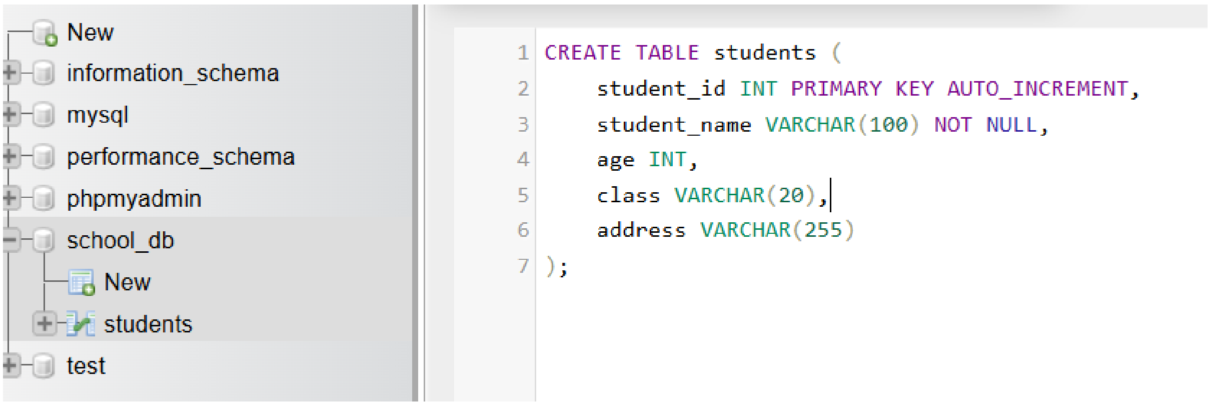
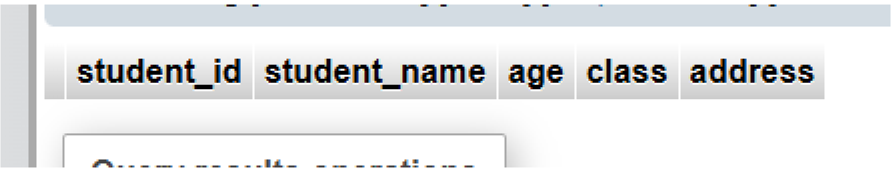
Module-5) Se - Introduction To Dbms (LAB)

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.

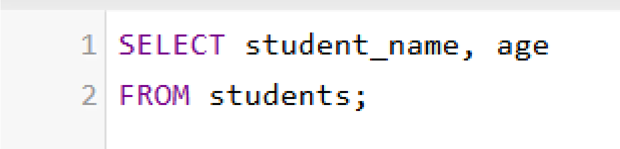


```
CREATE TABLE students (
    student_id INT PRIMARY KEY AUTO_INCREMENT,
    student_name VARCHAR(100) NOT NULL,
    age INT,
    class VARCHAR(20),
    address VARCHAR(255)
);
```

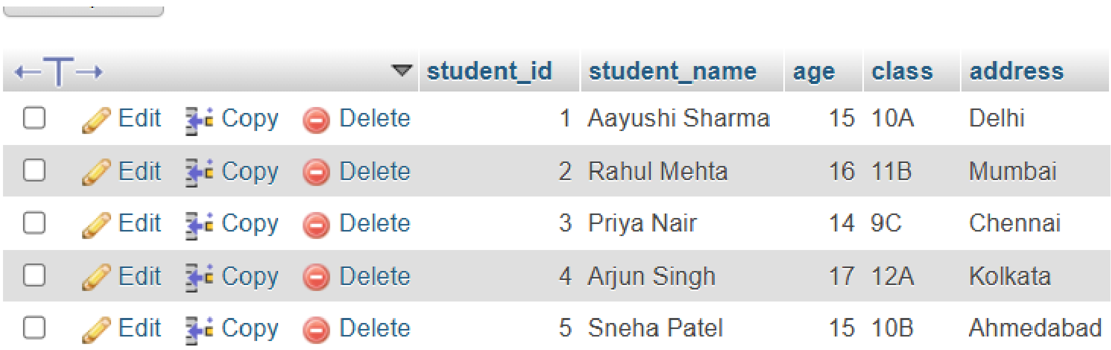


student_id	student_name	age	class	address

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



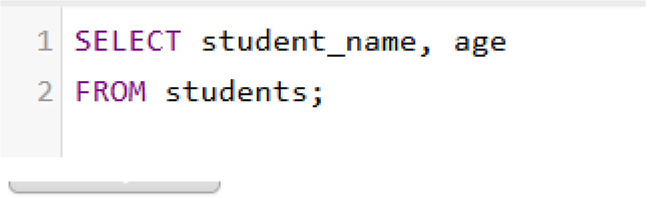
```
SELECT student_name, age
FROM students;
```



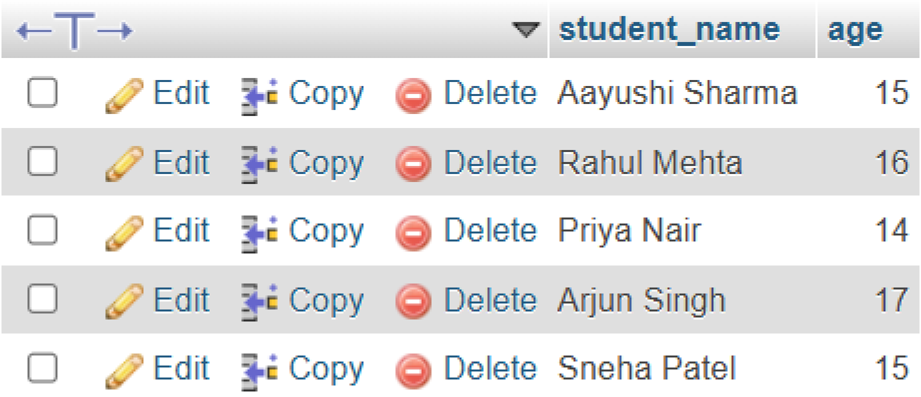
←T→			▼	student_id	student_name	age	class	address
□	Edit	Copy	Delete	1	Aayushi Sharma	15	10A	Delhi
□	Edit	Copy	Delete	2	Rahul Mehta	16	11B	Mumbai
□	Edit	Copy	Delete	3	Priya Nair	14	9C	Chennai
□	Edit	Copy	Delete	4	Arjun Singh	17	12A	Kolkata
□	Edit	Copy	Delete	5	Sneha Patel	15	10B	Ahmedabad

2. SQL Syntax

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

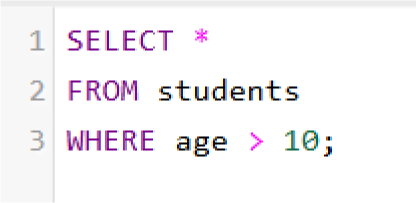


```sql
SELECT student_name, age
FROM students;
```

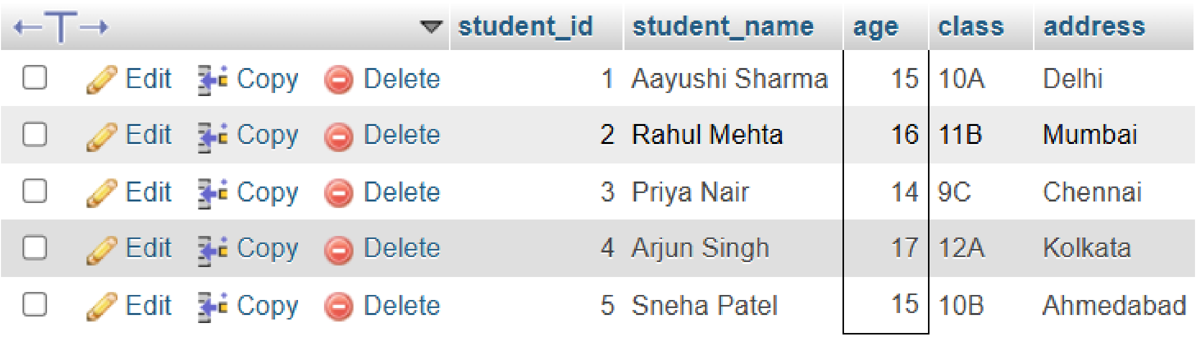


←T→			▼	student_name	age
□	Edit	Copy	Delete	Aayushi Sharma	15
□	Edit	Copy	Delete	Rahul Mehta	16
□	Edit	Copy	Delete	Priya Nair	14
□	Edit	Copy	Delete	Arjun Singh	17
□	Edit	Copy	Delete	Sneha Patel	15

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



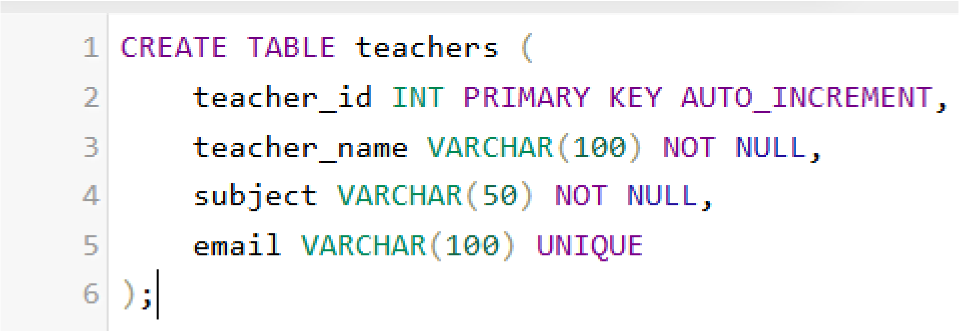
```sql
SELECT *
FROM students
WHERE age > 10;
```



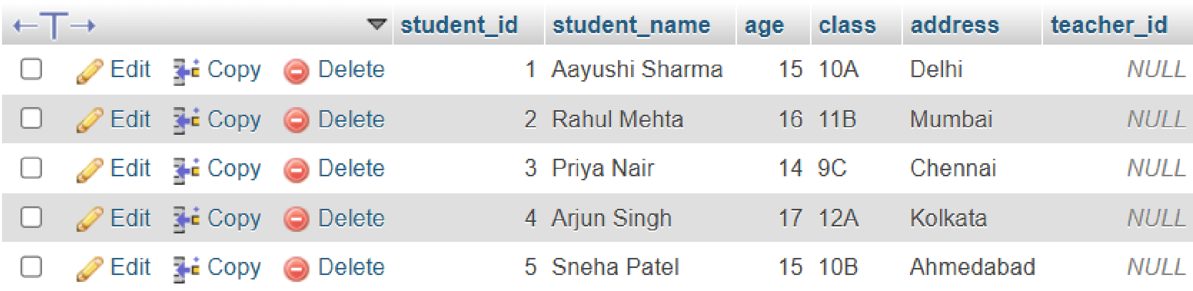
←T→			▼	student_id	student_name	age	class	address
□	Edit	Copy	Delete	1	Aayushi Sharma	15	10A	Delhi
□	Edit	Copy	Delete	2	Rahul Mehta	16	11B	Mumbai
□	Edit	Copy	Delete	3	Priya Nair	14	9C	Chennai
□	Edit	Copy	Delete	4	Arjun Singh	17	12A	Kolkata
□	Edit	Copy	Delete	5	Sneha Patel	15	10B	Ahmedabad

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).

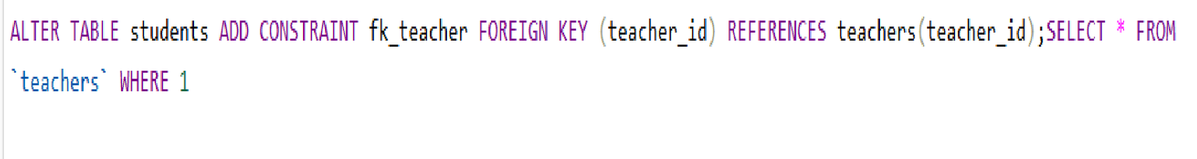


```sql
CREATE TABLE teachers (
    teacher_id INT PRIMARY KEY AUTO_INCREMENT,
    teacher_name VARCHAR(100) NOT NULL,
    subject VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE
);
```

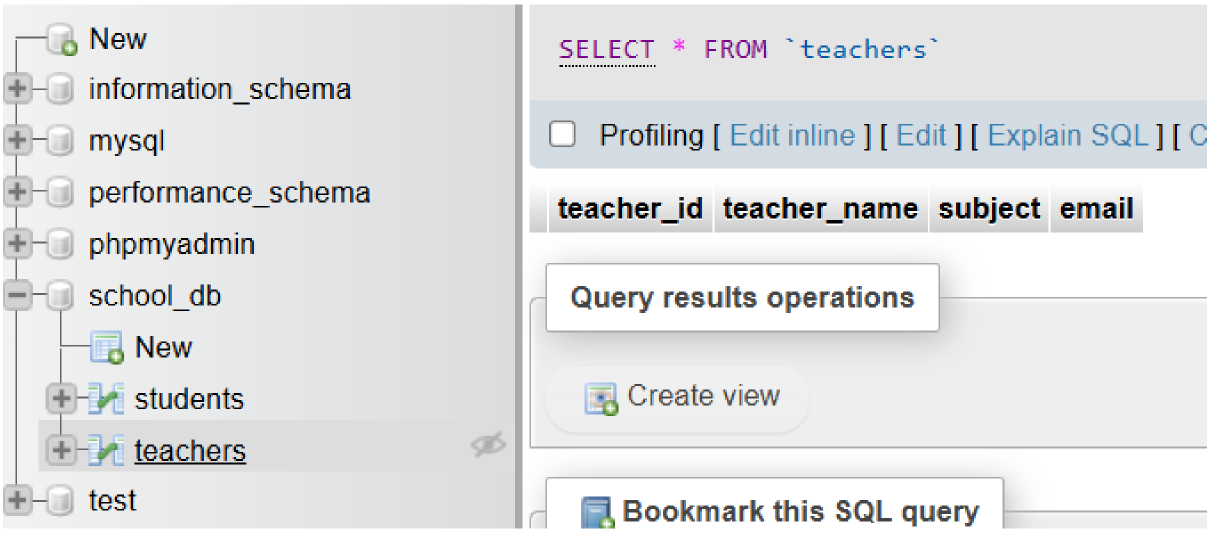


←T→	student_id	student_name	age	class	address	teacher_id
□ Edit Copy Delete	1	Aayushi Sharma	15	10A	Delhi	*NULL*
□ Edit Copy Delete	2	Rahul Mehta	16	11B	Mumbai	*NULL*
□ Edit Copy Delete	3	Priya Nair	14	9C	Chennai	*NULL*
□ Edit Copy Delete	4	Arjun Singh	17	12A	Kolkata	*NULL*
□ Edit Copy Delete	5	Sneha Patel	15	10B	Ahmedabad	*NULL*

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



```sql
ALTER TABLE students ADD CONSTRAINT fk_teacher FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id);SELECT * FROM
`teachers` WHERE 1
```

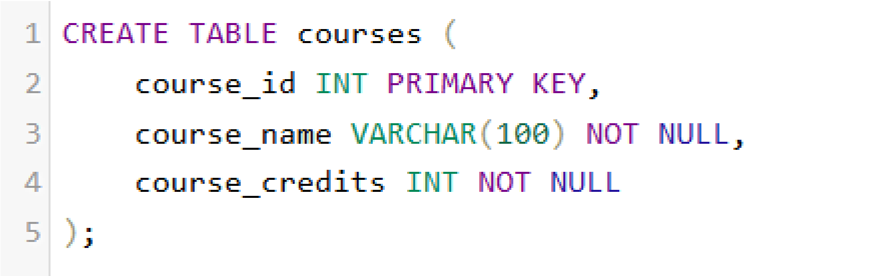


```sql
SELECT * FROM `teachers`
```

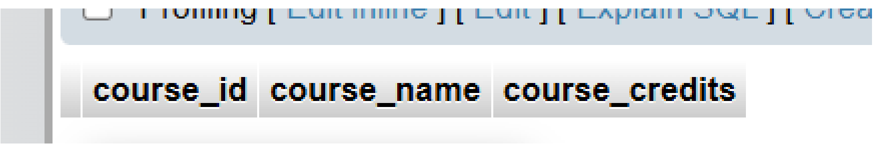
teacher_id	teacher_name	subject	email

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.

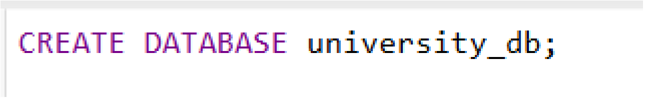


```sql
CREATE TABLE courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL,
    course_credits INT NOT NULL
);
```

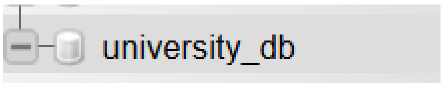


course_id	course_name	course_credits

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

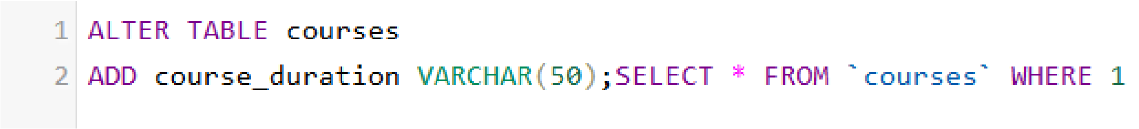


```sql
CREATE DATABASE university_db;
```

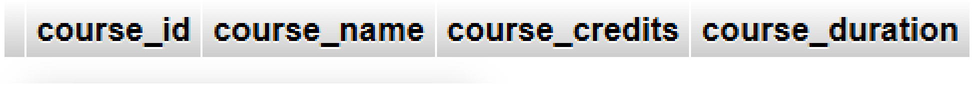


5. ALTER Command

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

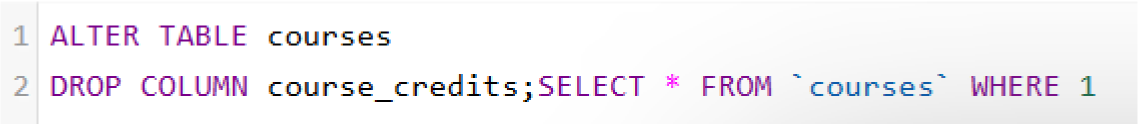


```
ALTER TABLE courses
ADD course_duration VARCHAR(50);SELECT * FROM `courses` WHERE 1
```

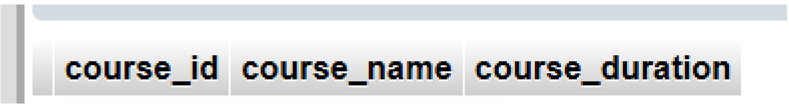


course_id	course_name	course_credits	course_duration

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



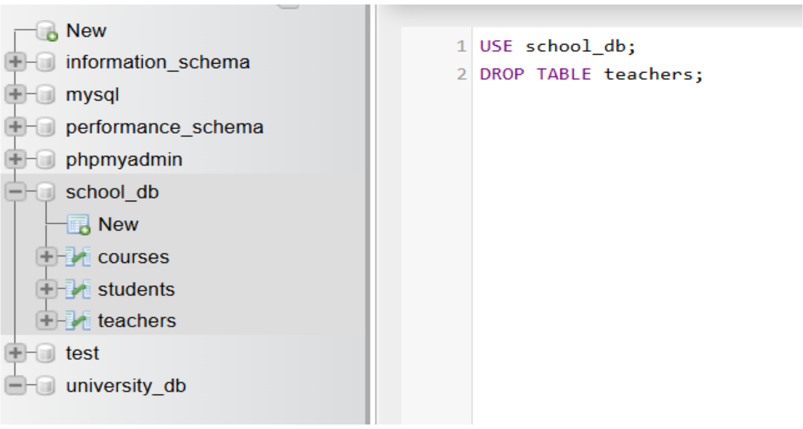
```
ALTER TABLE courses
DROP COLUMN course_credits;SELECT * FROM `courses` WHERE 1
```



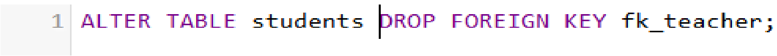
course_id	course_name	course_duration

6. DROP Command

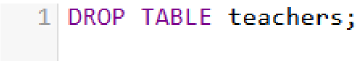
Lab 1: Drop the teachers table from the school_db database.



```sql
USE school_db;
DROP TABLE teachers;
```

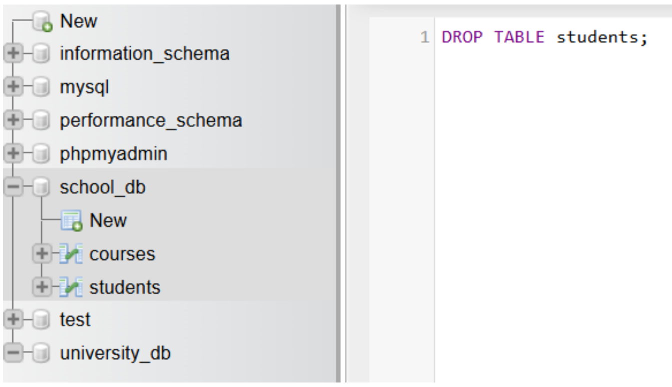


```sql
ALTER TABLE students DROP FOREIGN KEY fk_teacher;
```

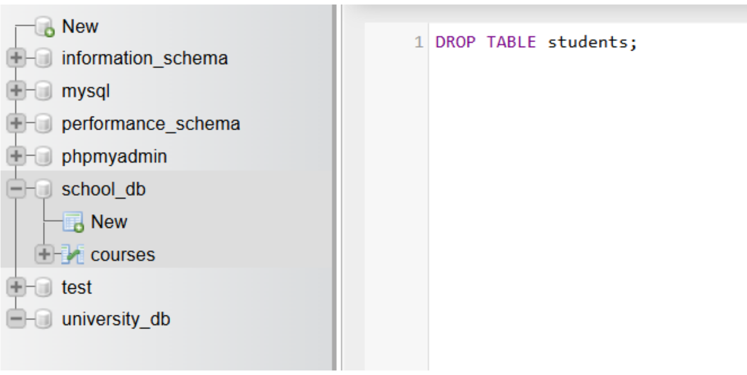


```sql
DROP TABLE teachers;
```

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



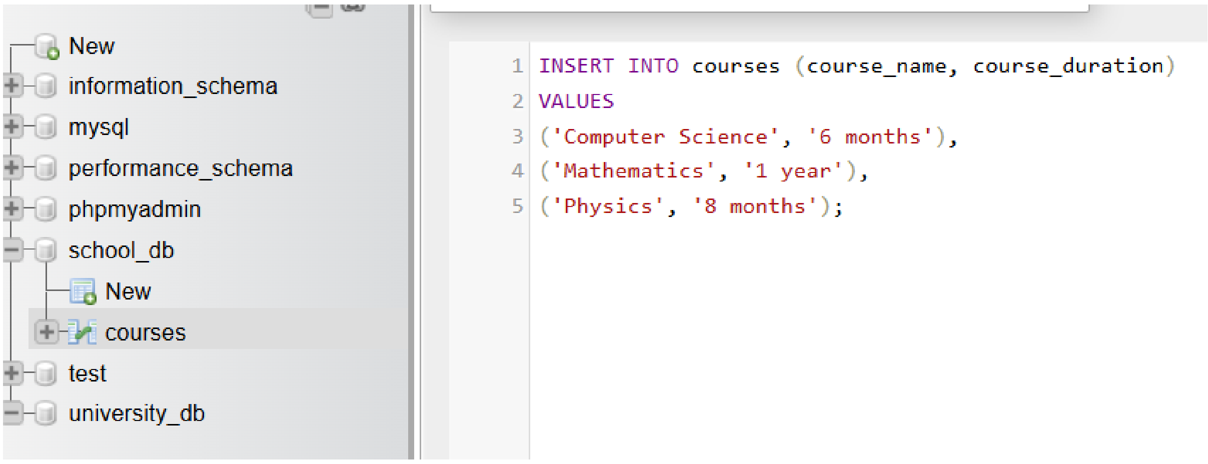
```sql
DROP TABLE students;
```



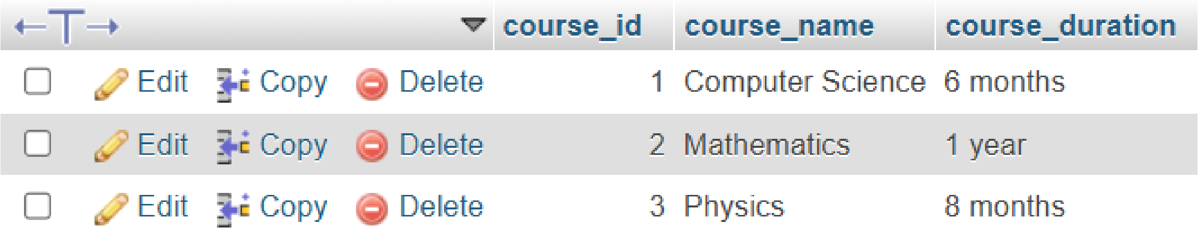
```sql
DROP TABLE students;
```

7. Data Manipulation Language (DML)

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

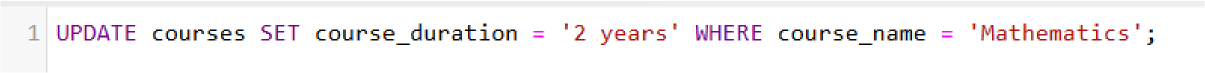


```sql
INSERT INTO courses (course_name, course_duration)
VALUES
('Computer Science', '6 months'),
('Mathematics', '1 year'),
('Physics', '8 months');
```

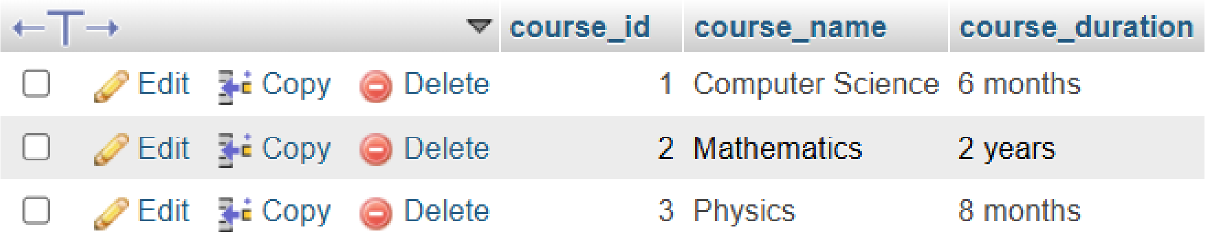


←T→		course_id	course_name	course_duration
□	Edit Copy Delete	1	Computer Science	6 months
□	Edit Copy Delete	2	Mathematics	1 year
□	Edit Copy Delete	3	Physics	8 months

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

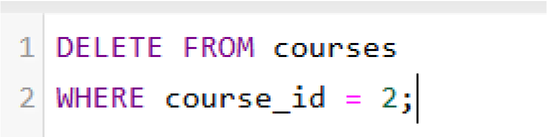


```sql
UPDATE courses SET course_duration = '2 years' WHERE course_name = 'Mathematics';
```

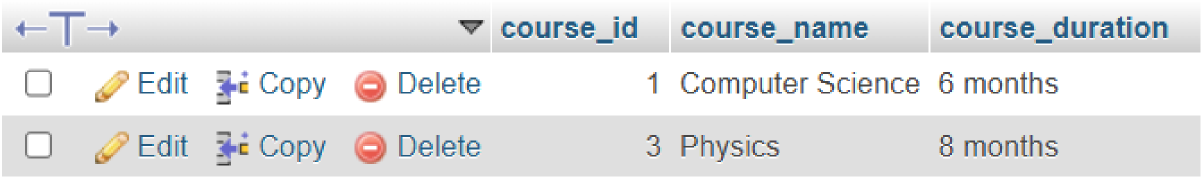


←T→		course_id	course_name	course_duration
□	Edit Copy Delete	1	Computer Science	6 months
□	Edit Copy Delete	2	Mathematics	2 years
□	Edit Copy Delete	3	Physics	8 months

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



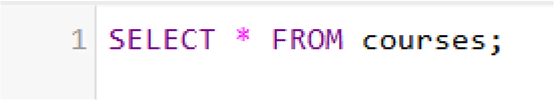
```sql
DELETE FROM courses
WHERE course_id = 2;
```



←T→ ▼	course_id	course_name	course_duration
☐ Edit Copy Delete	1	Computer Science	6 months
☐ Edit Copy Delete	3	Physics	8 months

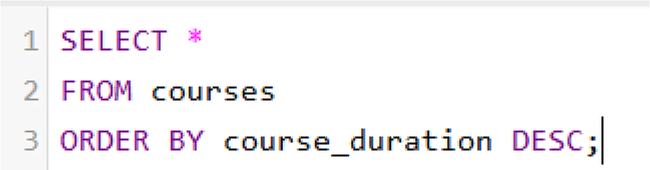
8. Data Query Language (DQL)

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

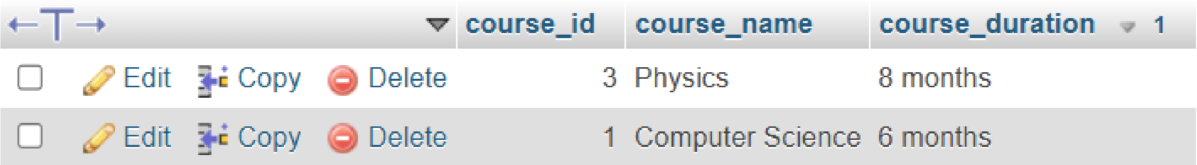


```sql
SELECT * FROM courses;
```

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

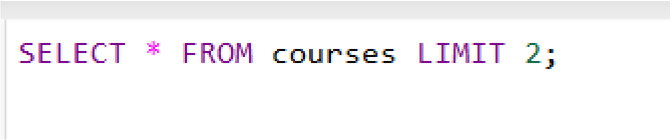


```sql
SELECT *
FROM courses
ORDER BY course_duration DESC;
```

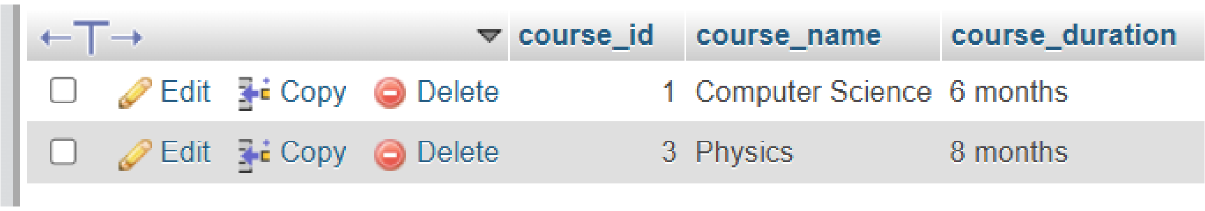


←T→	course_id	course_name	course_duration ▾ 1
☐ Edit Copy Delete	3	Physics	8 months
☐ Edit Copy Delete	1	Computer Science	6 months

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



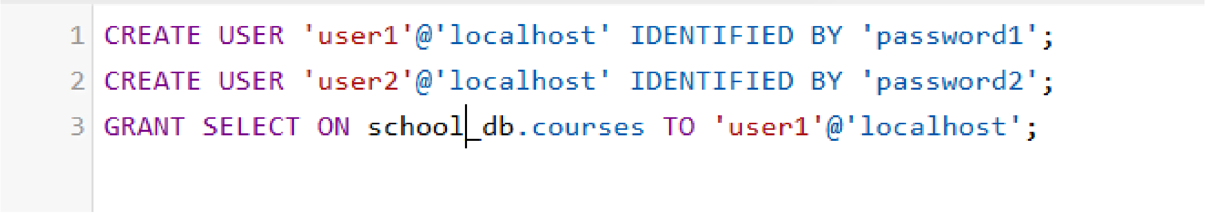
```sql
SELECT * FROM courses LIMIT 2;
```



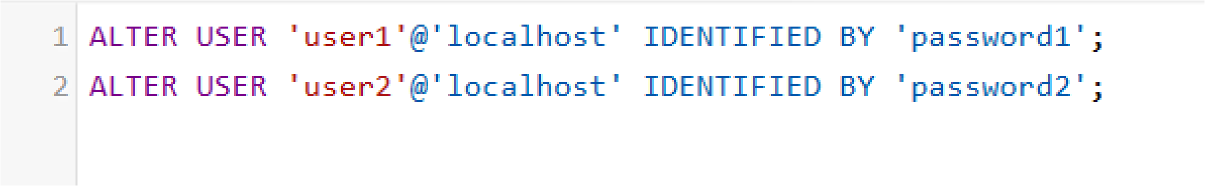
←T→	course_id	course_name	course_duration
☐ Edit Copy Delete	1	Computer Science	6 months
☐ Edit Copy Delete	3	Physics	8 months

9. Data Control Language (DCL)

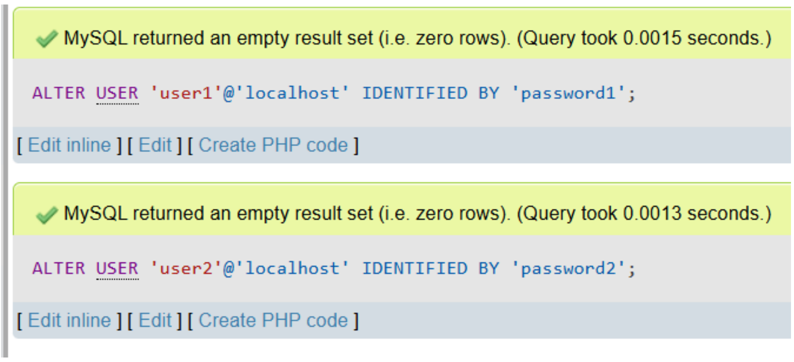
Lab 1: Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.



```
CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password1';
CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';
GRANT SELECT ON school_db.courses TO 'user1'@'localhost';
```



```
ALTER USER 'user1'@'localhost' IDENTIFIED BY 'password1';
ALTER USER 'user2'@'localhost' IDENTIFIED BY 'password2';
```



✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0015 seconds.)

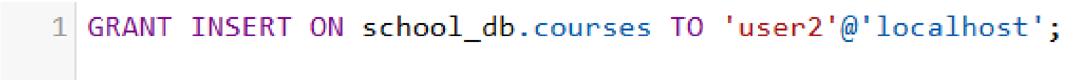
```
ALTER USER 'user1'@'localhost' IDENTIFIED BY 'password1';
```

[Edit inline] [Edit] [Create PHP code]

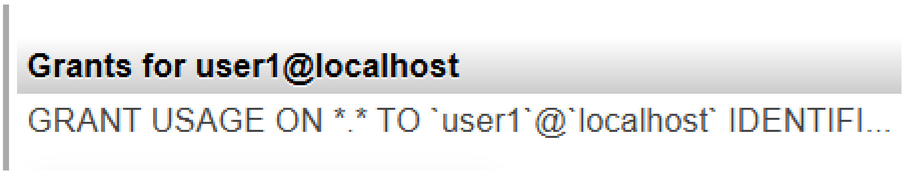
✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0013 seconds.)

```
ALTER USER 'user2'@'localhost' IDENTIFIED BY 'password2';
```

[Edit inline] [Edit] [Create PHP code]

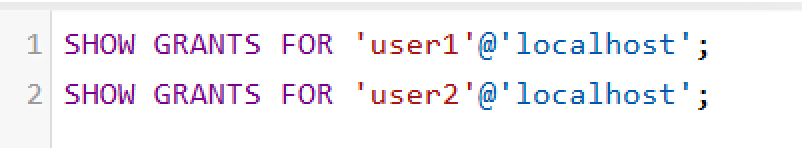


```
GRANT INSERT ON school_db.courses TO 'user2'@'localhost';
```



Grants for user1@localhost
GRANT USAGE ON *.* TO \`user1\`@\`localhost\` IDENTIFI...

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



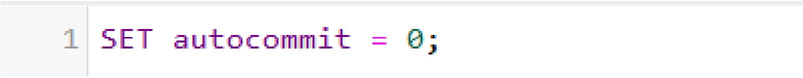
```
SHOW GRANTS FOR 'user1'@'localhost';
SHOW GRANTS FOR 'user2'@'localhost';
```

User1- only has select (no insert)

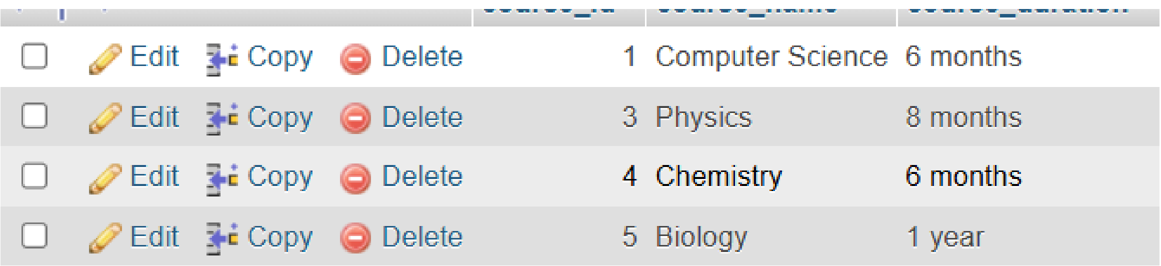
User2-has insert permission on school_db.courses.

10. Transaction Control Language (TCL)

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

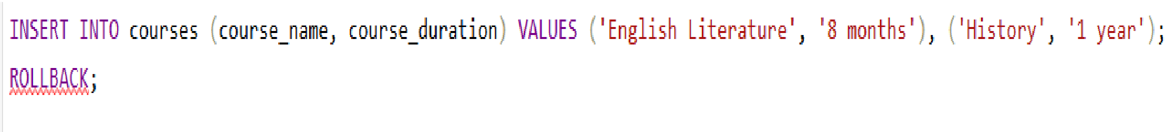


```
SET autocommit = 0;
```

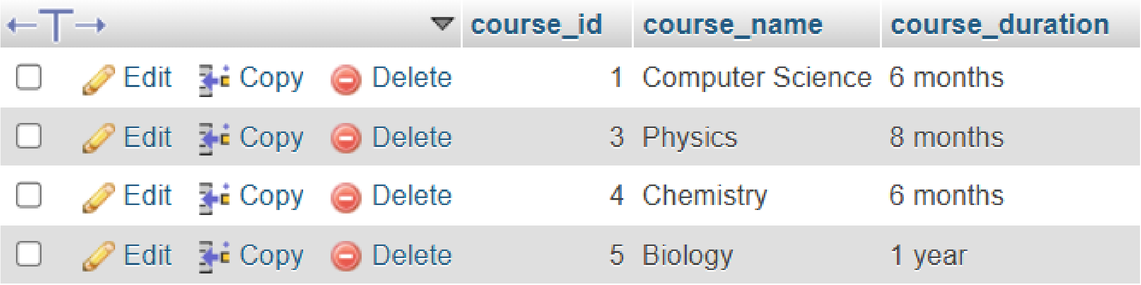


				course_id	course_name	course_duration
□	Edit	Copy	Delete	1	Computer Science	6 months
□	Edit	Copy	Delete	3	Physics	8 months
□	Edit	Copy	Delete	4	Chemistry	6 months
□	Edit	Copy	Delete	5	Biology	1 year

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

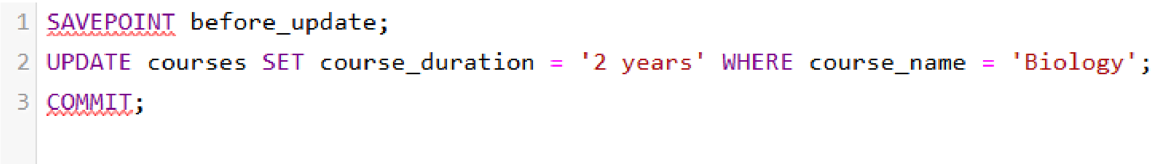


```
INSERT INTO courses (course_name, course_duration) VALUES ('English Literature', '8 months'), ('History', '1 year');
ROLLBACK;
```

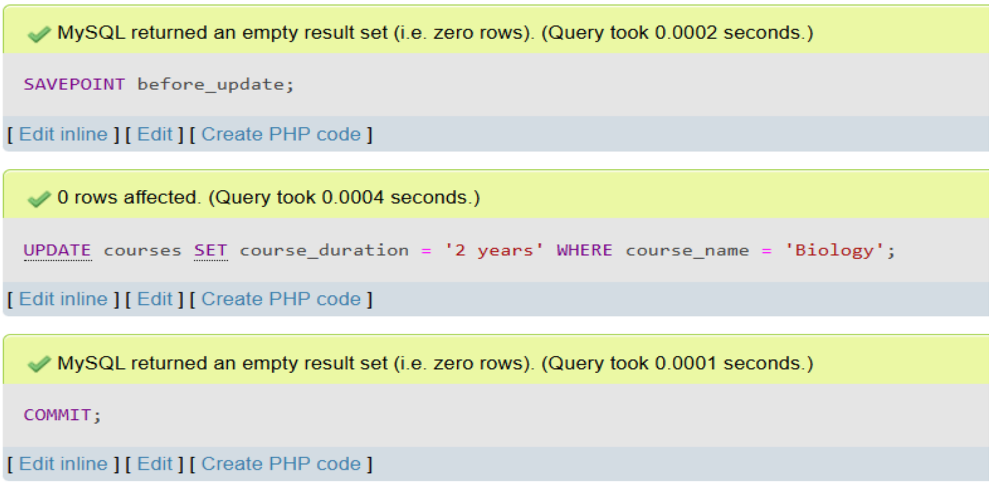


				course_id	course_name	course_duration
□	Edit	Copy	Delete	1	Computer Science	6 months
□	Edit	Copy	Delete	3	Physics	8 months
□	Edit	Copy	Delete	4	Chemistry	6 months
□	Edit	Copy	Delete	5	Biology	1 year

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



```
SAVEPOINT before_update;
UPDATE courses SET course_duration = '2 years' WHERE course_name = 'Biology';
COMMIT;
```



✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0002 seconds.)

```
SAVEPOINT before_update;
```

[Edit inline] [Edit] [Create PHP code]

✓ 0 rows affected. (Query took 0.0004 seconds.)

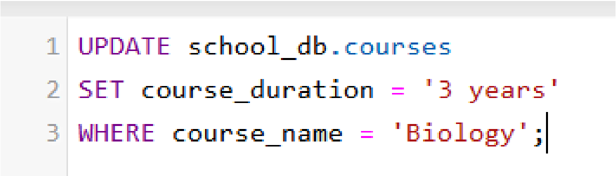
```
UPDATE courses SET course_duration = '2 years' WHERE course_name = 'Biology';
```

[Edit inline] [Edit] [Create PHP code]

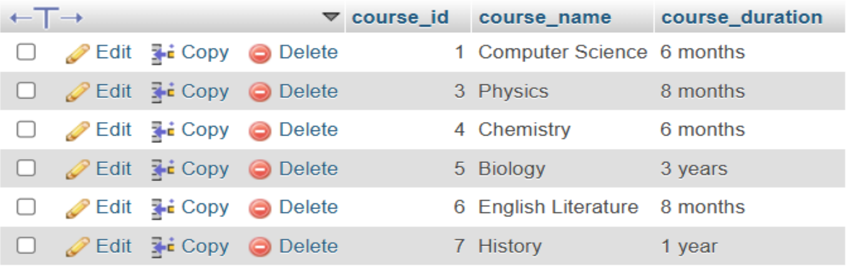
✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0001 seconds.)

```
COMMIT;
```

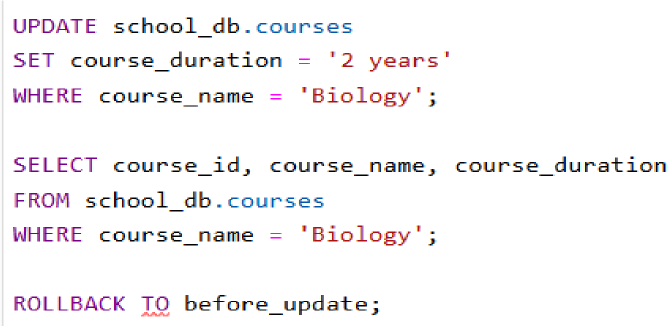
[Edit inline] [Edit] [Create PHP code]



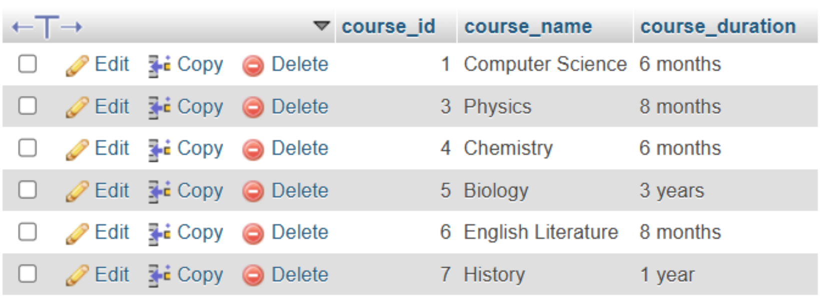
```
UPDATE school_db.courses
SET course_duration = '3 years'
WHERE course_name = 'Biology';
```



←T→				course_id	course_name	course_duration
□	Edit	Copy	Delete	1	Computer Science	6 months
□	Edit	Copy	Delete	3	Physics	8 months
□	Edit	Copy	Delete	4	Chemistry	6 months
□	Edit	Copy	Delete	5	Biology	3 years
□	Edit	Copy	Delete	6	English Literature	8 months
□	Edit	Copy	Delete	7	History	1 year



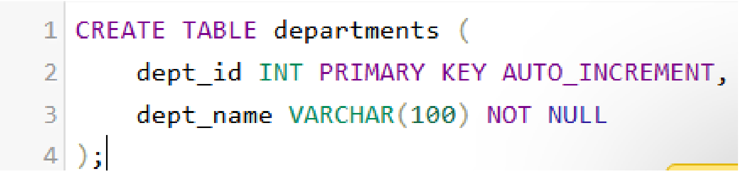
```
UPDATE school_db.courses
SET course_duration = '2 years'
WHERE course_name = 'Biology';

SELECT course_id, course_name, course_duration
FROM school_db.courses
WHERE course_name = 'Biology';

ROLLBACK TO before_update;
```



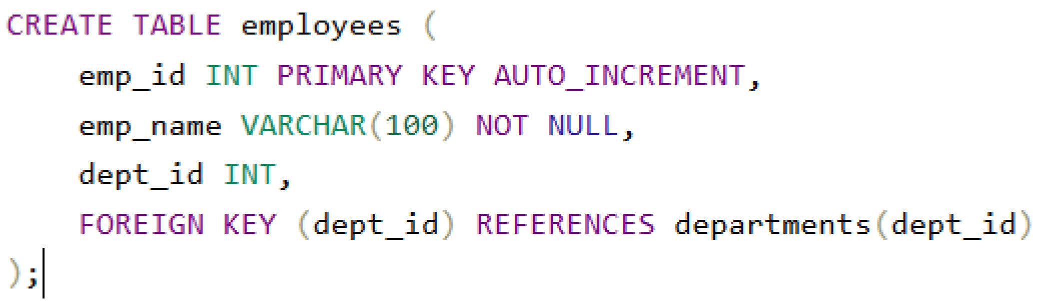
←T→				course_id	course_name	course_duration
□	Edit	Copy	Delete	1	Computer Science	6 months
□	Edit	Copy	Delete	3	Physics	8 months
□	Edit	Copy	Delete	4	Chemistry	6 months
□	Edit	Copy	Delete	5	Biology	3 years
□	Edit	Copy	Delete	6	English Literature	8 months
□	Edit	Copy	Delete	7	History	1 year

11. SQL Joins

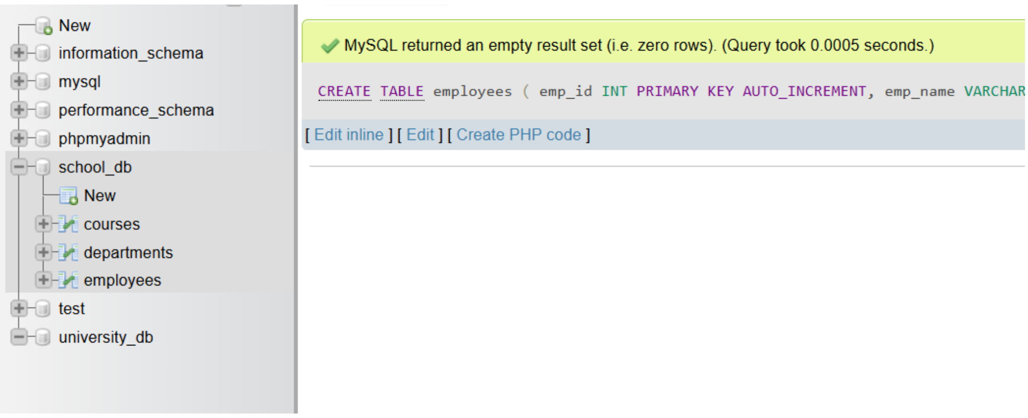
Lab 1: Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.



```
CREATE TABLE departments (
    dept_id INT PRIMARY KEY AUTO_INCREMENT,
    dept_name VARCHAR(100) NOT NULL
);
```



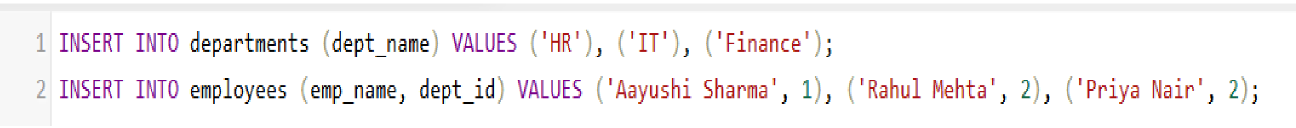
```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY AUTO_INCREMENT,
    emp_name VARCHAR(100) NOT NULL,
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);
```



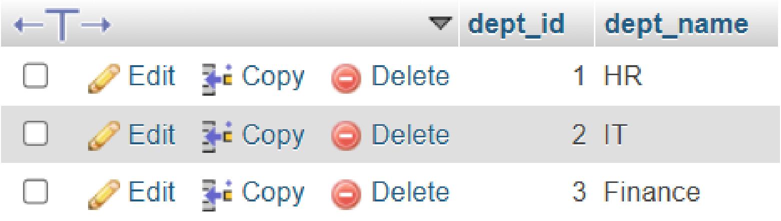
✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0005 seconds.)

```
CREATE TABLE employees ( emp_id INT PRIMARY KEY AUTO_INCREMENT, emp_name VARCHAR
```

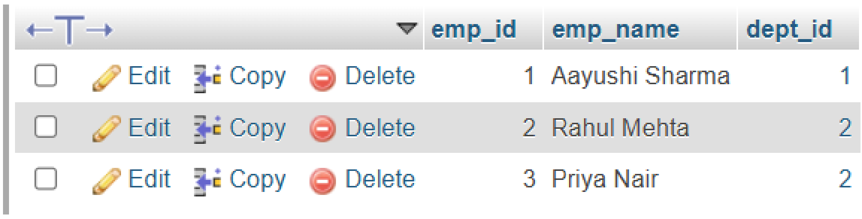
[Edit inline] [Edit] [Create PHP code]



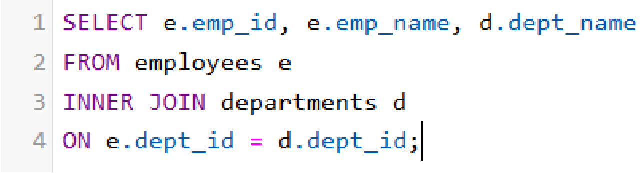
```
INSERT INTO departments (dept_name) VALUES ('HR'), ('IT'), ('Finance');
INSERT INTO employees (emp_name, dept_id) VALUES ('Aayushi Sharma', 1), ('Rahul Mehta', 2), ('Priya Nair', 2);
```



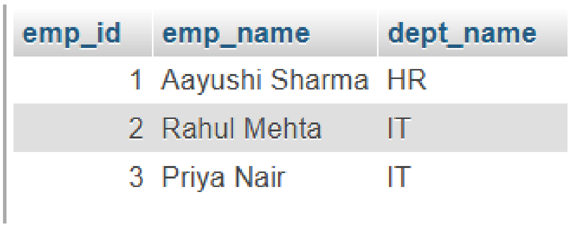
←T→			▼	dept_id	dept_name
□	Edit	Copy	Delete	1	HR
□	Edit	Copy	Delete	2	IT
□	Edit	Copy	Delete	3	Finance



←T→	emp_id	emp_name	dept_id
□ Edit Copy Delete	1	Aayushi Sharma	1
□ Edit Copy Delete	2	Rahul Mehta	2
□ Edit Copy Delete	3	Priya Nair	2

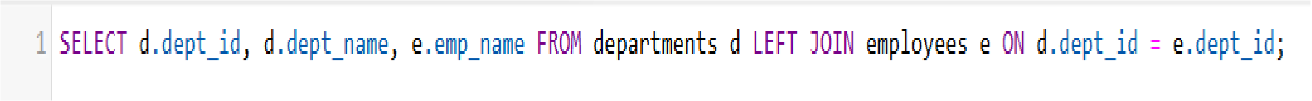


```sql
SELECT e.emp_id, e.emp_name, d.dept_name
FROM employees e
INNER JOIN departments d
ON e.dept_id = d.dept_id;
```

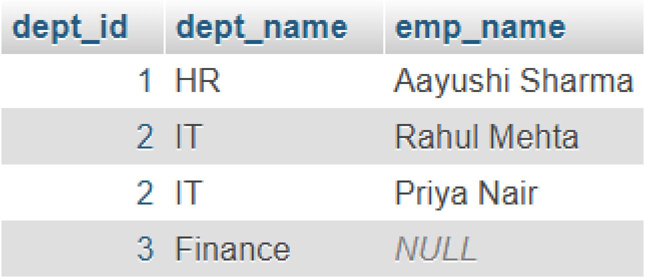


emp_id	emp_name	dept_name
1	Aayushi Sharma	HR
2	Rahul Mehta	IT
3	Priya Nair	IT

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



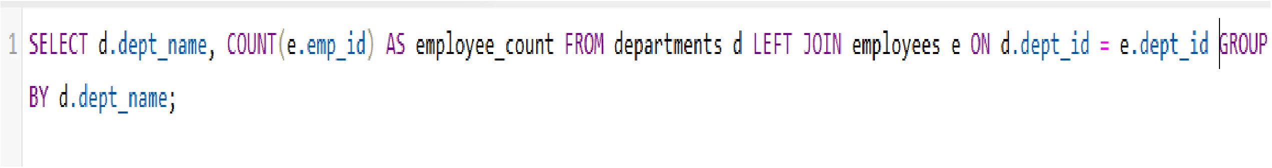
```sql
SELECT d.dept_id, d.dept_name, e.emp_name FROM departments d LEFT JOIN employees e ON d.dept_id = e.dept_id;
```



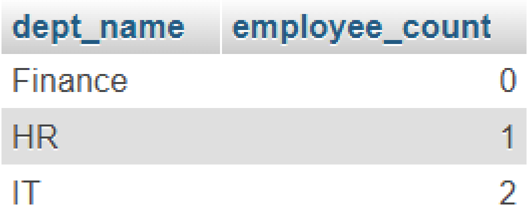
dept_id	dept_name	emp_name
1	HR	Aayushi Sharma
2	IT	Rahul Mehta
2	IT	Priya Nair
3	Finance	*NULL*

12. SQL Group By

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

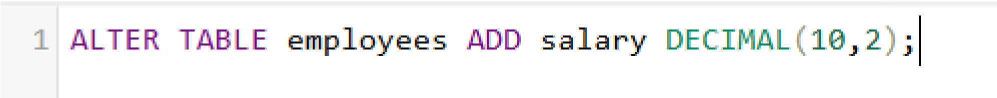


```sql
SELECT d.dept_name, COUNT(e.emp_id) AS employee_count FROM departments d LEFT JOIN employees e ON d.dept_id = e.dept_id GROUP BY d.dept_name;
```

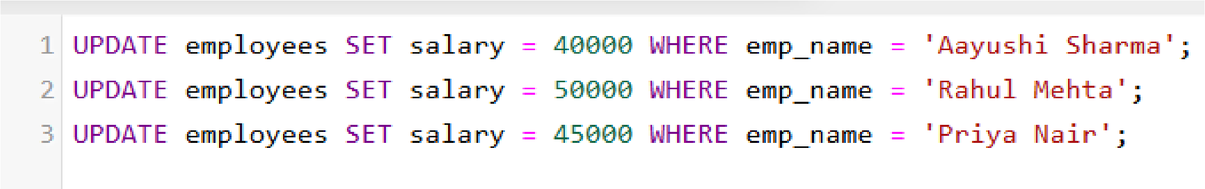


dept_name	employee_count
Finance	0
HR	1
IT	2

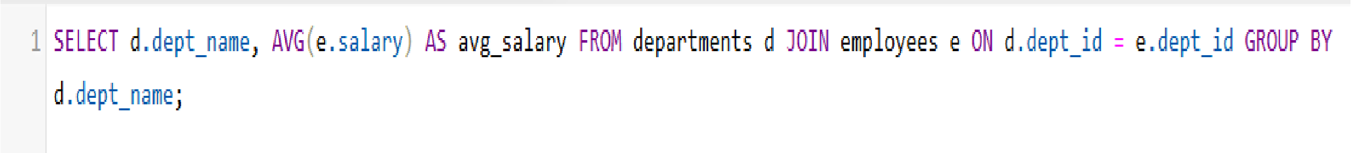
Lab 2: Use the AVG aggregate function to find the average salary of employees in each department.



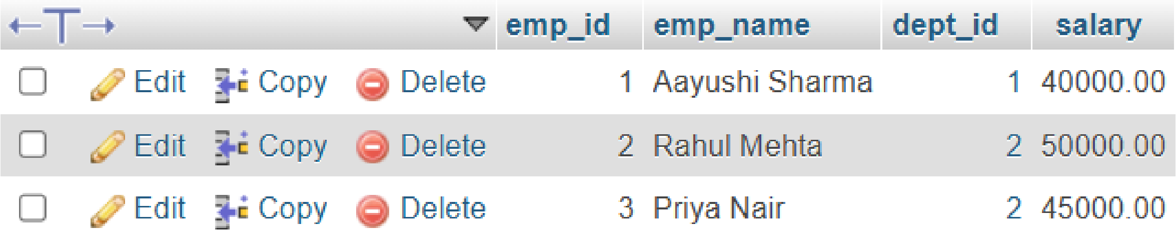
```sql
ALTER TABLE employees ADD salary DECIMAL(10,2);
```



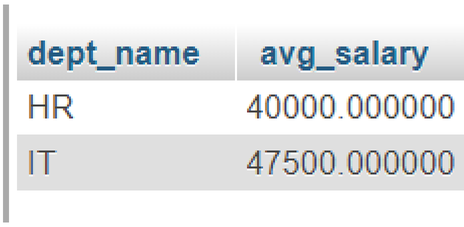
```sql
UPDATE employees SET salary = 40000 WHERE emp_name = 'Aayushi Sharma';
UPDATE employees SET salary = 50000 WHERE emp_name = 'Rahul Mehta';
UPDATE employees SET salary = 45000 WHERE emp_name = 'Priya Nair';
```



```sql
SELECT d.dept_name, AVG(e.salary) AS avg_salary FROM departments d JOIN employees e ON d.dept_id = e.dept_id GROUP BY d.dept_name;
```



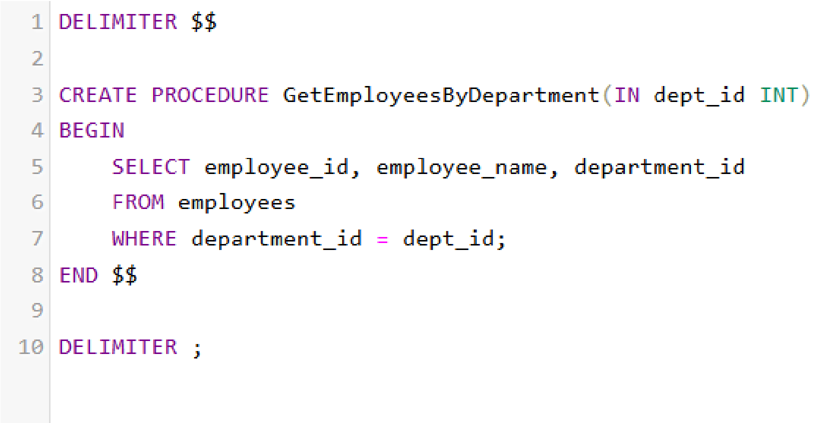
←T→			▼	emp_id	emp_name	dept_id	salary
□	Edit	Copy	Delete	1	Aayushi Sharma	1	40000.00
□	Edit	Copy	Delete	2	Rahul Mehta	2	50000.00
□	Edit	Copy	Delete	3	Priya Nair	2	45000.00



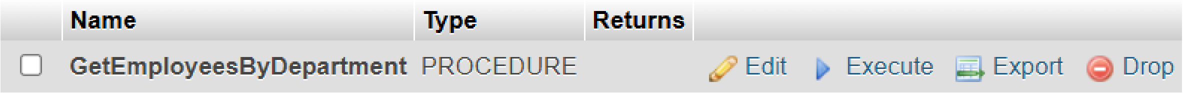
dept_name	avg_salary
HR	40000.000000
IT	47500.000000

13. SQL Stored Procedure

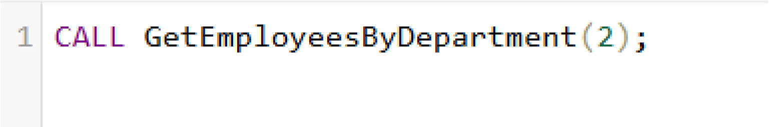
Lab 1: Write a stored procedure to retrieve all employees from the employees table based on department.



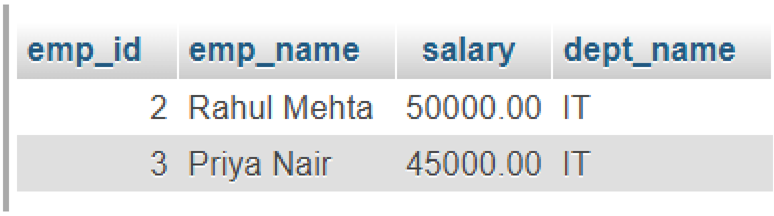
```sql
DELIMITER $$

CREATE PROCEDURE GetEmployeesByDepartment(IN dept_id INT)
BEGIN
    SELECT employee_id, employee_name, department_id
    FROM employees
    WHERE department_id = dept_id;
END $$

DELIMITER ;
```



	Name	Type	Returns	
□	GetEmployeesByDepartment	PROCEDURE		Edit Execute Export Drop

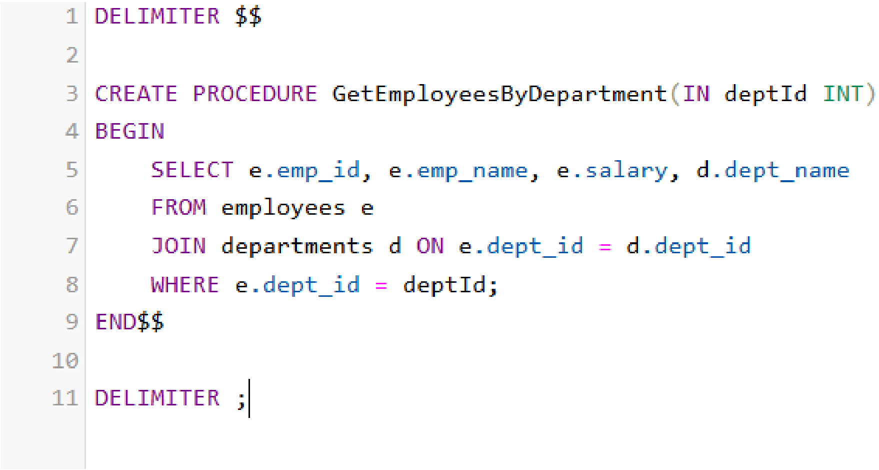


```sql
CALL GetEmployeesByDepartment(2);
```

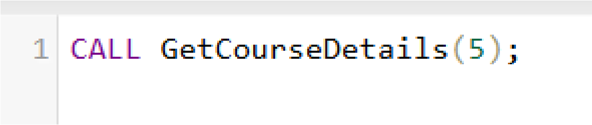


emp_id	emp_name	salary	dept_name
2	Rahul Mehta	50000.00	IT
3	Priya Nair	45000.00	IT

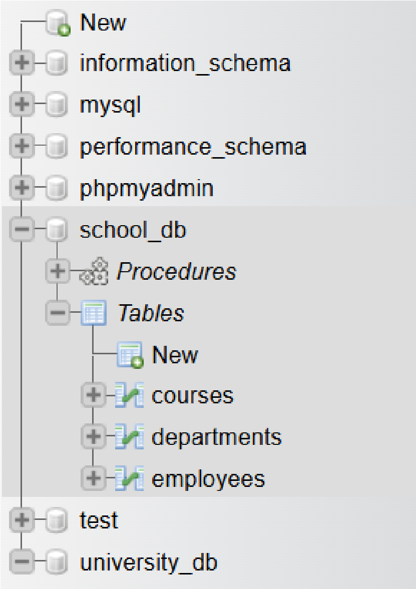
Lab 2: Write a stored procedure that accepts course_id as input and returns the course details.

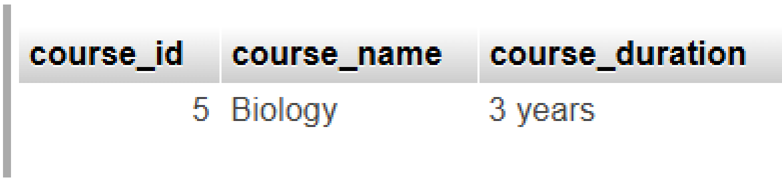


```sql
DELIMITER $$

CREATE PROCEDURE GetEmployeesByDepartment(IN deptId INT)
BEGIN
    SELECT e.emp_id, e.emp_name, e.salary, d.dept_name
    FROM employees e
    JOIN departments d ON e.dept_id = d.dept_id
    WHERE e.dept_id = deptId;
END$$

DELIMITER ;
```



```sql
CALL GetCourseDetails(5);
```

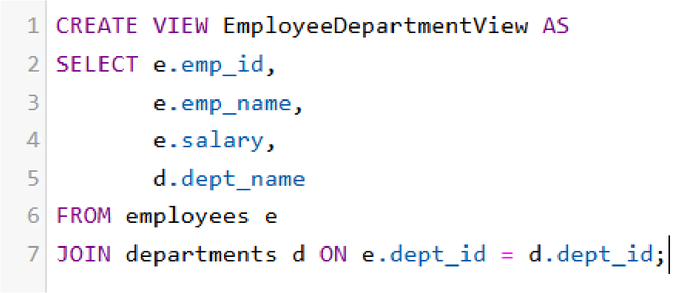




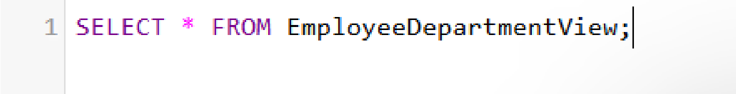
course_id	course_name	course_duration
5	Biology	3 years

14. SQL View

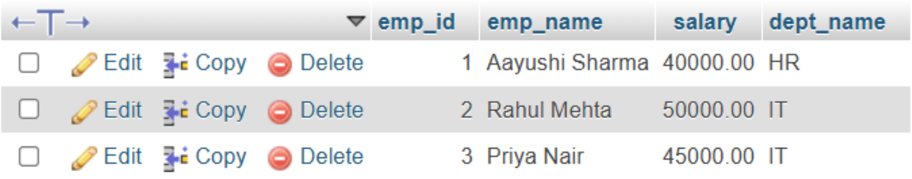
Lab 1: Create a view to show all employees along with their department names.



```
CREATE VIEW EmployeeDepartmentView AS
SELECT e.emp_id,
       e.emp_name,
       e.salary,
       d.dept_name
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id;
```

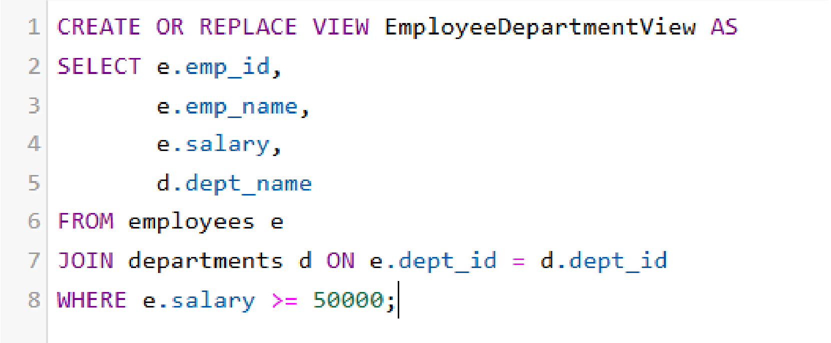


```
SELECT * FROM EmployeeDepartmentView;
```

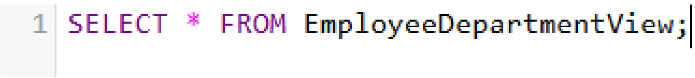


←T→	emp_id	emp_name	salary	dept_name
□ Edit Copy Delete	1	Aayushi Sharma	40000.00	HR
□ Edit Copy Delete	2	Rahul Mehta	50000.00	IT
□ Edit Copy Delete	3	Priya Nair	45000.00	IT

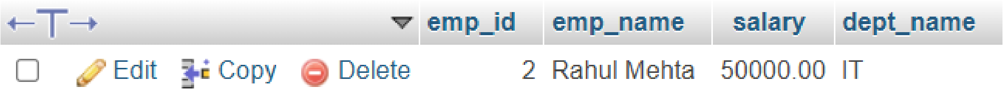
Lab 2: Modify the view to exclude employees whose salaries are below $50,000.



```
CREATE OR REPLACE VIEW EmployeeDepartmentView AS
SELECT e.emp_id,
       e.emp_name,
       e.salary,
       d.dept_name
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id
WHERE e.salary >= 50000;
```



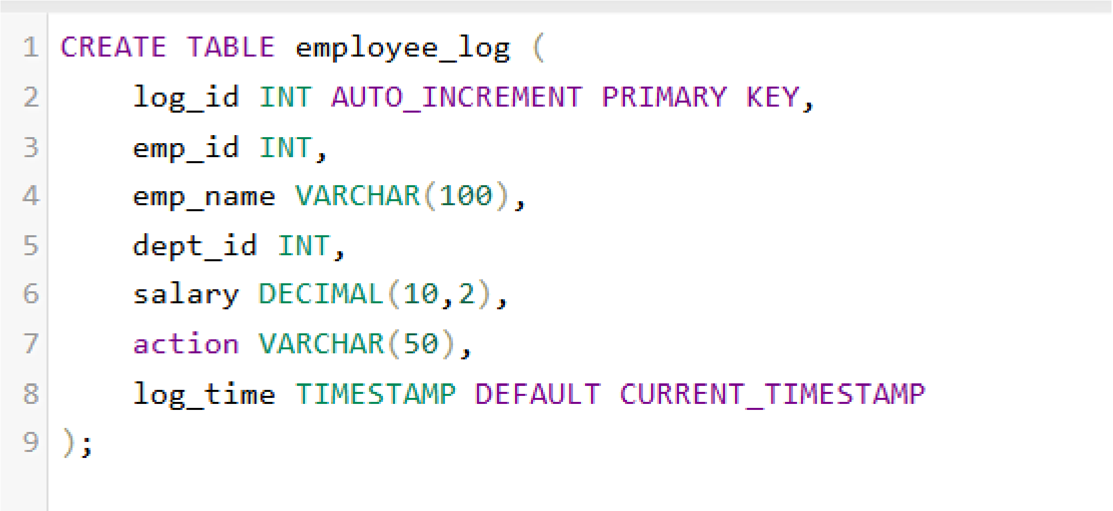
```
SELECT * FROM EmployeeDepartmentView;
```



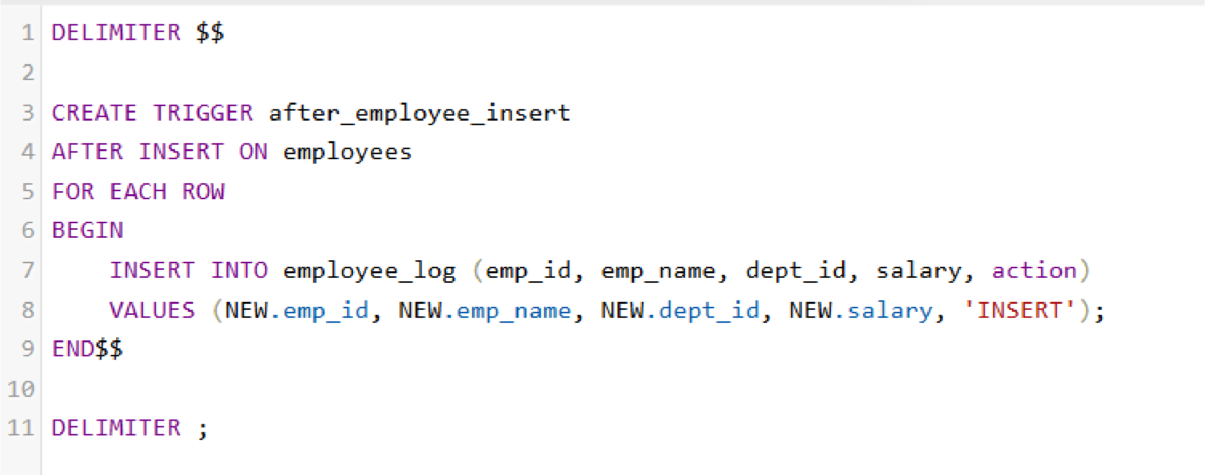
←T→	emp_id	emp_name	salary	dept_name
□ Edit Copy Delete	2	Rahul Mehta	50000.00	IT

15. SQL Triggers

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

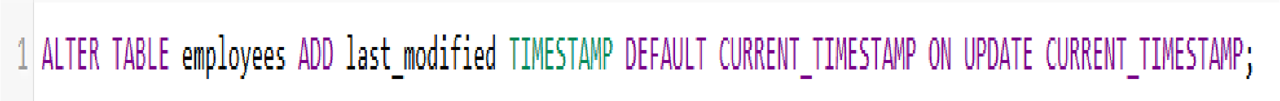


```sql
CREATE TABLE employee_log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    emp_id INT,
    emp_name VARCHAR(100),
    dept_id INT,
    salary DECIMAL(10,2),
    action VARCHAR(50),
    log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

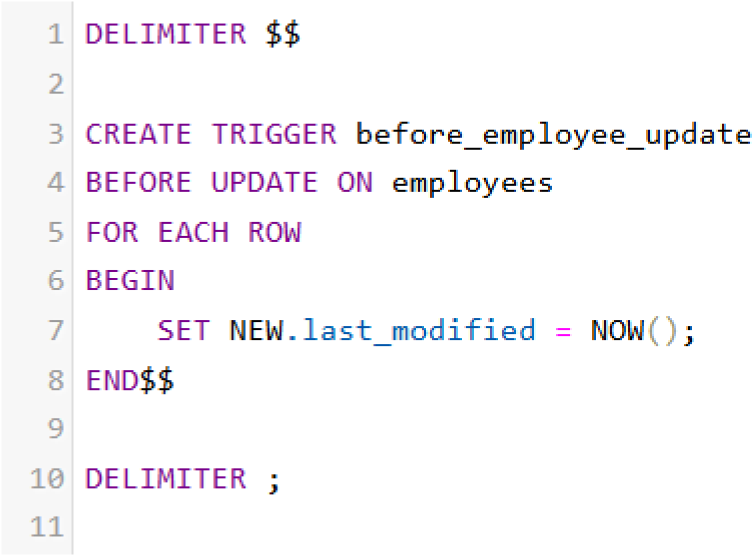


```sql
DELIMITER $$

CREATE TRIGGER after_employee_insert
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO employee_log (emp_id, emp_name, dept_id, salary, action)
    VALUES (NEW.emp_id, NEW.emp_name, NEW.dept_id, NEW.salary, 'INSERT');
END$$

DELIMITER ;
```

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



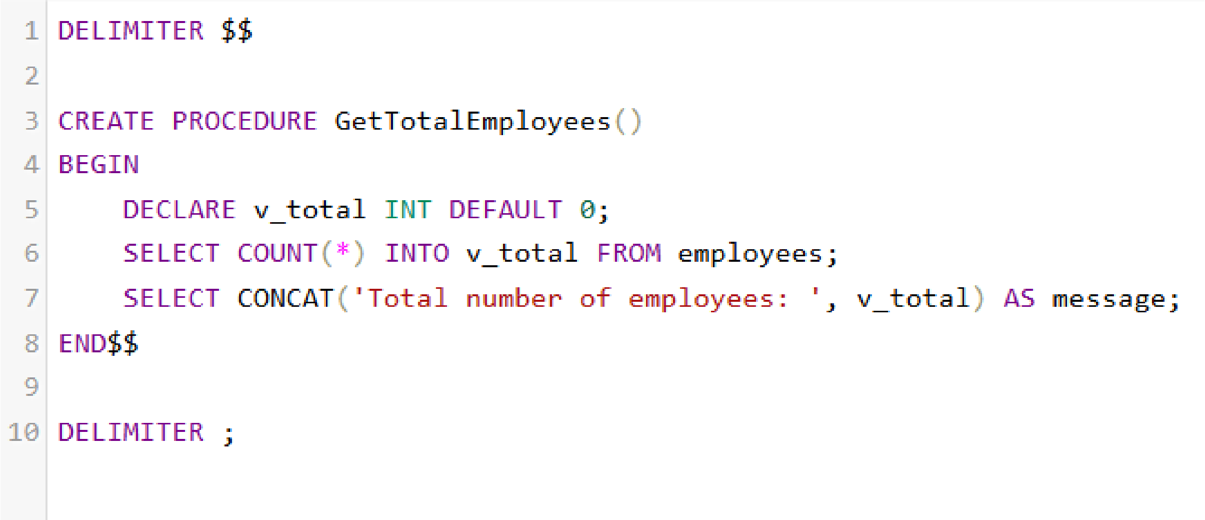
```sql
ALTER TABLE employees ADD last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;
```



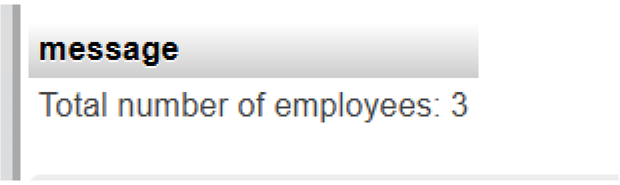
```sql
DELIMITER $$

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

DELIMITER ;

```

16. Introduction to PL/SQL

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

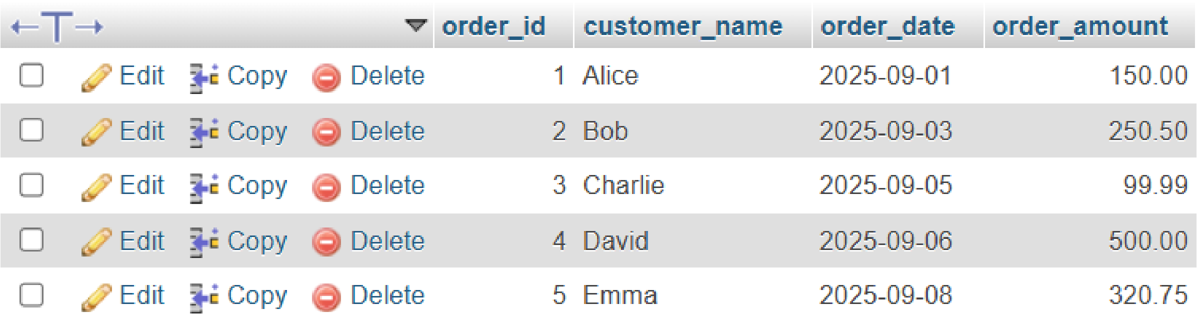


```
DELIMITER $$

CREATE PROCEDURE GetTotalEmployees()
BEGIN
    DECLARE v_total INT DEFAULT 0;
    SELECT COUNT(*) INTO v_total FROM employees;
    SELECT CONCAT('Total number of employees: ', v_total) AS message;
END$$

DELIMITER ;
```

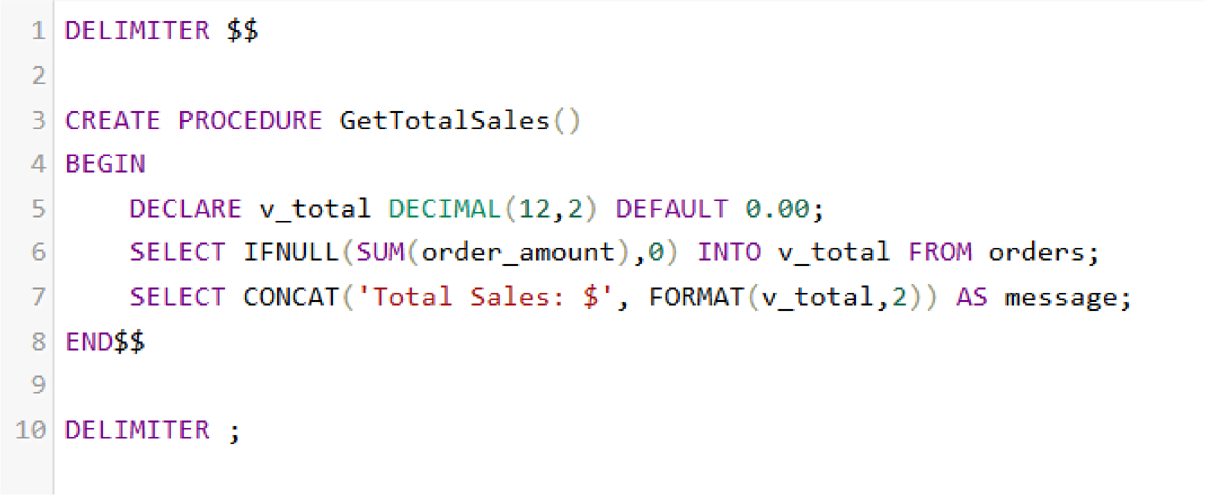


message
Total number of employees: 3

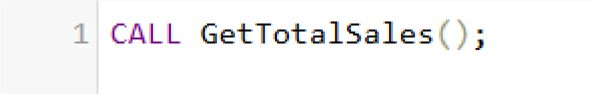
Lab 2: Create a PL/SQL block that calculates the total sales from an orders table.



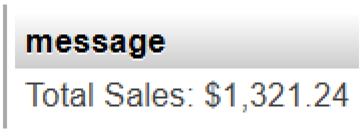
←T→				order_id	customer_name	order_date	order_amount
□	Edit	Copy	Delete	1	Alice	2025-09-01	150.00
□	Edit	Copy	Delete	2	Bob	2025-09-03	250.50
□	Edit	Copy	Delete	3	Charlie	2025-09-05	99.99
□	Edit	Copy	Delete	4	David	2025-09-06	500.00
□	Edit	Copy	Delete	5	Emma	2025-09-08	320.75



```sql
DELIMITER $$

CREATE PROCEDURE GetTotalSales()
BEGIN
    DECLARE v_total DECIMAL(12,2) DEFAULT 0.00;
    SELECT IFNULL(SUM(order_amount),0) INTO v_total FROM orders;
    SELECT CONCAT('Total Sales: $', FORMAT(v_total,2)) AS message;
END$$

DELIMITER ;
```



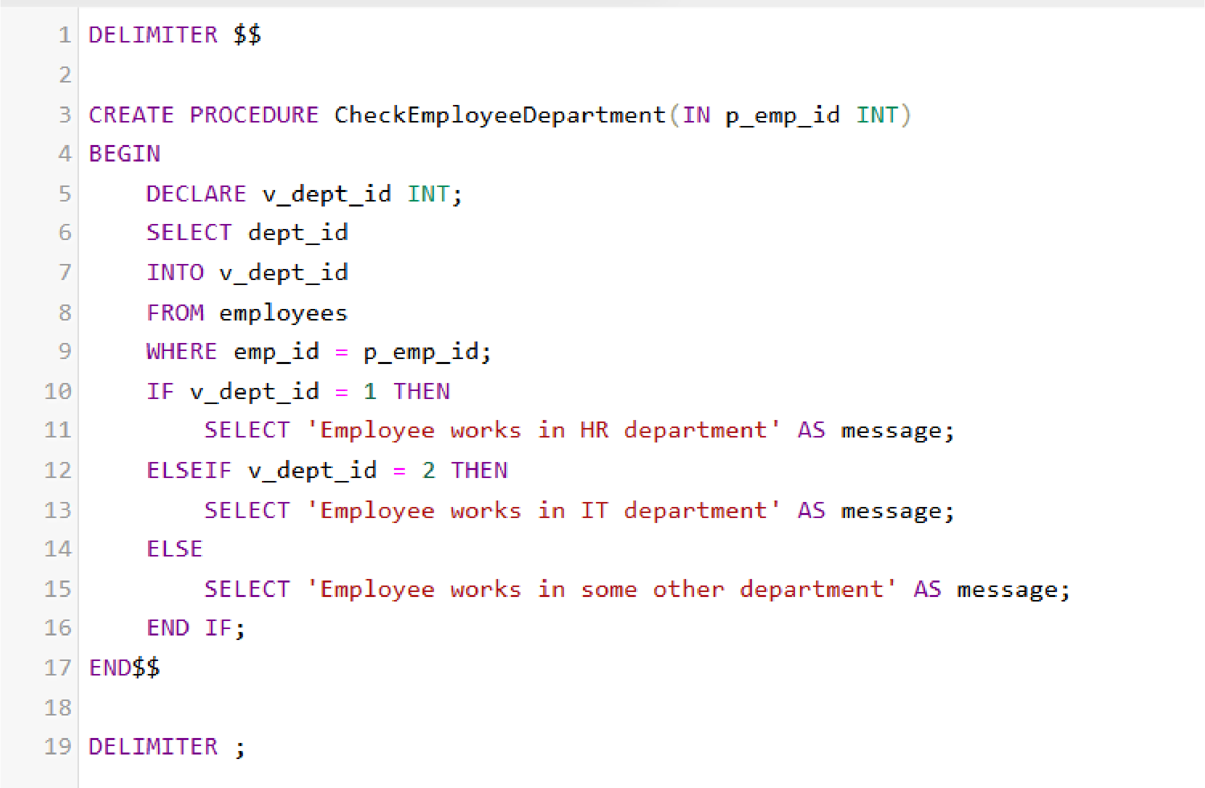
```sql
CALL GetTotalSales();
```



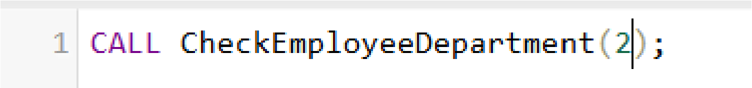
message
Total Sales: $1,321.24

17. PL/SQL Control Structures

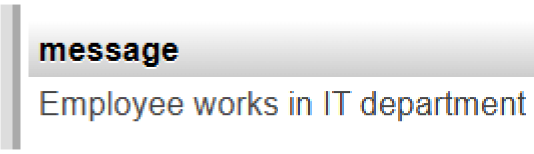
Lab 1: Write a PL/SQL block using an IF-THEN condition to check the department of an employee.



```
DELIMITER $$

CREATE PROCEDURE CheckEmployeeDepartment(IN p_emp_id INT)
BEGIN
    DECLARE v_dept_id INT;
    SELECT dept_id
    INTO v_dept_id
    FROM employees
    WHERE emp_id = p_emp_id;
    IF v_dept_id = 1 THEN
        SELECT 'Employee works in HR department' AS message;
    ELSEIF v_dept_id = 2 THEN
        SELECT 'Employee works in IT department' AS message;
    ELSE
        SELECT 'Employee works in some other department' AS message;
    END IF;
END$$

DELIMITER ;
```

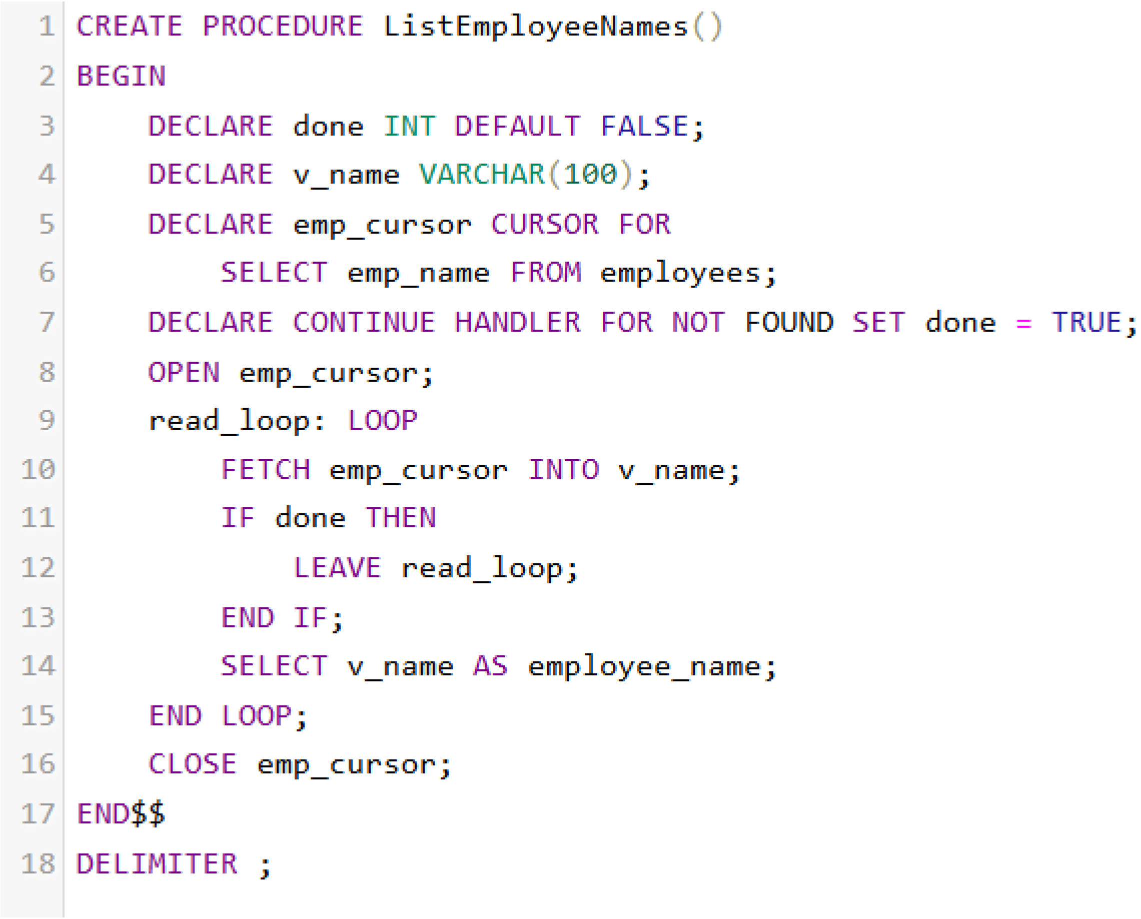


```
CALL CheckEmployeeDepartment(2);
```

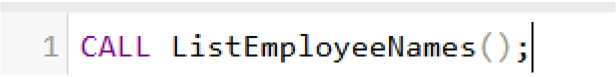


```
message
Employee works in IT department
```

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



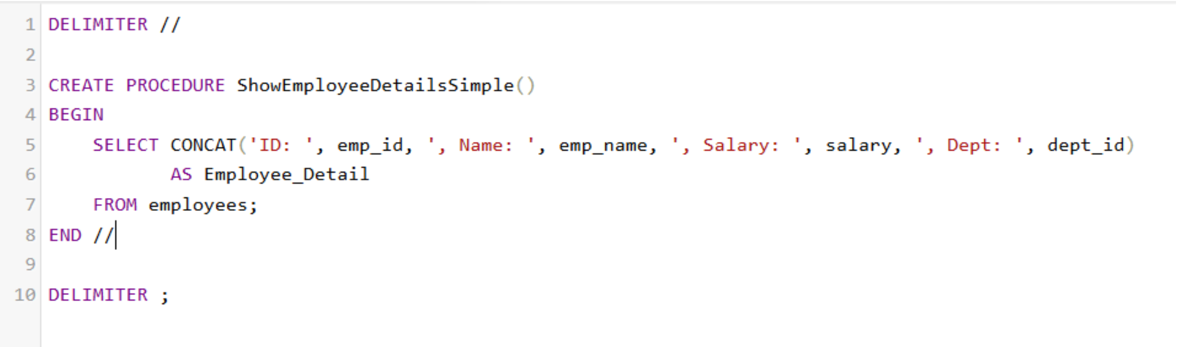
```sql
CREATE PROCEDURE ListEmployeeNames()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE v_name VARCHAR(100);
    DECLARE emp_cursor CURSOR FOR
        SELECT emp_name FROM employees;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN emp_cursor;
    read_loop: LOOP
      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



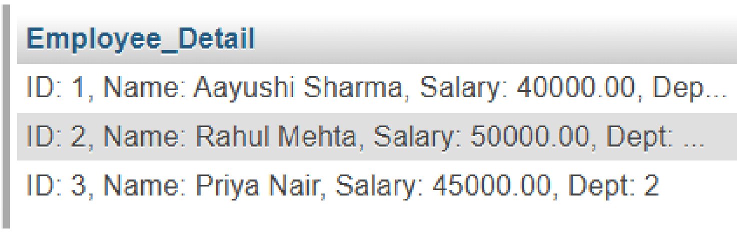
```sql
CALL ListEmployeeNames();
```

18. SQL Cursors

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

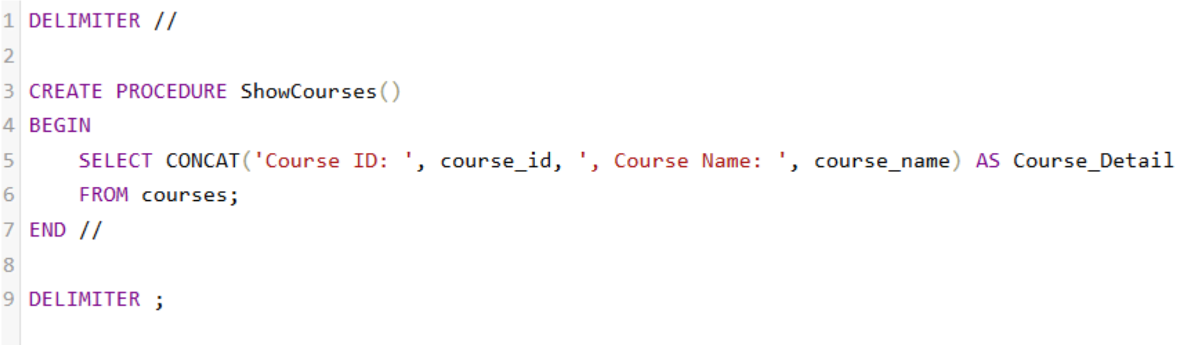


```sql
DELIMITER //

CREATE PROCEDURE ShowEmployeeDetailsSimple()
BEGIN
    SELECT CONCAT('ID: ', emp_id, ', Name: ', emp_name, ', Salary: ', salary, ', Dept: ', dept_id)
           AS Employee_Detail
    FROM employees;
END //

DELIMITER ;
```

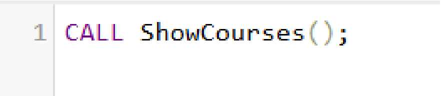


Employee_Detail
ID: 1, Name: Aayushi Sharma, Salary: 40000.00, Dep...
ID: 2, Name: Rahul Mehta, Salary: 50000.00, Dept: ...
ID: 3, Name: Priya Nair, Salary: 45000.00, Dept: 2

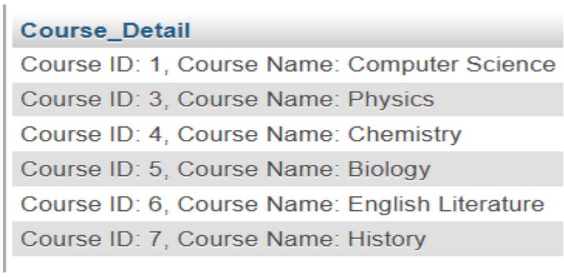
Lab 2: Create a cursor to retrieve all courses and display them one by one.



```sql
DELIMITER //

CREATE PROCEDURE ShowCourses()
BEGIN
    SELECT CONCAT('Course ID: ', course_id, ', Course Name: ', course_name) AS Course_Detail
    FROM courses;
END //

DELIMITER ;
```



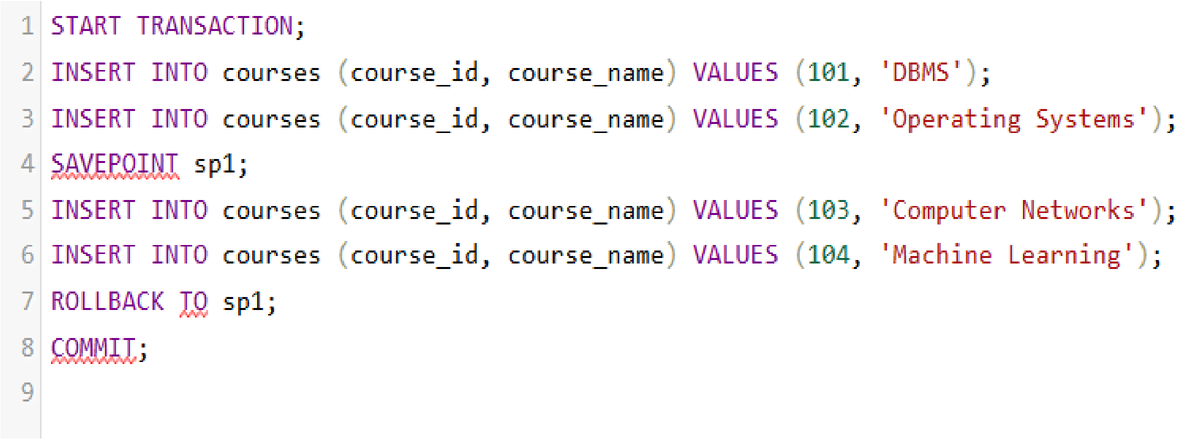
```sql
CALL ShowCourses();
```



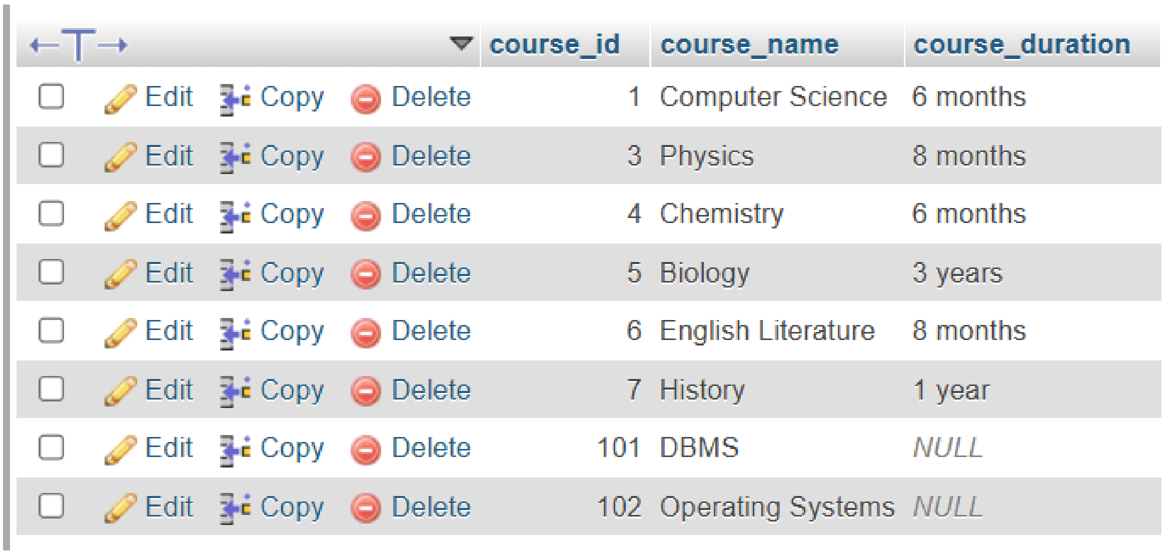
Course_Detail
Course ID: 1, Course Name: Computer Science
Course ID: 3, Course Name: Physics
Course ID: 4, Course Name: Chemistry
Course ID: 5, Course Name: Biology
Course ID: 6, Course Name: English Literature
Course ID: 7, Course Name: History

19. Rollback and Commit Savepoint

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

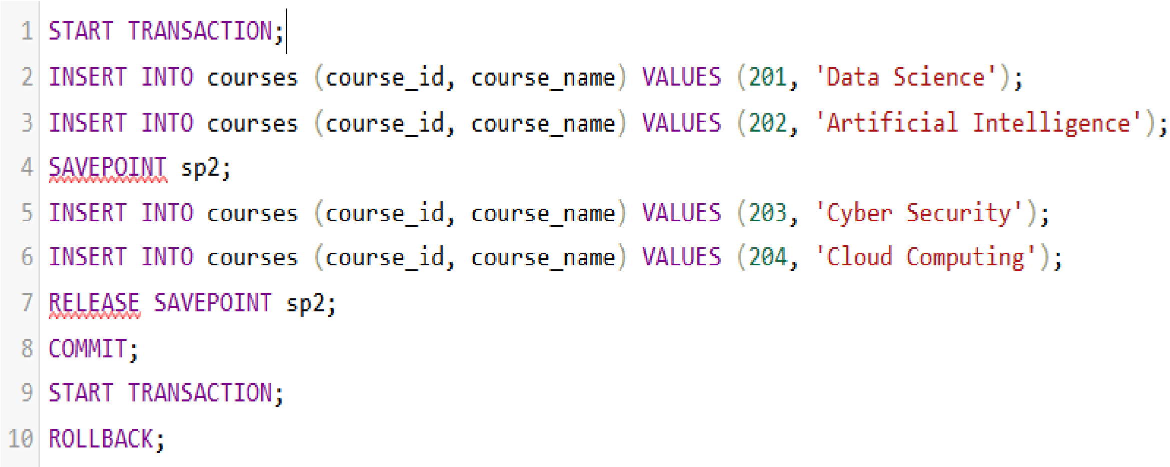


```
START TRANSACTION;
INSERT INTO courses (course_id, course_name) VALUES (101, 'DBMS');
INSERT INTO courses (course_id, course_name) VALUES (102, 'Operating Systems');
SAVEPOINT sp1;
INSERT INTO courses (course_id, course_name) VALUES (103, 'Computer Networks');
INSERT INTO courses (course_id, course_name) VALUES (104, 'Machine Learning');
ROLLBACK TO sp1;
COMMIT;

```

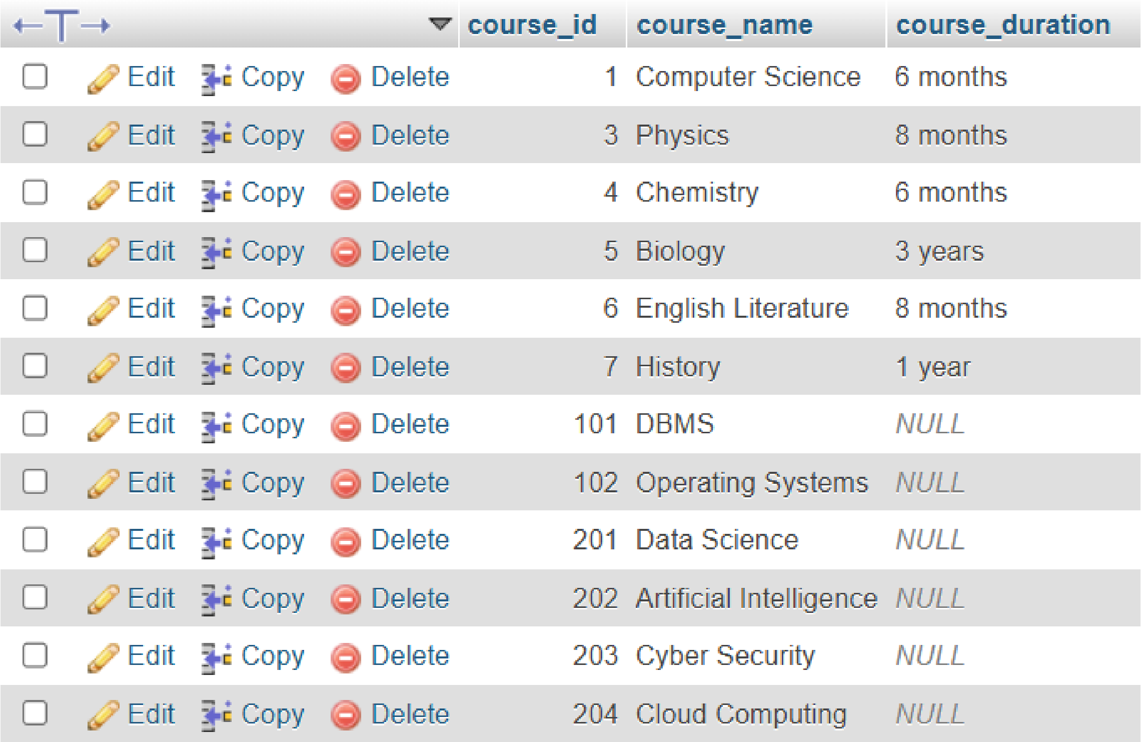


←T→				course_id	course_name	course_duration
☐	Edit	Copy	Delete	1	Computer Science	6 months
☐	Edit	Copy	Delete	3	Physics	8 months
☐	Edit	Copy	Delete	4	Chemistry	6 months
☐	Edit	Copy	Delete	5	Biology	3 years
☐	Edit	Copy	Delete	6	English Literature	8 months
☐	Edit	Copy	Delete	7	History	1 year
☐	Edit	Copy	Delete	101	DBMS	*NULL*
☐	Edit	Copy	Delete	102	Operating Systems	*NULL*

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



```sql
START TRANSACTION;
INSERT INTO courses (course_id, course_name) VALUES (201, 'Data Science');
INSERT INTO courses (course_id, course_name) VALUES (202, 'Artificial Intelligence');
SAVEPOINT sp2;
INSERT INTO courses (course_id, course_name) VALUES (203, 'Cyber Security');
INSERT INTO courses (course_id, course_name) VALUES (204, 'Cloud Computing');
RELEASE SAVEPOINT sp2;
COMMIT;
START TRANSACTION;
ROLLBACK;
```



←T→			▼	course_id	course_name	course_duration
□	Edit	Copy	Delete	1	Computer Science	6 months
□	Edit	Copy	Delete	3	Physics	8 months
□	Edit	Copy	Delete	4	Chemistry	6 months
□	Edit	Copy	Delete	5	Biology	3 years
□	Edit	Copy	Delete	6	English Literature	8 months
□	Edit	Copy	Delete	7	History	1 year
□	Edit	Copy	Delete	101	DBMS	*NULL*
□	Edit	Copy	Delete	102	Operating Systems	*NULL*
□	Edit	Copy	Delete	201	Data Science	*NULL*
□	Edit	Copy	Delete	202	Artificial Intelligence	*NULL*
□	Edit	Copy	Delete	203	Cyber Security	*NULL*
□	Edit	Copy	Delete	204	Cloud Computing	*NULL*