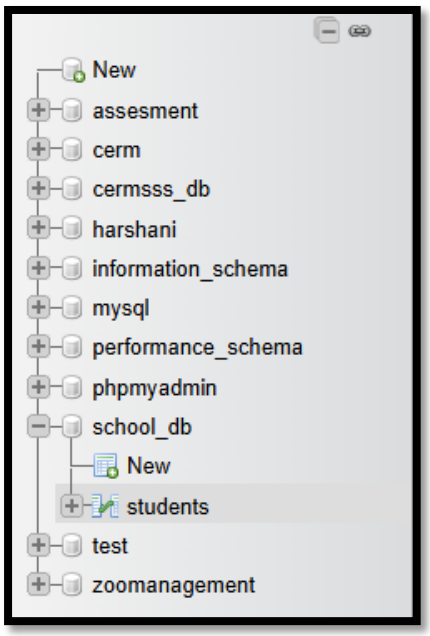
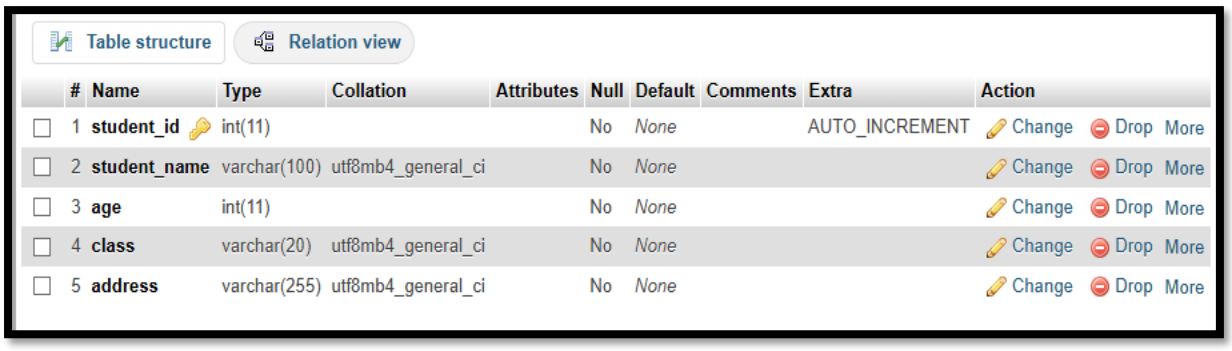
LAB EXERCISES:

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;



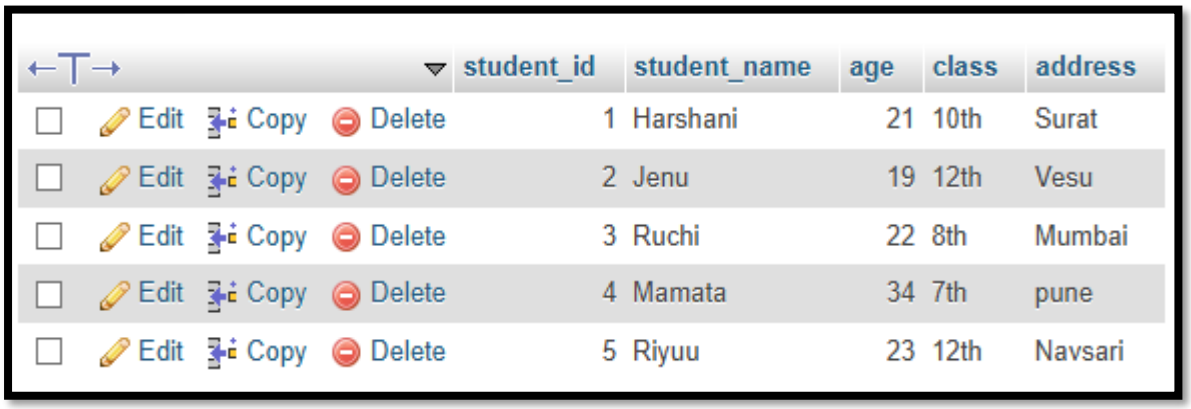
- CREATE TABLE students (student_id INT AUTO_INCREMENT PRIMARY KEY,
  student_name VARCHAR(100),
  age INT,
  class VARCHAR (50),
  address VARCHAR (255)
  );



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 ('Harshani', 21, '8th ', 'Surat, Gujarat'), ('Jenu', 19, '12th ', 'Vesu'),('Ruchi', 22, '8th ', 'Mumbai'),('Mamata', 34, '7th ', 'Pune'),('Riyu', 23, '12th ', 'Navsari');

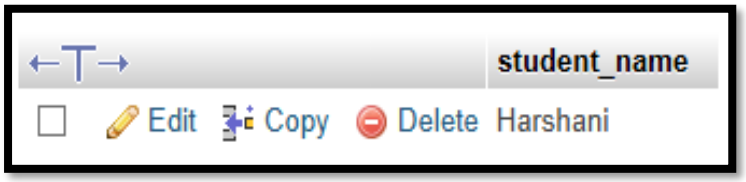
- SELECT * FROM students;



	student_id	student_name	age	class	address
Edit Copy Delete	1	Harshani	21	10th	Surat
Edit Copy Delete	2	Jenu	19	12th	Vesu
Edit Copy Delete	3	Ruchi	22	8th	Mumbai
Edit Copy Delete	4	Mamata	34	7th	pune
Edit Copy Delete	5	Riyuu	23	12th	Navsari

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

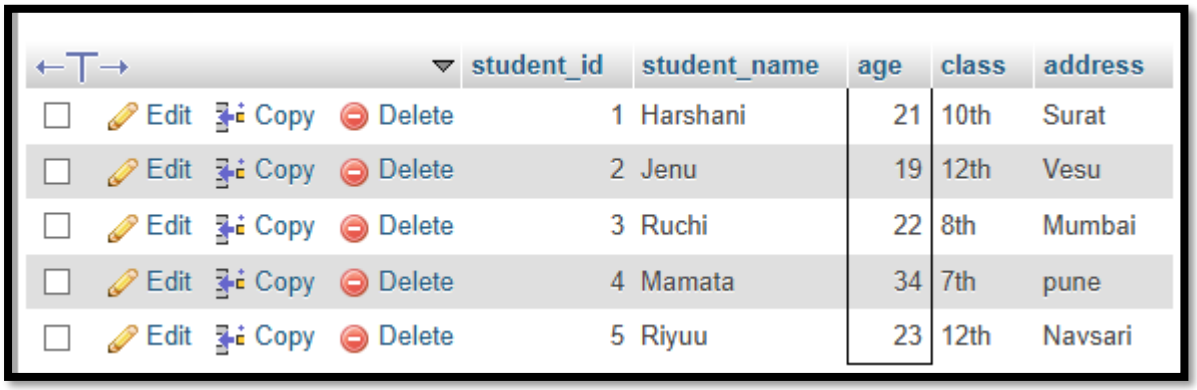
- SELECT student_name FROM `students` WHERE age=21;



	student_name
Edit Copy Delete	Harshani

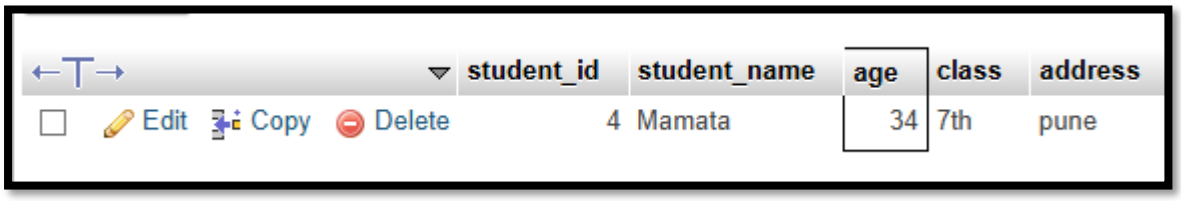
Lab 2: Write SQL queries to retrieve all students whose age is greater than 10.

- SELECT * FROM `students` WHERE age >10;



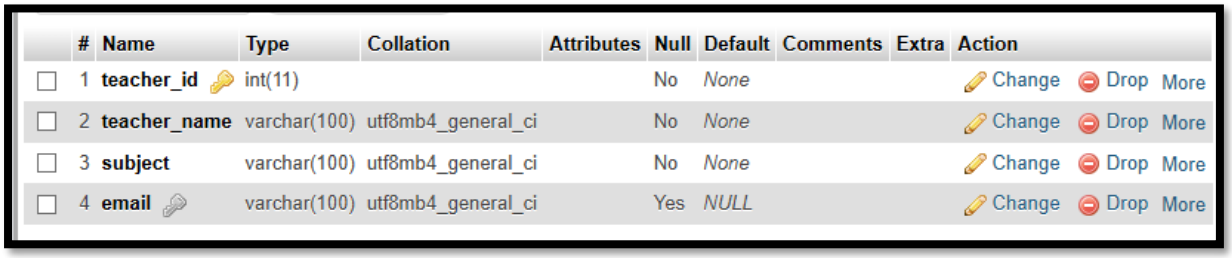
	student_id	student_name	age	class	address
Edit Copy Delete	1	Harshani	21	10th	Surat
Edit Copy Delete	2	Jenu	19	12th	Vesu
Edit Copy Delete	3	Ruchi	22	8th	Mumbai
Edit Copy Delete	4	Mamata	34	7th	pune
Edit Copy Delete	5	Riyuu	23	12th	Navsari

➢ SELECT * FROM `students` WHERE age >23;



	student_id	student_name	age	class	address
Edit Copy Delete	4	Mamata	34	7th	pune

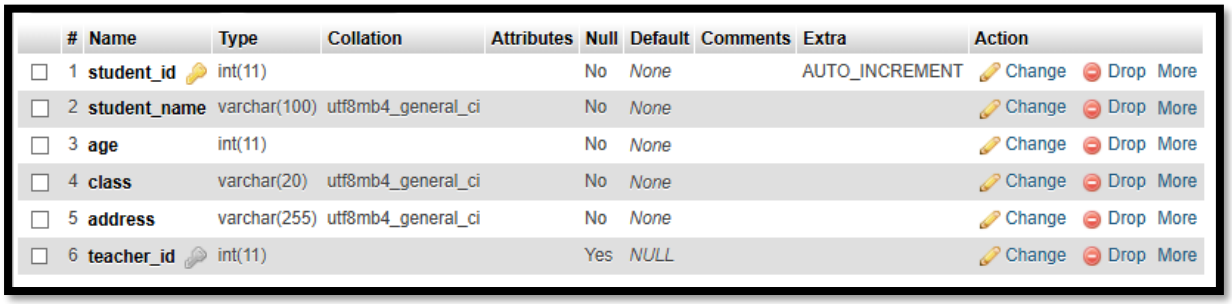
Lab 1: Create a table teachers with the following columns: teacher_id (Primary Key), teacher_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	teacher_id	int(11)			No	None			Change Drop More
2	teacher_name	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
3	subject	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
4	email	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More

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_teacher FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	student_id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	student_name	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
3	age	int(11)			No	None			Change Drop More
4	class	varchar(20)	utf8mb4_general_ci		No	None			Change Drop More
5	address	varchar(255)	utf8mb4_general_ci		No	None			Change Drop More
6	teacher_id	int(11)			Yes	NULL			Change Drop More

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

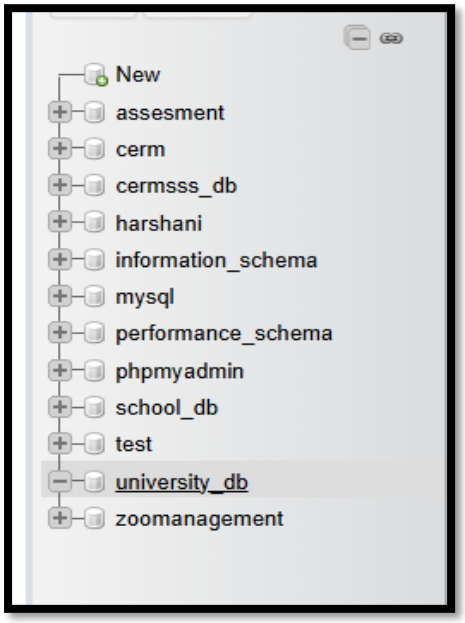
Lab 1: Create a table courses with columns: course_id, course_name, and course_credits. Set the course_id as the primary key.

➢ CREATE TABLE courses (course_id INT PRIMARY KEY,course_name VARCHAR (100), course_credits INT);

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	course_id	int(11)			No	None			Change Drop More
2	course_name	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	course_credits	int(11)			Yes	NULL			Change Drop More

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

```
CREATE DATABASE university_db;
```



5. ALTER Command

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

```
ALTER TABLE `courses` ADD course_duration varchar(50);
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	course_id	int(11)			No	None			Change Drop More
2	course_name	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	course_credits	int(11)			Yes	NULL			Change Drop More
4	course_duration	varchar(50)	utf8mb4_general_ci		Yes	NULL			Change Drop More

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

```
ALTER TABLE courses DROP COLUMN course_credits;
```

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	course_id	int(11)			No	None			Change Drop More
2	course_name	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	course_duration	varchar(50)	utf8mb4_general_ci		Yes	NULL			Change Drop More

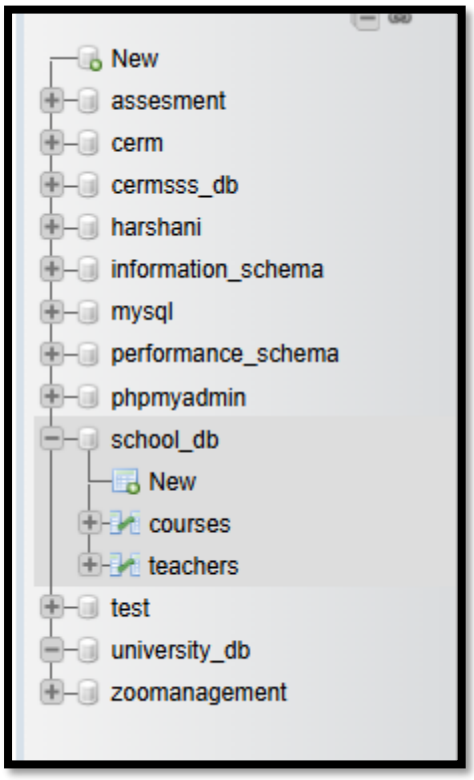
6. DROP Command:

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

- DROP TABLE `t teachers`;
- DROP TABLE `teachers`;
- SET FOREIGN_KEY_CHECKS = 0;
- DROP TABLE school_db.teachers;
- SET FOREIGN_KEY_CHECKS = 1;

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

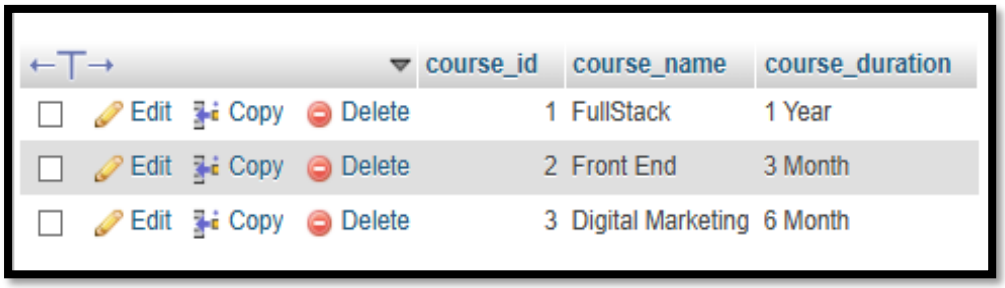
- DROP TABLE students;



7. Data Manipulation Language (DML)

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

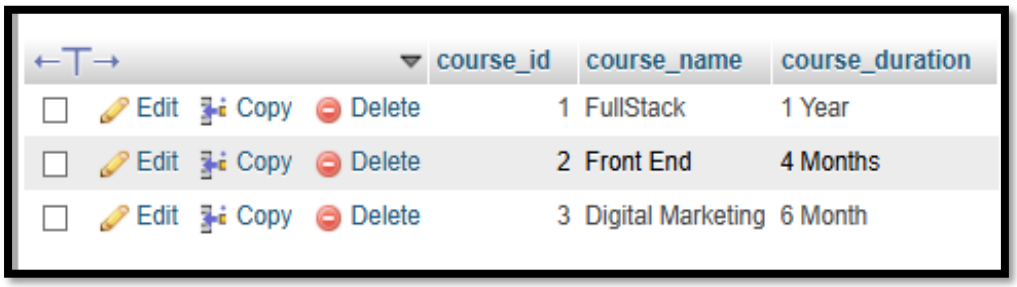
- INSERT into `courses` (course_id,course_name, course_duration)VALUES(1,"Full Stack","1 Year");
- INSERT into `courses` (course_id,course_name,course_duration)VALUES(2,"Front End","3 Month");
- INSERT into `courses` (course_id,course_name,course_duration)VALUES(3,"Digital Marketing","6 Month");



	course_id	course_name	course_duration
Edit Copy Delete	1	FullStack	1 Year
Edit Copy Delete	2	Front End	3 Month
Edit Copy Delete	3	Digital Marketing	6 Month

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

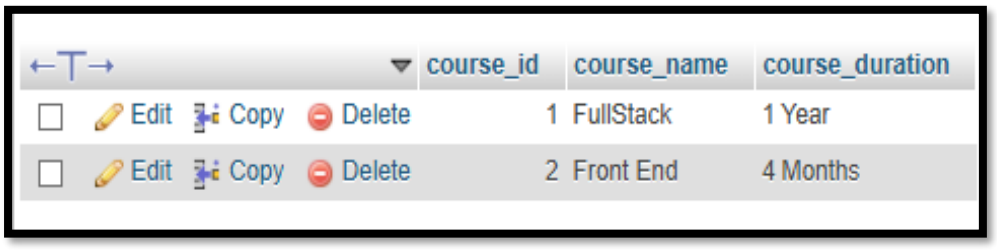
- SET course_duration = '4 Months' WHERE course_name = 'Front End';



	course_id	course_name	course_duration
Edit Copy Delete	1	FullStack	1 Year
Edit Copy Delete	2	Front End	4 Months
Edit Copy Delete	3	Digital Marketing	6 Month

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

- DELETE FROM`courses` WHERE course_id=3;

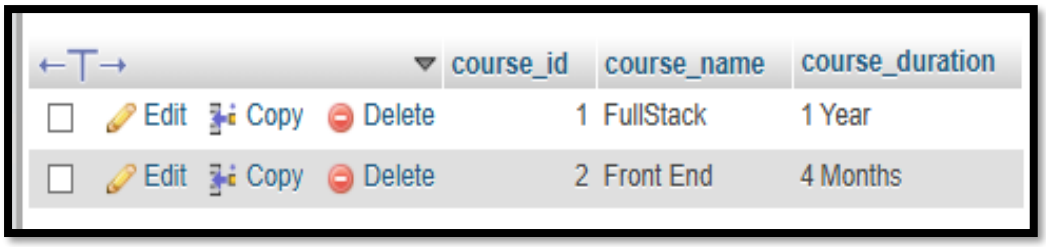


	course_id	course_name	course_duration
Edit Copy Delete	1	FullStack	1 Year
Edit Copy Delete	2	Front End	4 Months

8. Data Query Language (DQL)

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

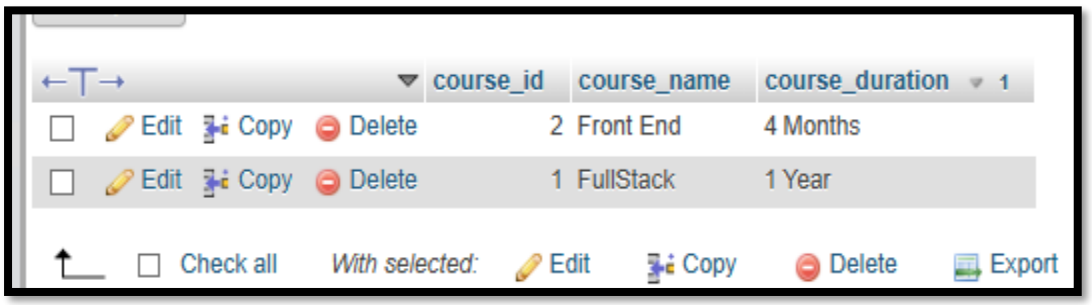
- SELECT * FROM `courses`;



	course_id	course_name	course_duration
Edit Copy Delete	1	FullStack	1 Year
Edit Copy Delete	2	Front End	4 Months

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

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



	course_id	course_name	course_duration
Edit Copy Delete	2	Front End	4 Months
Edit Copy Delete	1	FullStack	1 Year

Check all With selected: Edit Copy Delete Export

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

- SELECT * FROM `courses` LIMIT 1;

	course_id	course_name	course_duration
Edit Copy Delete	1	FullStack	1 Year

9. Data Control Language (DCL)

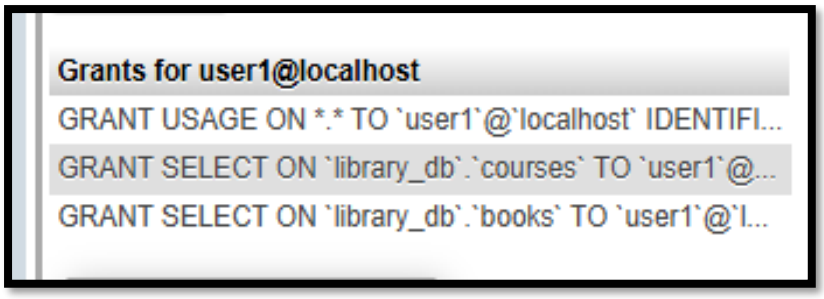
Lab 1: Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.

- CREATE OR REPLACE USER 'user1'@'localhost' IDENTIFIED BY 'password1';
- CREATE OR REPLACE USER 'user2'@'localhost' IDENTIFIED BY 'password2';

- GRANT SELECT ON library_db.books TO 'user1'@'localhost';
- FLUSH PRIVILEGES;

- CREATE TABLE library_db.courses (id INT AUTO_INCREMENT PRIMARY KEY, course_name VARCHAR(255),instructor VARCHAR(255));

- GRANT SELECT ON library_db.courses TO 'user1'@'localhost';
- FLUSH PRIVILEGES;



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

- REVOKE INSERT ON library_db.books FROM 'user1'@'localhost';

- GRANT INSERT ON library_db.books TO 'user2'@'localhost';
- FLUSH PRIVILEGES;

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

- REVOKE INSERT ON library_db. books FROM 'user1'@'localhost';
- GRANT INSERT ON library_db.books TO 'user2'@'localhost';
- FLUSH PRIVILEGES;

- SHOW GRANTS FOR 'user1'@'localhost';
- SHOW GRANTS FOR 'user2'@'localhost';

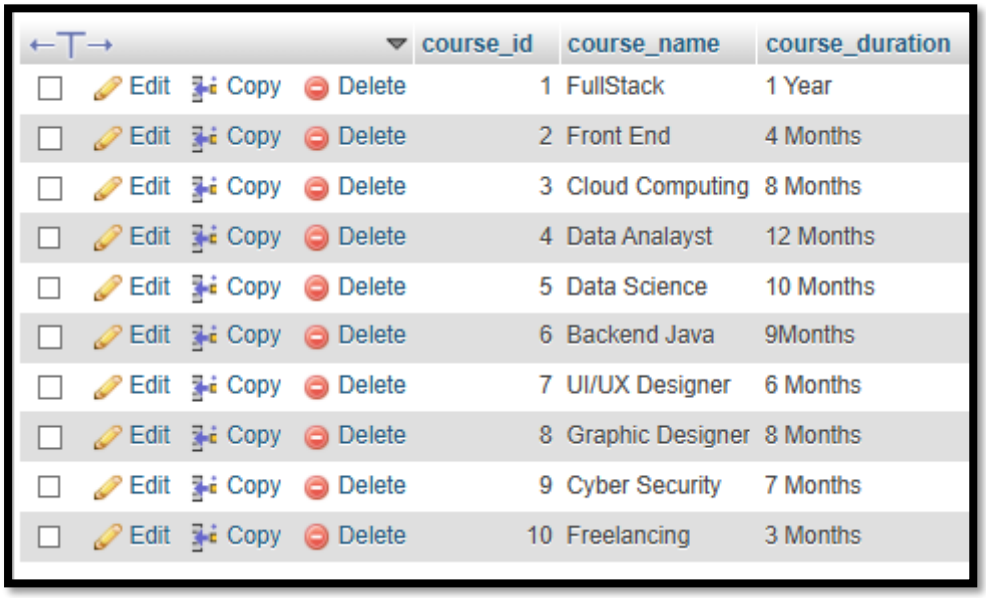
10. Transaction Control Language (TCL)

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

- INSERT INTO `courses` (course_id,course_name,course_duration)VALUES(3,"Cloud Computing","8 Months");
- INSERT INTO `courses`(course_id,course_name,course_duration)VALUES(4,"Data Analayst","12 Months");
- INSERT INTO `courses`(course_id,course_name,course_duration)VALUES(5,"Data Science","10 Months");
- INSERT INTO `courses`(course_id,course_name,course_duration)VALUES(6,"Backend Java","9Months");
- INSERT INTO `courses`(course_id,course_name,course_duration)VALUES(7,"UI/UX Designer","6 Months");
- INSERT INTO `courses`(course_id,course_name,course_duration)VALUES(8,"Graphic Designer","8 Months");
- INSERT INTO `courses`(course_id,course_name,course_duration)VALUES(9,"Cyber Security","7 Months");
- INSERT INTO `courses`(course_id,course_name,course_duration)VALUES(10,"Freelancing","3 Months");

- COMMIT;

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



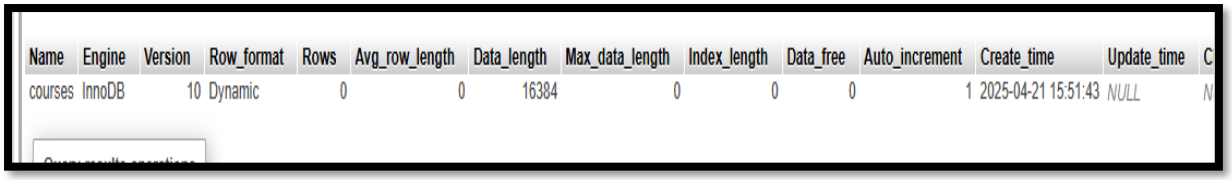
	course_id	course_name	course_duration
Edit Copy Delete	1	FullStack	1 Year
Edit Copy Delete	2	Front End	4 Months
Edit Copy Delete	3	Cloud Computing	8 Months
Edit Copy Delete	4	Data Analayst	12 Months
Edit Copy Delete	5	Data Science	10 Months
Edit Copy Delete	6	Backend Java	9Months
Edit Copy Delete	7	UI/UX Designer	6 Months
Edit Copy Delete	8	Graphic Designer	8 Months
Edit Copy Delete	9	Cyber Security	7 Months
Edit Copy Delete	10	Freelancing	3 Months

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

-- 1. Start the transaction
START TRANSACTION;

```
-- 2. Create a savepoint
SAVEPOINT before_update;

-- 3. Update the courses table
UPDATE courses
SET instructor = 'Dr. Sharma'
WHERE id = 1;

-- 4. Immediately check the updated value (optional)
SELECT * FROM courses WHERE id = 1;

-- 5. Now roll back to undo the change
ROLLBACK TO SAVEPOINT before_update;

-- 6. Check again to confirm rollback worked
SELECT * FROM courses WHERE id = 1;

-- 7. Commit transaction
COMMIT;
```

- SHOW TABLE STATUS LIKE 'courses';



Name	Engine	Version	Row_format	Rows	Avg_row_length	Data_length	Max_data_length	Index_length	Data_free	Auto_increment	Create_time	Update_time	C
courses	InnoDB	10	Dynamic	0	0	16384	0	0	0	1	2025-04-21 15:51:43	NULL	N

x_length	Data_free	Auto_increment	Create_time	Update_time	Check_time	Collation	Checksum	Create_options	Comment	Max_index_length	Temporary
0	0	1	2025-04-21 15:51:43	NULL	NULL	utf8mb4_general_ci	NULL			0	N

11. SQL Joins:

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

- CREATE TABLE departments (d_id INT PRIMARY KEY,d_name VARCHAR (50));

- CREATE TABLE employees (emp_id INT PRIMARY KEY,emp_name VARCHAR(100),d_id INT,FOREIGN KEY (d_id) REFERENCES departments(d_id));

//insert the data departments

- INSERT INTO `departments`(d_name) VALUES("COMPUTER");
- INSERT into `departments`(d_name) VALUES("IT");
- INSERT into `departments`(d_name) VALUES("AUTOMOBILES");
- INSERT INTO `departments`(d_name) VALUES("ELECTRICAL");
- INSERT INTO `departments`(d_name) VALUES ("MEDICAL SCIENCE");

	d_id	d_name
Edit Copy Delete	11	COMPUTER
Edit Copy Delete	12	IT
Edit Copy Delete	13	AUTOMOBILES
Edit Copy Delete	14	ELECTRICAL
Edit Copy Delete	15	MEDICAL SCIENCE

//inserts into employees

- INSERT INTO `employees`(emp_id,emp_name)VALUES(1,"Harshani Patil");
- INSERT INTO `employees`(emp_id,emp_name)VALUES(2,"Jenu Shah");
- INSERT INTO `employees`(emp_id, emp_name)VALUES(3,"Pratham Tiwari");
- INSERT INTO `employees`(emp_id,emp_name)VALUES(4,"Piyu Patel");
- INSERT INTO `employees`(emp_id, emp_name) VALUES(5,"Ruchi Desai");

←T→				emp_id	emp_name	d_id
☐	Edit	Copy	Delete	1	Harshani Patil	NULL
☐	Edit	Copy	Delete	2	Jenu Shah	NULL
☐	Edit	Copy	Delete	3	Pratham Tiwari	NULL
☐	Edit	Copy	Delete	4	Piyu Patel	NULL
☐	Edit	Copy	Delete	5	Ruchi Desai	NULL

//perform the inner join

- SELECT employees.emp_id, employees.emp_name,departments.d_name AS department FROM employees INNER JOIN departments ON employees.d_id = departments.d_id;

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

- SELECTdepartments.d_id,departments.d_name,employees.emp_id,employees.emp_name FROM departments LEFT JOIN employees ON departments.d_id = employees. d_id;

d_id	d_name	emp_id	emp_name
11	COMPUTER	NULL	NULL
12	IT	NULL	NULL
13	AUTOMOBILES	NULL	NULL
14	ELECTRICAL	NULL	NULL
15	MEDICAL SCIENCE	NULL	NULL

12. SQL Group By:

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

- SELECT
  departments.d_name AS department,
  COUNT(employees.emp_id) AS employee_count
  FROM
  departments
  LEFT JOIN
  employees ON departments.d_id = employees.d_id
  GROUP BY
  departments.d_name;

department	employee_count
AUTOMOBILES	0
COMPUTER	0
ELECTRICAL	0
IT	0
MEDICAL SCIENCE	0

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

- ALTER TABLE employees ADD salary INT;

- UPDATE employees SET salary = 50000 WHERE emp_id = 1;
- UPDATE employees SET salary = 45000 WHERE emp_id = 2;
- UPDATE employees SET salary = 55000 WHERE emp_id = 3;
- UPDATE employees SET salary = 60000 WHERE emp_id = 4;
- UPDATE employees SET salary = 47000 WHERE emp_id = 5;

- UPDATE employees SET d_id = 11 WHERE emp_id = 1;
- UPDATE employees SET d_id = 12 WHERE emp_id = 2;
- UPDATE employees SET d_id = 11 WHERE emp_id = 3;
- UPDATE employees SET d_id = 13 WHERE emp_id = 4;
- UPDATE employees SET d_id = 11 WHERE emp_id = 5;

- SELECT
  departments.d_name AS department,
  AVG(employees.salary) AS average_salary
  FROM
  departments
  LEFT JOIN
  employees ON departments.d_id = employees.d_id
  GROUP BY
  departments.d_name;

department	average_salary
AUTOMOBILES	60000.0000
COMPUTER	50666.6667
ELECTRICAL	*NULL*
IT	45000.0000
MEDICAL SCIENCE	*NULL*

13. SQL Stored Procedure:

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

- DELIMITER $$
  CREATE PROCEDURE getEmployeesByDepartment(IN dept_id INT)
  BEGIN
  SELECT * FROM employees
  WHERE d_id = dept_id;
  END$$
  DELIMITER ;

- CALL getEmployeesByDepartment(11);

emp_id	emp_name	d_id	salary
1	Harshani Patil	11	50000
3	Pratham Tiwari	11	55000
5	Ruchi Desai	11	47000

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

- DELIMITER $$
  CREATE PROCEDURE GetCourseDetails (IN cid INT)
  BEGIN
  SELECT * FROM courses WHERE course_id = cid;
  END$$
  DELIMITER;

- CALL GetCourseDetails(3);

course_id	course_name	course_duration
3	Cloud Computing	8 Months

14. SQL View

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

- SELECT e.*, d.d_name AS department_nameFROM employees e JOIN departments d ON e.d_id = d.d_id;

emp_id	emp_name	d_id	salary	department_name
1	Harshani Patil	11	50000	COMPUTER
2	Jenu Shah	12	45000	IT
3	Pratham Tiwari	11	55000	COMPUTER
4	Piyu Patel	13	60000	AUTOMOBILES
5	Ruchi Desai	11	47000	COMPUTER

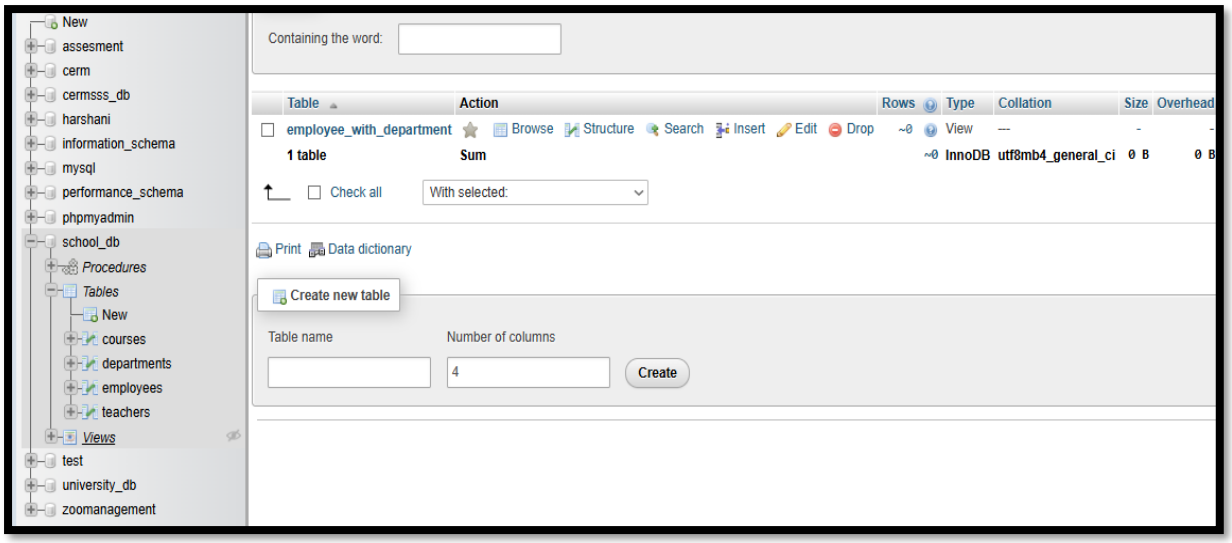
- CREATE VIEW employee_with_department AS SELECT e.*, d.d_name AS department_name FROM employees e JOIN departments d ON e.d_id = d.d_id;

←T→				emp_id	emp_name	d_id	salary	department_name
☐	Edit	Copy	Delete	1	Harshani Patil	11	50000	COMPUTER
☐	Edit	Copy	Delete	2	Jenu Shah	12	45000	IT
☐	Edit	Copy	Delete	3	Pratham Tiwari	11	55000	COMPUTER
☐	Edit	Copy	Delete	4	Piyu Patel	13	60000	AUTOMOBILES
☐	Edit	Copy	Delete	5	Ruchi Desai	11	47000	COMPUTER

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

- DROP VIEW IF EXISTS employee_with_department;

- CREATE VIEW employee_with_department AS SELECT e.*, d.d_name AS department_name FROM employees e JOIN departments d ON e.d_id = d.d_id WHERE e.salary >= 50000;



15. SQL Triggers

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

Create a log table:

- CREATE TABLE employee_log (
  log_id INT AUTO_INCREMENT PRIMARY KEY,
  emp_id INT,
  action VARCHAR(50),log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP);

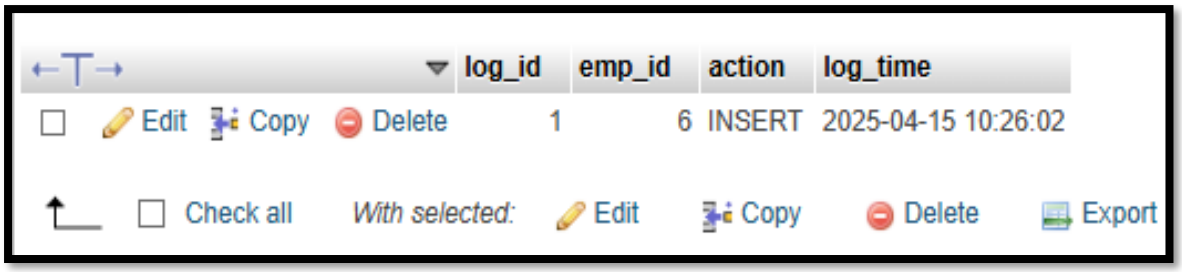
Create the trigger:

- 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 ;

- INSERT INTO employees (emp_id, emp_name, d_id, salary)VALUES (6, 'Test User', 14, 58000);

- SELECT * FROM employee_log;



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

- ALTER TABLE employees ADD COLUMN last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;

Step 2: Create the Trigger:

- DELIMITER $$

  CREATE TRIGGER before_employee_update
  BEFORE UPDATE ON employees
  FOR EACH ROW
  BEGIN
  SET NEW.last_modified = CURRENT_TIMESTAMP;
  END $$

  DELIMITER ;

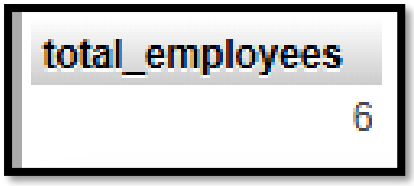
- UPDATE employees SET salary = 61000 WHERE emp_id = 1;

- SELECT * FROM employees;

16. Introduction to PL/SQL

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

- SELECT COUNT(*) AS total_employees FROM employees;



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

- CREATE TABLE orders (order_id INT AUTO_INCREMENT PRIMARY KEY, customer_name VARCHAR(100),amount DECIMAL(10,2));

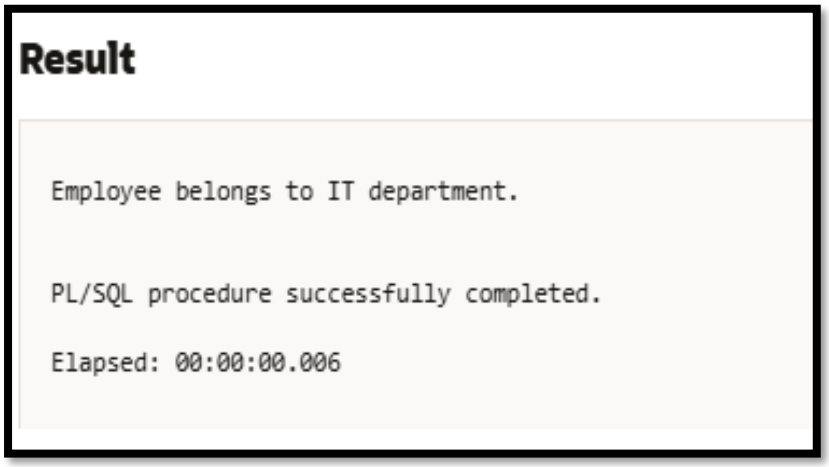
- INSERT INTO orders (customer_name, amount) VALUES('John Doe', 500.00), ('Jane Smith', 1200.50),('Alice', 850.75);

- SELECT SUM(amount) AS total_sales FROM orders;

17. PL/SQL Control Structures

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

- DECLARE
  v_dept_id NUMBER:= 20; -- Example: IT Department
  BEGIN
  IF v_dept_id = 10 THEN
  DBMS_OUTPUT.PUT_LINE('Employee belongs to HR department.');
  ELSIF v_dept_id = 20 THEN
  DBMS_OUTPUT.PUT_LINE('Employee belongs to IT department.');
  ELSIF v_dept_id = 30 THEN
  DBMS_OUTPUT.PUT_LINE('Employee belongs to Finance department.');
  ELSE
  DBMS_OUTPUT.PUT_LINE('Employee belongs to an unknown department.');
  END IF;
END;

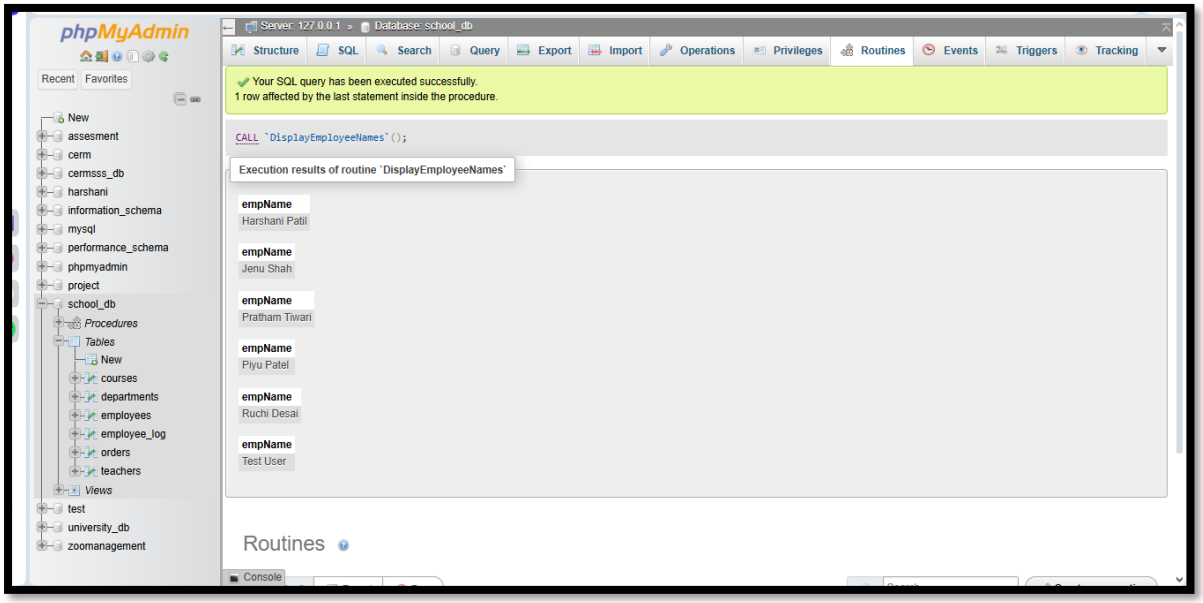


 Lab 2: Use a FOR LOOP to iterate through employee records and display their names

- DELIMITER $$

```
CREATE PROCEDURE DisplayEmployeeNames()
BEGIN
DECLARE done INT DEFAULT FALSE;
DECLARE empName VARCHAR(100);

Declare cursor
DECLARE emp_cursor CURSOR FOR
SELECT emp_name FROM employees;

Declare continue handler
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN emp_cursor;
read_loop: LOOP
FETCH emp_cursor INTO empName;
IF done THEN
LEAVE read_loop;
END IF;
SELECT empName;
END LOOP;

CLOSE emp_cursor;
END$$

DELIMITER;
```

- CALL `DisplayEmployeeNames`();



18. SQL Cursors:

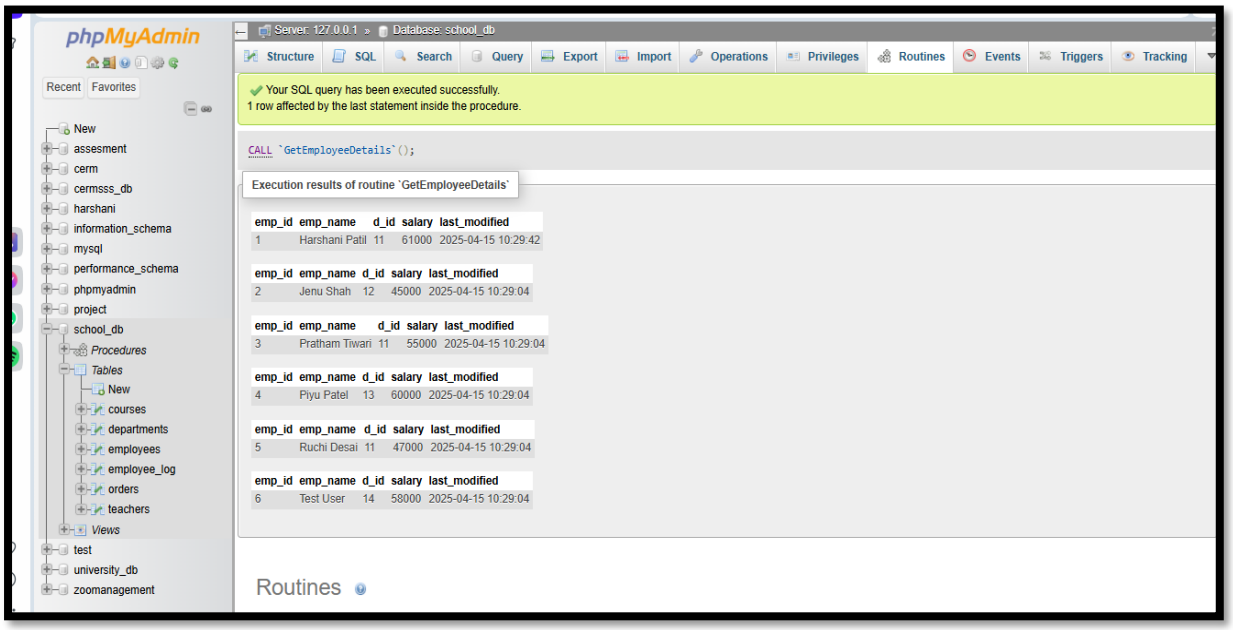
Lab 1: Write a PL/SQL block using an explicit cursor to retrieve and display employee details.

- DELIMITER $$

```
    CREATE PROCEDURE GetEmployeeDetails()
    BEGIN
Declare variables
    DECLARE done INT DEFAULT FALSE;
    DECLARE v_emp_id INT;
    DECLARE v_emp_name VARCHAR(100);
    DECLARE v_d_id INT;
    DECLARE v_salary INT;
    DECLARE v_last_modified DATETIME;

Declare cursor
DECLARE emp_cursor CURSOR FOR
SELECT emp_id, emp_name, d_id, salary, last_modified FROM employees;

Declare continue handler
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
Open cursor
OPEN emp_cursor;

read_loop: LOOP
FETCH emp_cursor INTO v_emp_id, v_emp_name, v_d_id, v_salary, v_last_modified;

IF done THEN
LEAVE read_loop;
END IF;
Display result for each employee
SELECT v_emp_id AS emp_id,
v_emp_name AS emp_name,
v_d_id AS d_id,
v_salary AS salary,
v_last_modified AS last_modified;
END LOOP;
Close cursor
    CLOSE emp_cursor;
    END$$
```

- DELIMITER ;

- CALL `GetEmployeeDetails`();

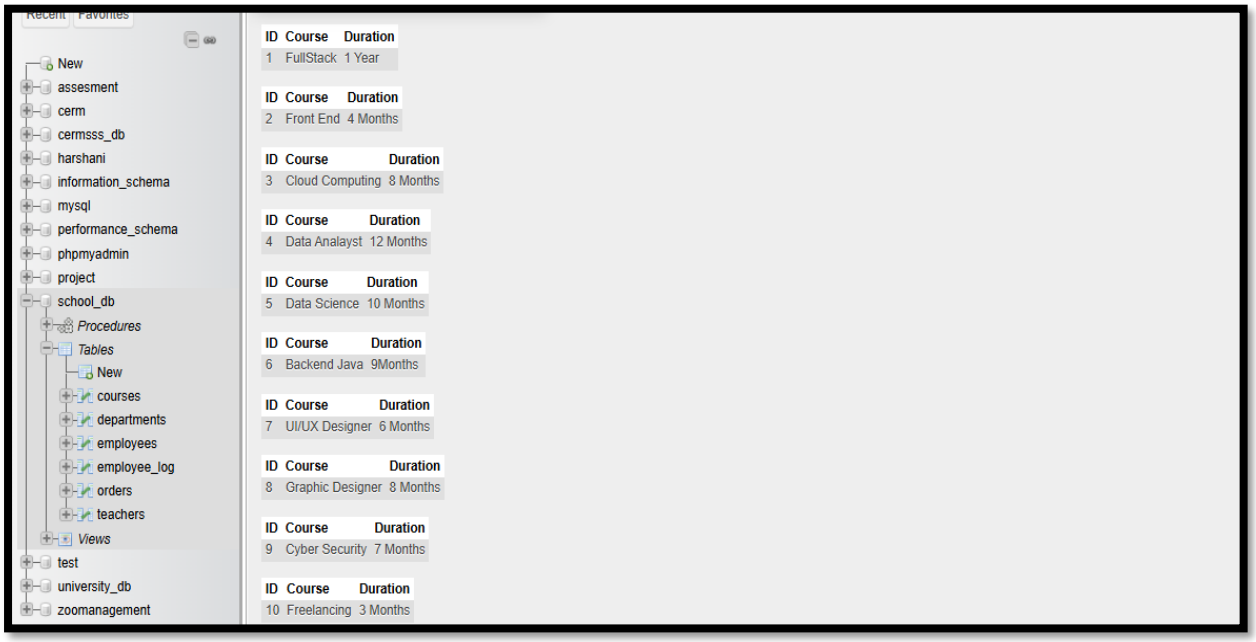


Lab 2: Create a cursor to retrieve all courses and display them one by one

```
DELIMITER //

CREATE PROCEDURE DisplayCourses()
BEGIN
   DECLARE c_id INT;
   DECLARE c_name VARCHAR(100);
   DECLARE c_duration VARCHAR(50);
   DECLARE done INT DEFAULT 0;

   -- Cursor declaration
   DECLARE course_cursor CURSOR FOR
      SELECT course_id, course_name, course_duration FROM courses;

   -- Handler for end of data
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

   -- Open cursor
   OPEN course_cursor;

   read_loop: LOOP
      FETCH course_cursor INTO c_id, c_name, c_duration;

      IF done THEN
```

```
        LEAVE read_loop;
    END IF;

    -- Display the course
    SELECT c_id AS ID, c_name AS Course, c_duration AS Duration;

  END LOOP;

  -- Close cursor
  CLOSE course_cursor;
END //

DELIMITER ;
```

- `CALL `DisplayCourses`();`



19. Rollback and Commit Savepoint:

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

```
START TRANSACTION;

-- Step 1: Insert some initial employees
INSERT INTO employees (emp_id, emp_name, d_id, salary, last_modified)
VALUES (101, 'Neha Sharma', 11, 50000, NOW());

INSERT INTO employees (emp_id, emp_name, d_id, salary, last_modified)
```

VALUES (102, 'Amit Patel', 12, 48000, NOW());

-- Step 2: Create savepoint
SAVEPOINT before_extra_insert;

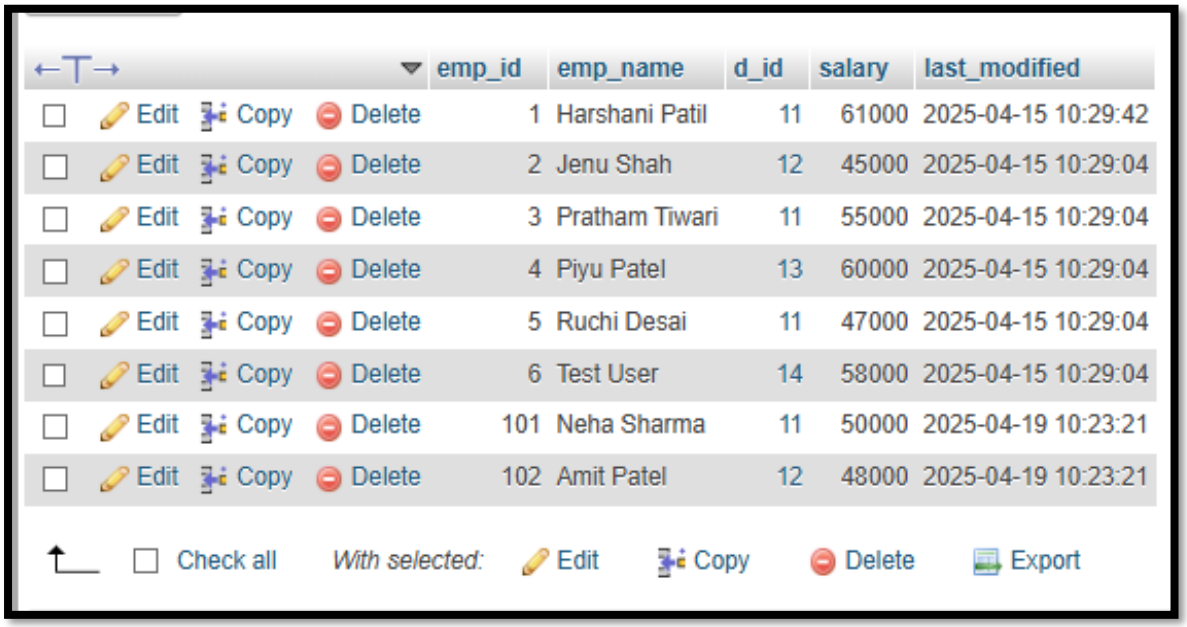
-- Step 3: Insert more employees
INSERT INTO employees (emp_id, emp_name, d_id, salary, last_modified)
VALUES (103, 'Priya Singh', 13, 55000, NOW ());

INSERT INTO employees (emp_id, emp_name, d_id, salary, last_modified)
VALUES (104, 'Rahul Mehta', 14, 51000, NOW());

-- Step 4: Rollback to savepoint (undo Priya & Rahul)
ROLLBACK TO SAVEPOINT before_extra_insert;

-- Step 5: Commit the remaining transaction
COMMIT;

➢ SELECT * FROM employees;



	emp_id	emp_name	d_id	salary	last_modified
Edit Copy Delete	1	Harshani Patil	11	61000	2025-04-15 10:29:42
Edit Copy Delete	2	Jenu Shah	12	45000	2025-04-15 10:29:04
Edit Copy Delete	3	Pratham Tiwari	11	55000	2025-04-15 10:29:04
Edit Copy Delete	4	Piyu Patel	13	60000	2025-04-15 10:29:04
Edit Copy Delete	5	Ruchi Desai	11	47000	2025-04-15 10:29:04
Edit Copy Delete	6	Test User	14	58000	2025-04-15 10:29:04
Edit Copy Delete	101	Neha Sharma	11	50000	2025-04-19 10:23:21
Edit Copy Delete	102	Amit Patel	12	48000	2025-04-19 10:23:21

Check all *With selected:* Edit Copy Delete Export

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

-- Step 1: Some inserts/updates/deletes
UPDATE employees SET salary = 65000 WHERE emp_id = 1;

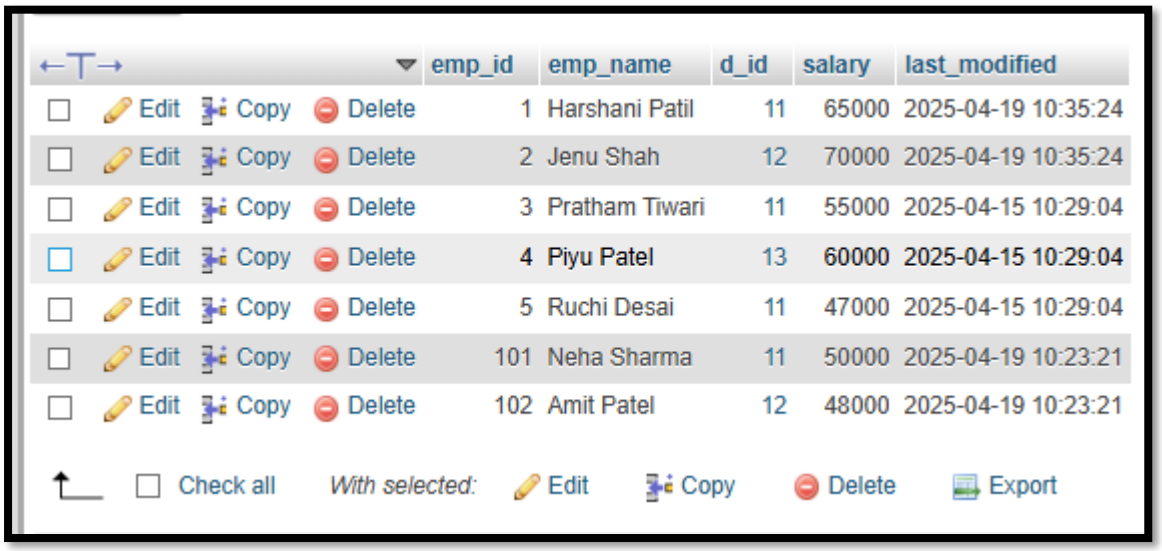
-- Create a savepoint after the above query
SAVEPOINT part_done;

-- Step 2: More operations
UPDATE employees SET salary = 70000 WHERE emp_id = 2;
DELETE FROM employees WHERE emp_id = 6;

-- Commit changes made before the savepoint
RELEASE SAVEPOINT part_done;
COMMIT;

-- Rollback the remaining changes
ROLLBACK;

- SELECT * FROM `employees`



	emp_id	emp_name	d_id	salary	last_modified
Edit Copy Delete	1	Harshani Patil	11	65000	2025-04-19 10:35:24
Edit Copy Delete	2	Jenu Shah	12	70000	2025-04-19 10:35:24
Edit Copy Delete	3	Pratham Tiwari	11	55000	2025-04-15 10:29:04
Edit Copy Delete	4	Piyu Patel	13	60000	2025-04-15 10:29:04
Edit Copy Delete	5	Ruchi Desai	11	47000	2025-04-15 10:29:04
Edit Copy Delete	101	Neha Sharma	11	50000	2025-04-19 10:23:21
Edit Copy Delete	102	Amit Patel	12	48000	2025-04-19 10:23:21

Check all With selected: Edit Copy Delete Export