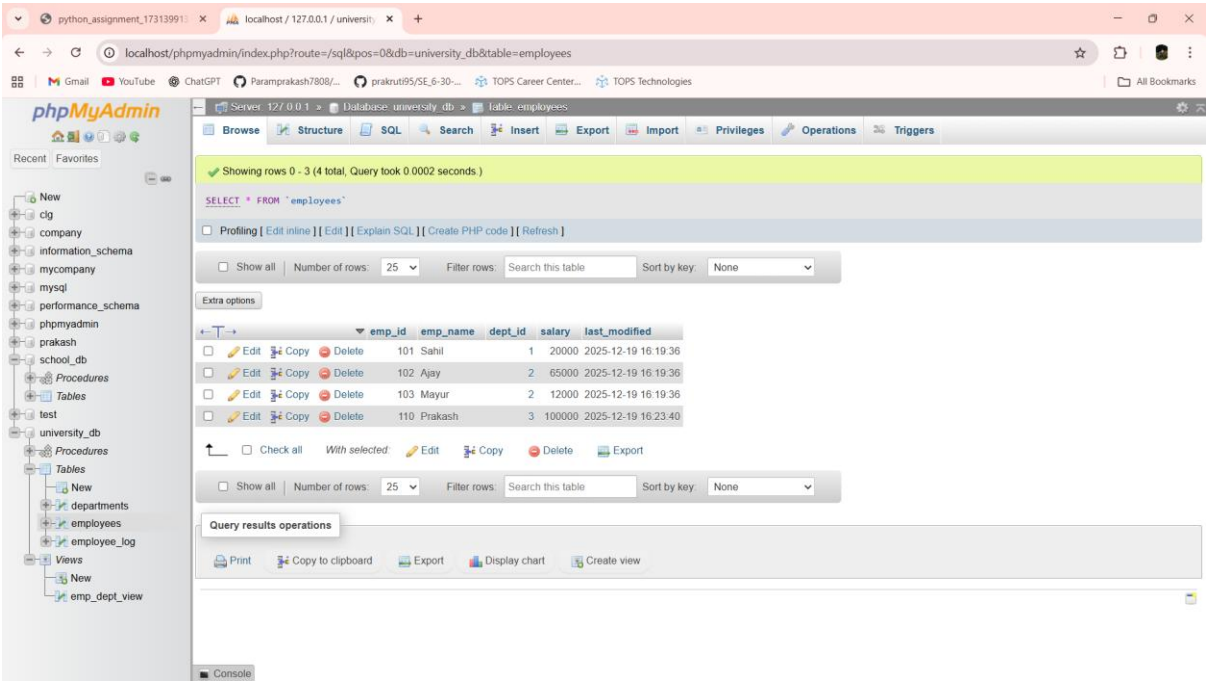
FOR EACH ROW

BEGIN

  SET NEW.last_modified = CURRENT_TIMESTAMP;

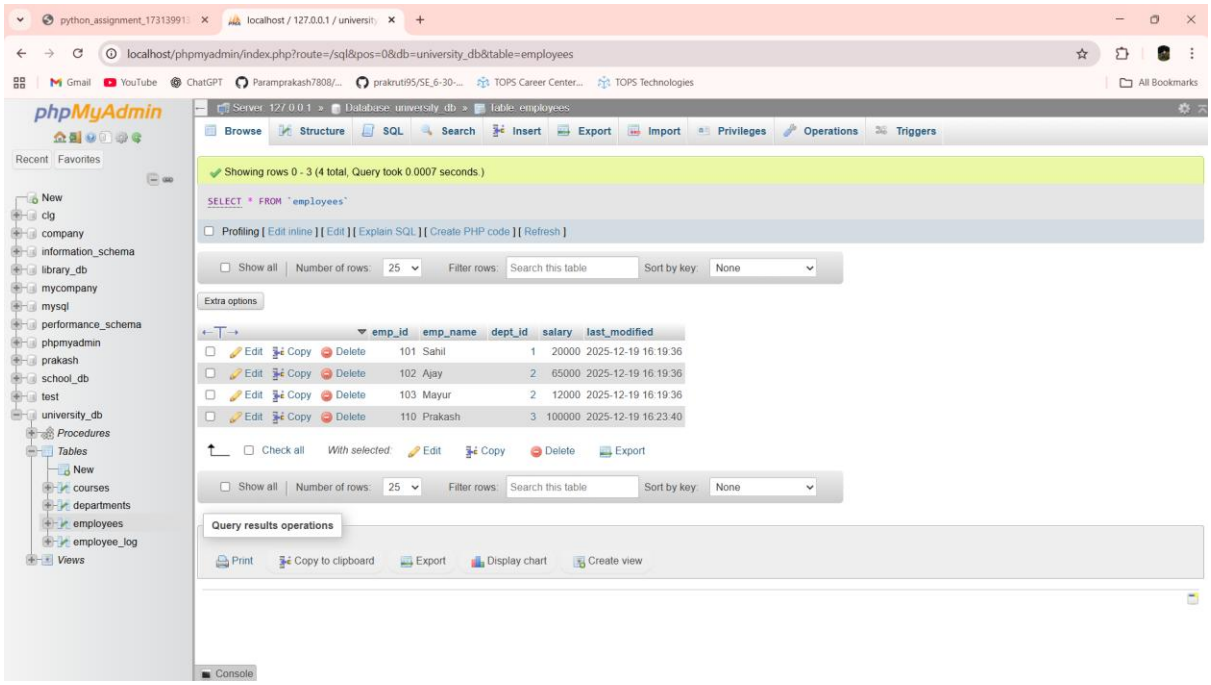
END $$

DELIMITER ;

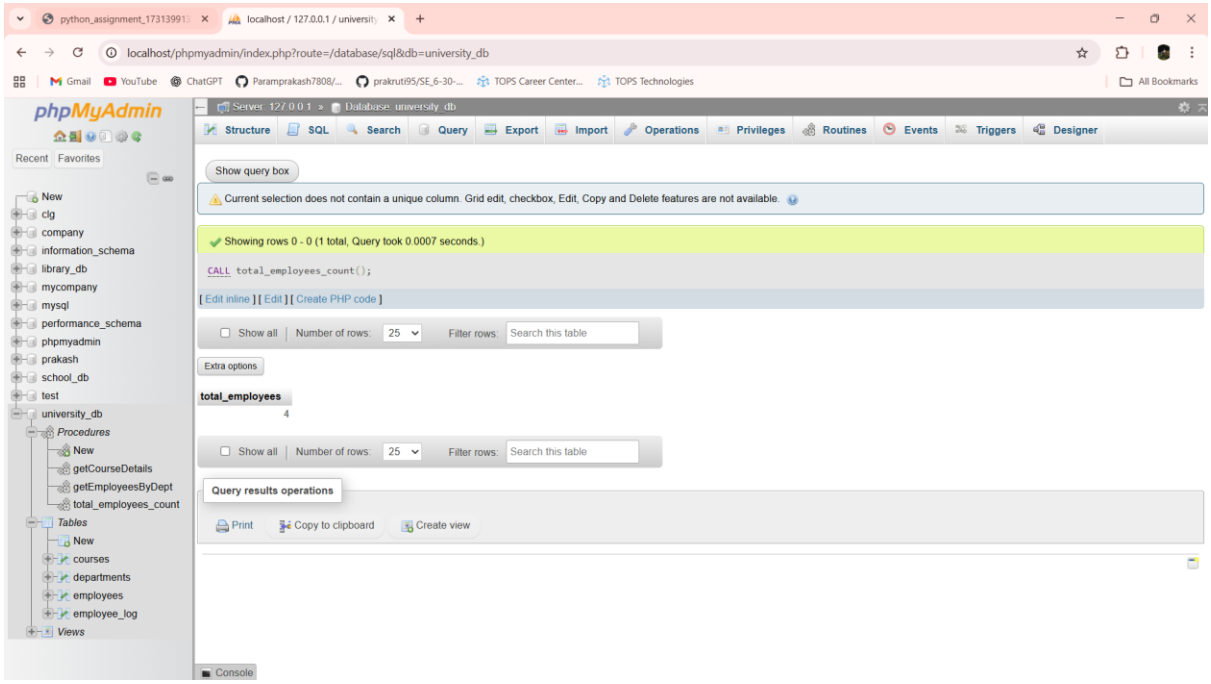


Lab 16.1: Write a PL/SQL block to print the total number of employees from the employees table.

Ans:-

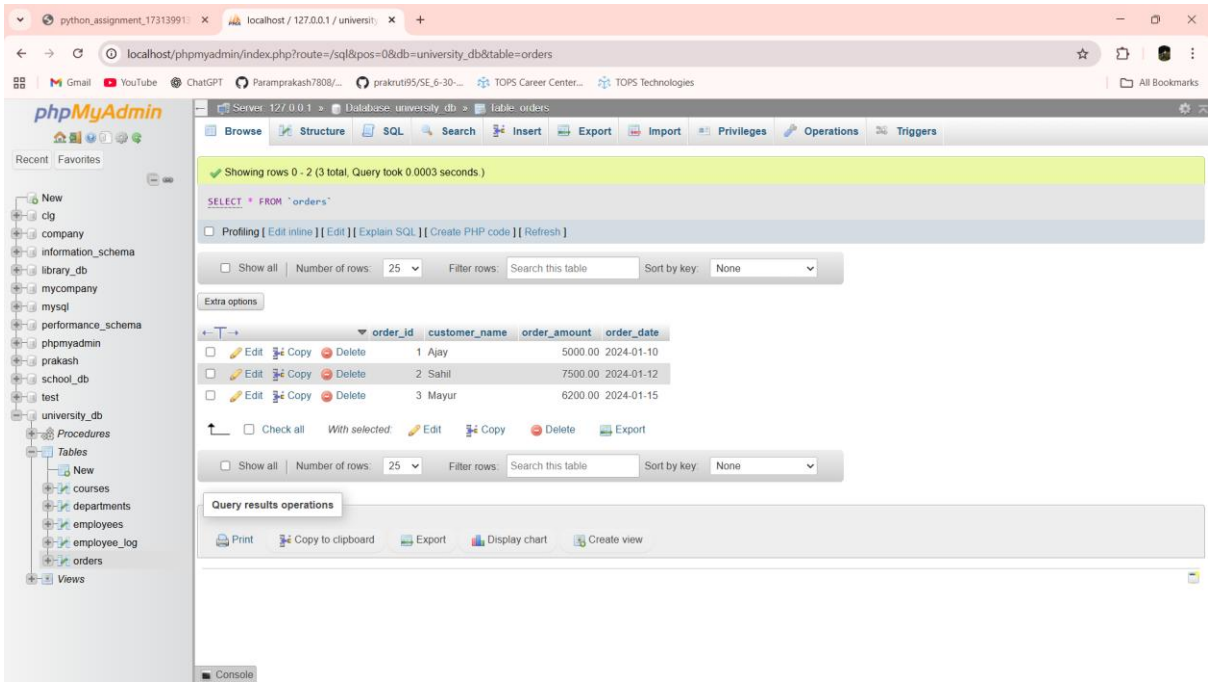


```
DELIMITER $$

CREATE PROCEDURE total_employees_count()

BEGIN

   DECLARE total_employees INT;

   SELECT COUNT(*) INTO total_employees

   FROM employees;

   SELECT total_employees AS total_employees;

END $$

DELIMITER ;

CALL total_employees_count();
```

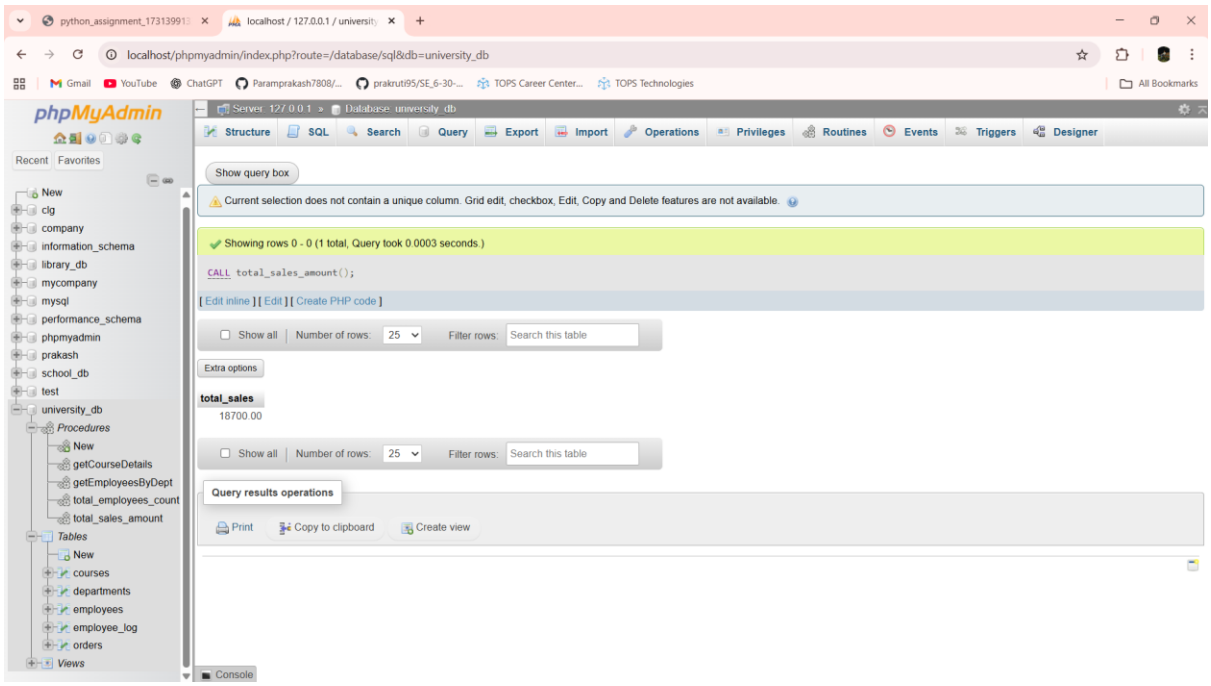


Lab 16.2: Create a PL/SQL block that calculates the total sales from an orders table.

Ans:-



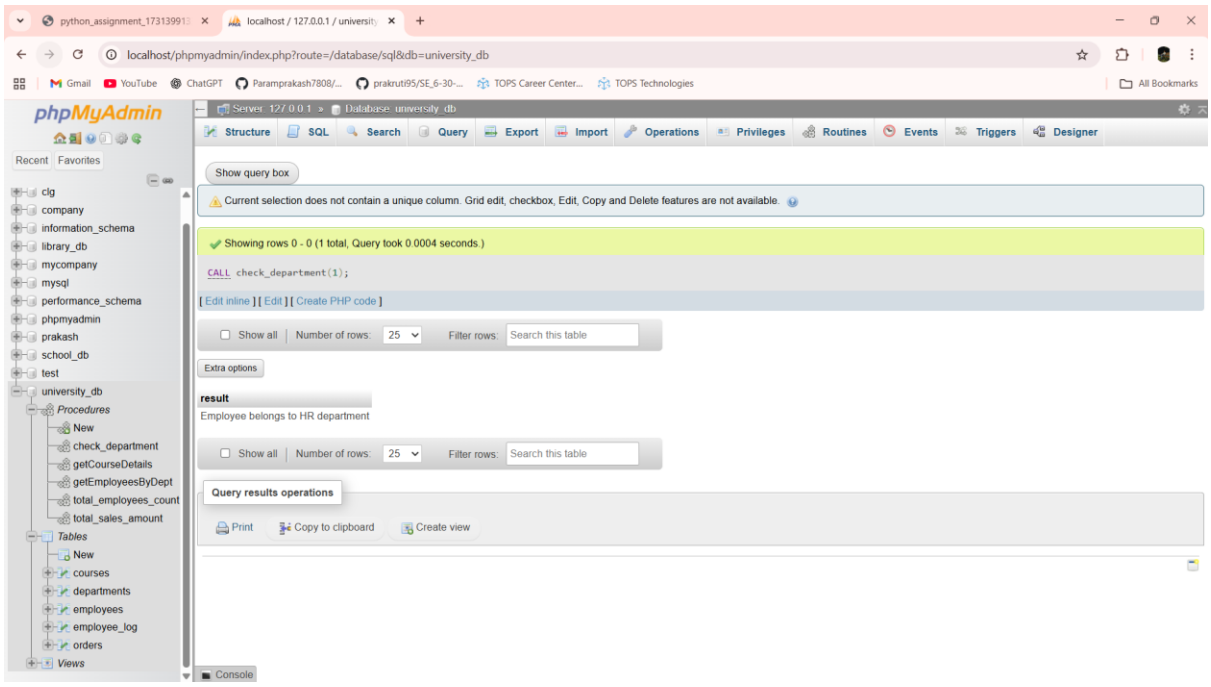
```
DELIMITER $$

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();
```



Lab 17.1: Write a PL/SQL block using an IF-THEN condition to check the department of an employee.

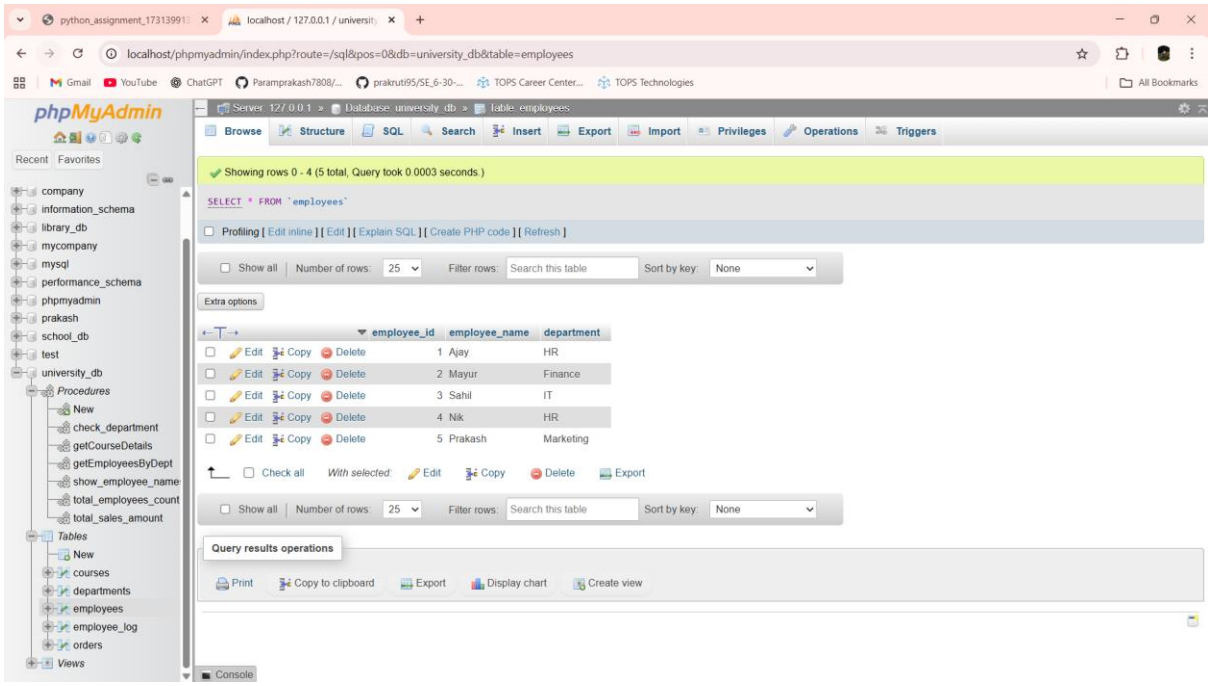
Ans:- DELIMITER $$

```
CREATE PROCEDURE check_department(IN emp_id INT)
BEGIN
  DECLARE dept VARCHAR(50);
  SELECT department INTO dept
  FROM employees
  WHERE employee_id = emp_id;
  IF dept = 'HR' THEN
    SELECT 'Employee belongs to HR department' AS result;
  ELSE
    SELECT 'Employee belongs to another department' AS result;
  END IF;
END $$
DELIMITER ;
CALL check_department(1);
```



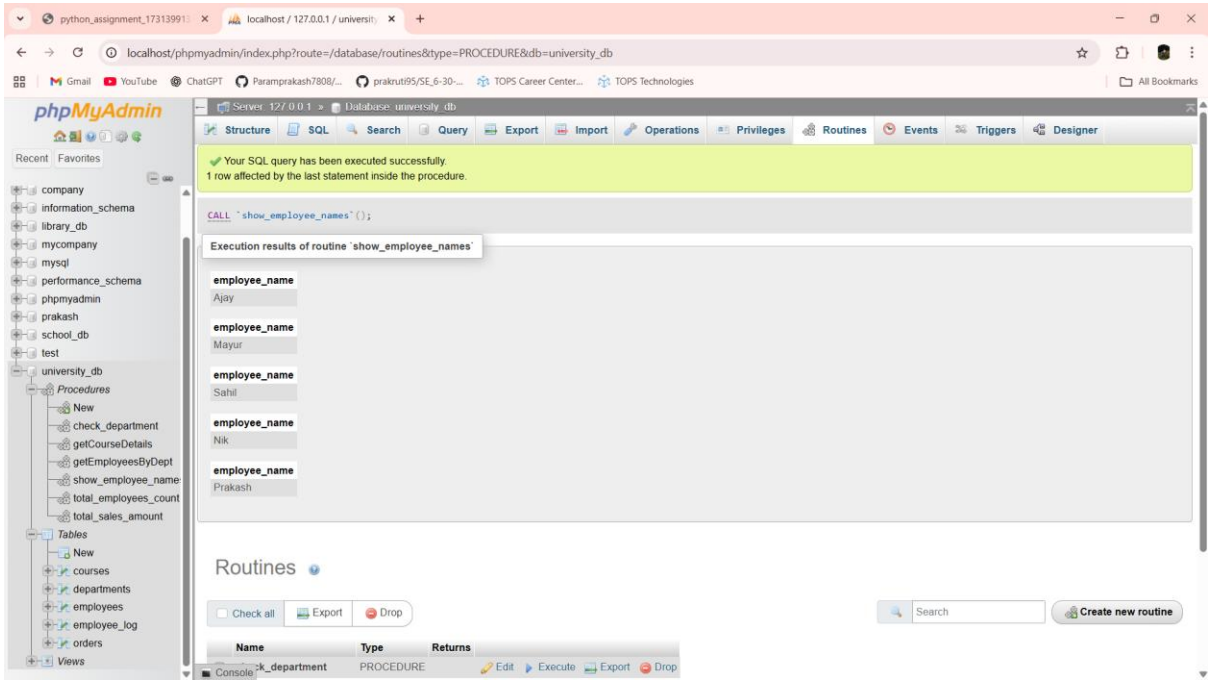
Lab 17.2: Use a FOR LOOP to iterate through employee records and display their names.

Ans:-




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

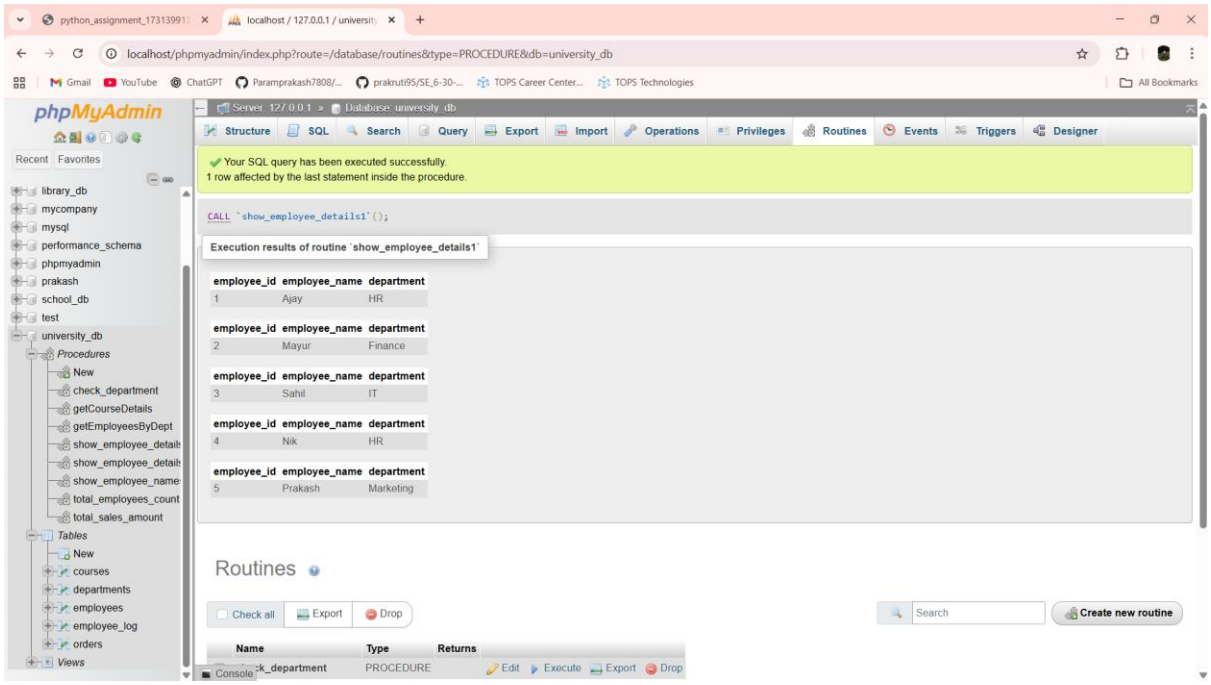
```
    SELECT emp_name AS employee_name;
  END LOOP;
  CLOSE emp_cursor;
END $$
DELIMITER ;
CALL show_employee_names();
```



Lab 18.1: Write a PL/SQL block using an explicit cursor to retrieve and display employee details.

Ans:- DELIMITER $$

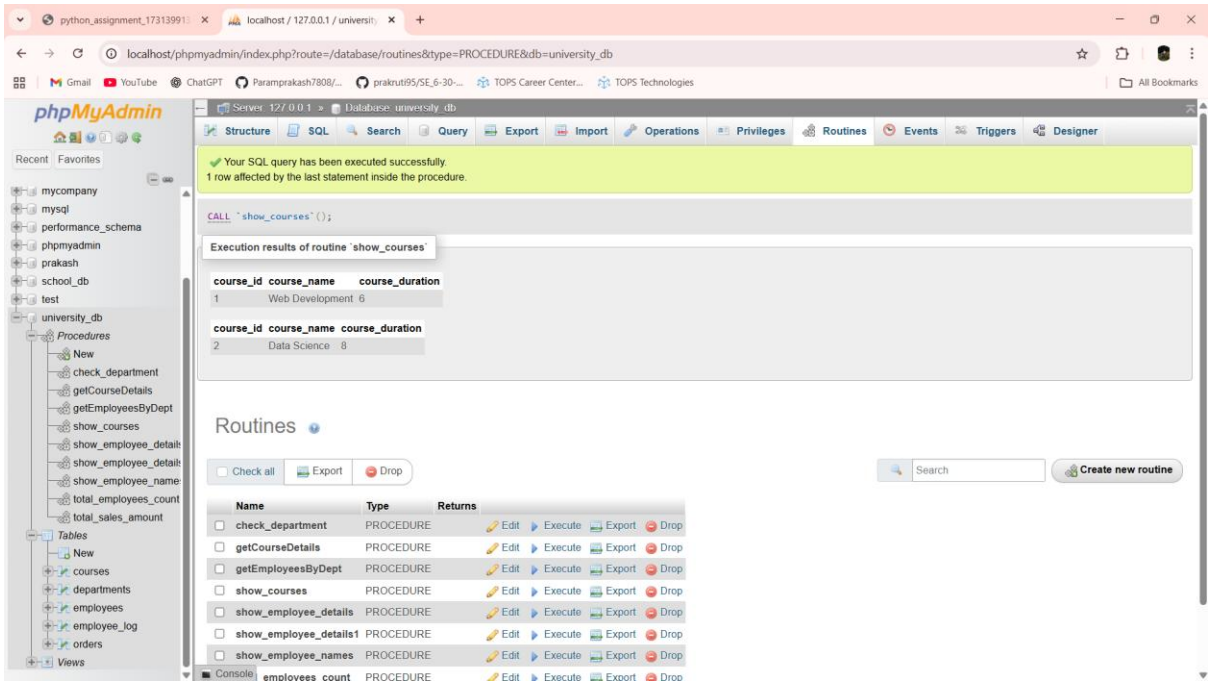
```
CREATE PROCEDURE show_employee_details1()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE v_id INT;
  DECLARE v_name VARCHAR(100);
  DECLARE v_dept VARCHAR(50);
  DECLARE emp_cursor CURSOR FOR
    SELECT employee_id, employee_name, department FROM employees;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
  OPEN emp_cursor;
  read_loop: LOOP
    FETCH emp_cursor INTO v_id, v_name, v_dept;
    IF done = 1 THEN
      LEAVE read_loop;
    END IF;
    SELECT v_id AS employee_id, v_name AS employee_name,
        v_dept AS department;
  END LOOP;
  CLOSE emp_cursor;
END $$
DELIMITER ;
CALL show_employee_details();
```


localhost/phpmyadmin/index.php?route=/database/routines&type=PROCEDURE&db=university_db
phpMyAdmin
Server: 127.0.0.1 » Database: university_db
Structure SQL Search Query Export Import Operations Privileges Routines Events Triggers Designer
Your SQL query has been executed successfully.
1 row affected by the last statement inside the procedure.
CALL `show_employee_details1`();
Execution results of routine `show_employee_details1`
employee_id employee_name department
1 Ajay HR
employee_id employee_name department
2 Mayur Finance
employee_id employee_name department
3 Sahil IT
employee_id employee_name department
4 Nik HR
employee_id employee_name department
5 Prakash Marketing
Routines
Check all Export Drop
Search
Create new routine
Name Type Returns
PROCEDURE Edit Execute Export Drop
Console

Lab 18.2: Create a cursor to retrieve all courses and display them one by one.

Ans:- 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;
  read_loop: LOOP
    FETCH course_cursor INTO v_id, v_name, v_duration;
    IF done = 1 THEN
      LEAVE read_loop;
    END IF;
    SELECT v_id AS course_id, v_name AS course_name, v_duration AS
course_duration;
  END LOOP;
  CLOSE course_cursor;
END $$
DELIMITER ;
CALL show_courses();
```


phpMyAdmin
Recent
Favorites
mycompany
mysql
performance_schema
phpmyadmin
prakash
school_db
test
university_db
Procedures
New
check_department
getCourseDetails
getEmployeesByDept
show_courses
Tables
courses
departments
employees
employee_log
orders
Views
Server: 127.0.0.1 » Database: university_db
Structure
SQL
Search
Query
Export
Import
Operations
Privileges
Routines
Events
Triggers
Designer
Your SQL query has been executed successfully.
1 row affected by the last statement inside the procedure.
CALL `show_courses`();
Execution results of routine `show_courses`
course_id course_name course_duration
1 Web Development 6
course_id course_name course_duration
2 Data Science 8
Routines
Check all
Export
Drop
Search
Create new routine
Name
Type
Returns
check_department PROCEDURE
getCourseDetails PROCEDURE
getEmployeesByDept PROCEDURE
show_courses PROCEDURE
show_employee_details PROCEDURE
show_employee_details1 PROCEDURE
show_employee_names PROCEDURE
Edit
Execute
Export
Drop
Console

Lab 19.1: Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.

Ans:- START TRANSACTION;

INSERT INTO employees (employee_id, employee_name, department)

VALUES (1, 'Alice', 'HR');

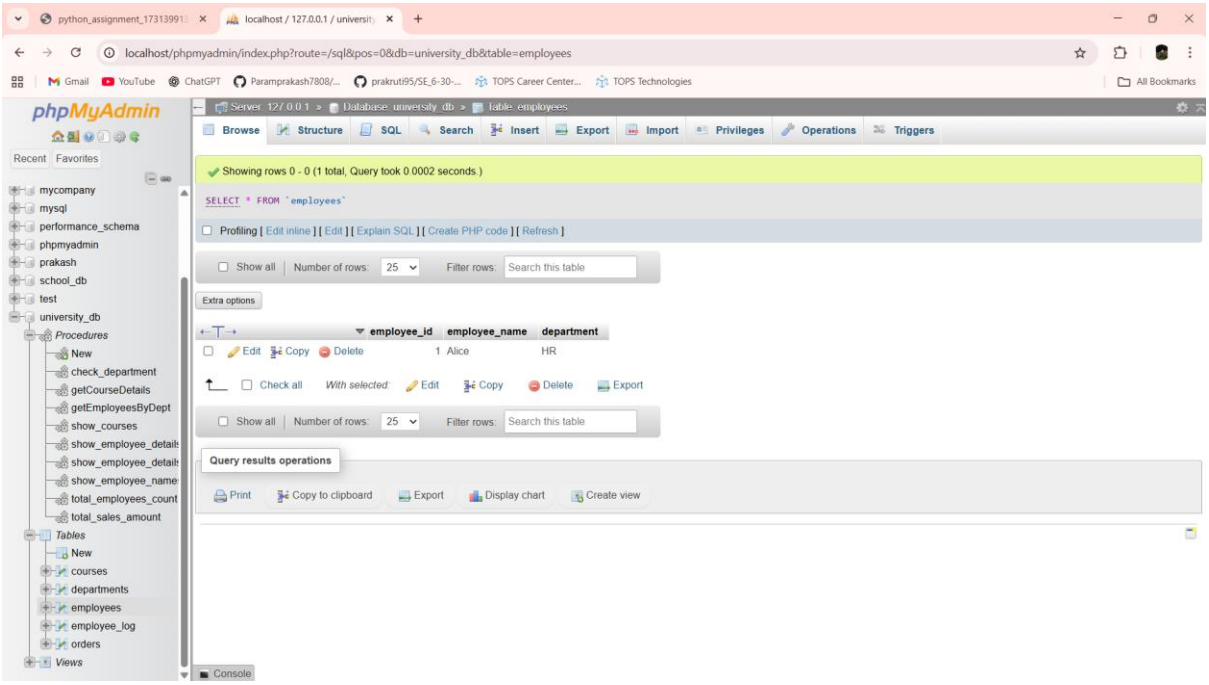
SAVEPOINT sp1;

INSERT INTO employees (employee_id, employee_name, department)

VALUES (2, 'Bob', 'IT');

ROLLBACK TO SAVEPOINT sp1;

COMMIT;



Lab 19.2: Commit part of a transaction after using a savepoint and then rollback the remaining changes.

Ans:- START TRANSACTION;

INSERT INTO employees (employee_id, employee_name, department)

VALUES (3, 'Charlie', 'Finance');

SAVEPOINT sp2;

INSERT INTO employees (employee_id, employee_name, department)

VALUES (4, 'David', 'Marketing');

COMMIT;

START TRANSACTION;

INSERT INTO employees (employee_id, employee_name, department)

VALUES (5, 'Eve', 'Sales');

ROLLBACK;