*Name: - Bhavesh Goswami*

**Module 5 – Introduction to DBMS**

**lab exercises:-**

* **Introduction to SQL**

**Lab 1: Create a new database named school\_db and a table called students with the following columns: student\_id, student\_name, age, class, and address.**

* CREATE DATABASE school\_db;
* CREATE TABLE student

(

student\_id int AUTO\_INCREMENT PRIMARY KEY,

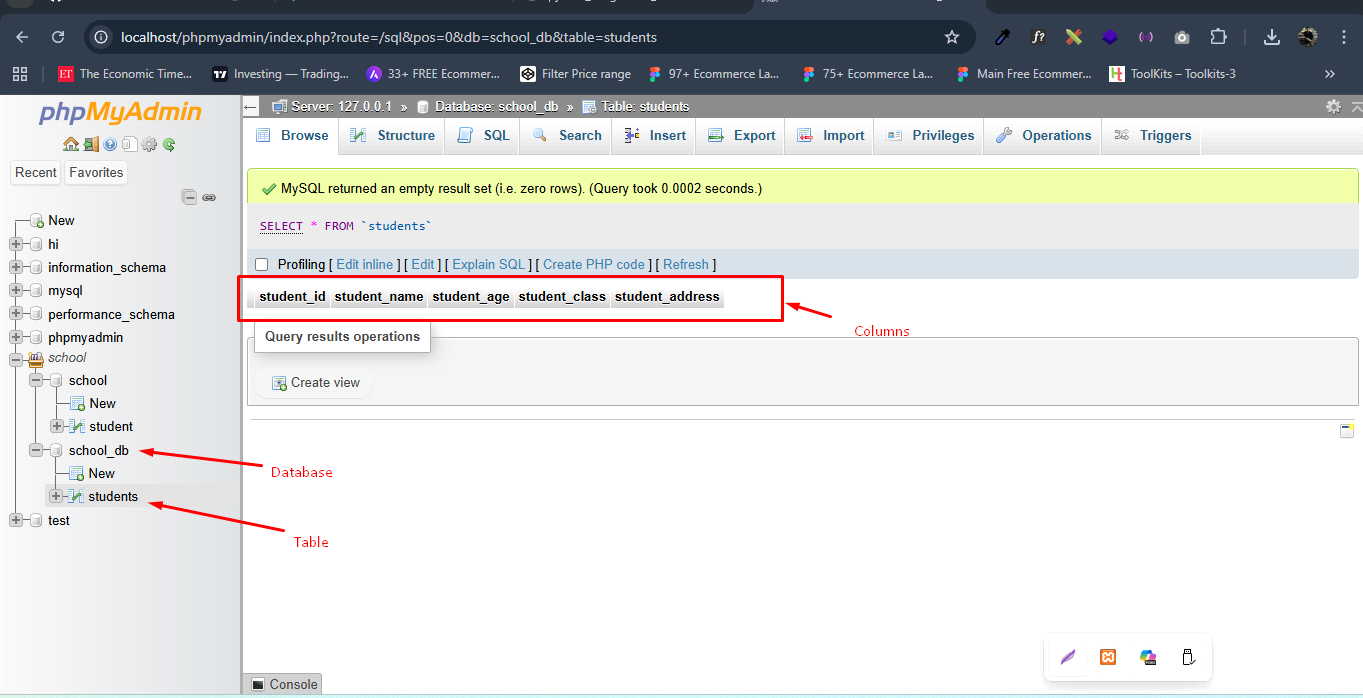
student\_name varchar(50),

student\_age int ,

student\_class int,

student\_address varchar(100)

);



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

* INSERT INTO students(student\_name, student\_age, student\_class, student\_address)

VALUES

('Bhavesh Goswami', 25, 12, 'B-230, subbham socity, Ahmedabad'),

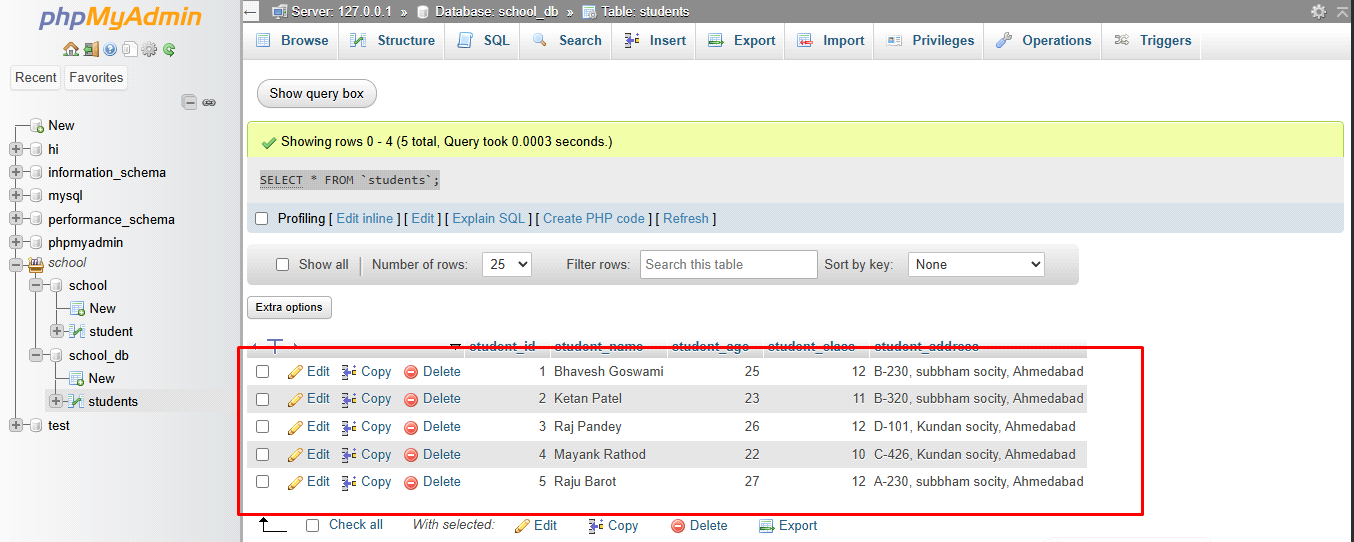
('Ketan Patel', 23, 11,'B-320, subbham socity, Ahmedabad'),

('Raj Pandey', 26, 12, 'D-101, Kundan socity, Ahmedabad'),

('Mayank Rathod', 22, 10, 'C-426, Kundan socity, Ahmedabad'),

('Raju Barot', 27, 12, 'A-230, subbham socity, Ahmedabad');

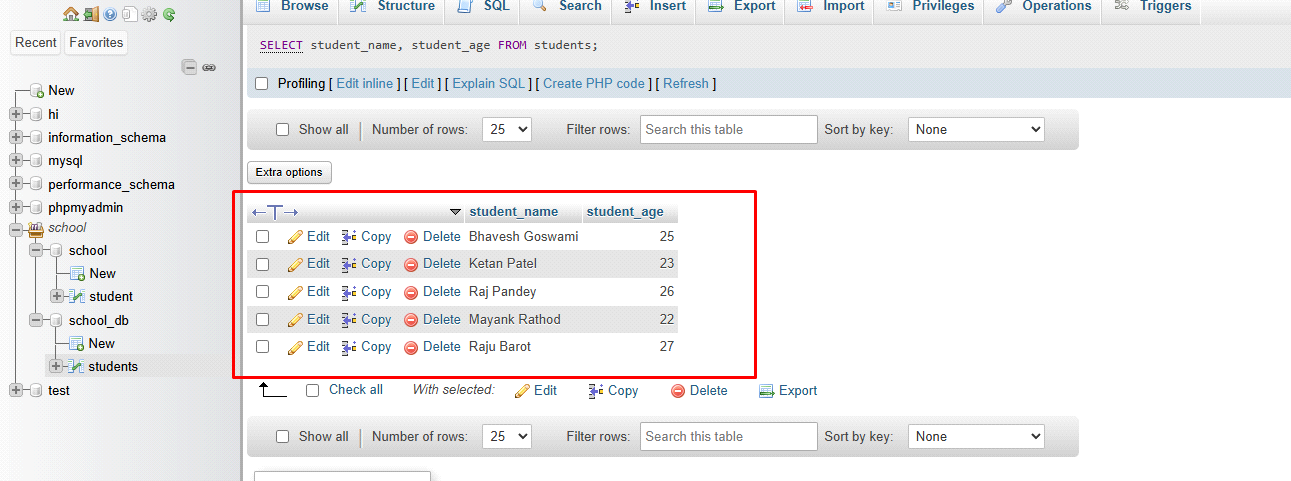
* [SELECT](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/select.html) \* FROM students;



* **SQL Syntax**

**Lab 1: Write SQL queries to retrieve specific columns (student\_name and age) from the students table.**

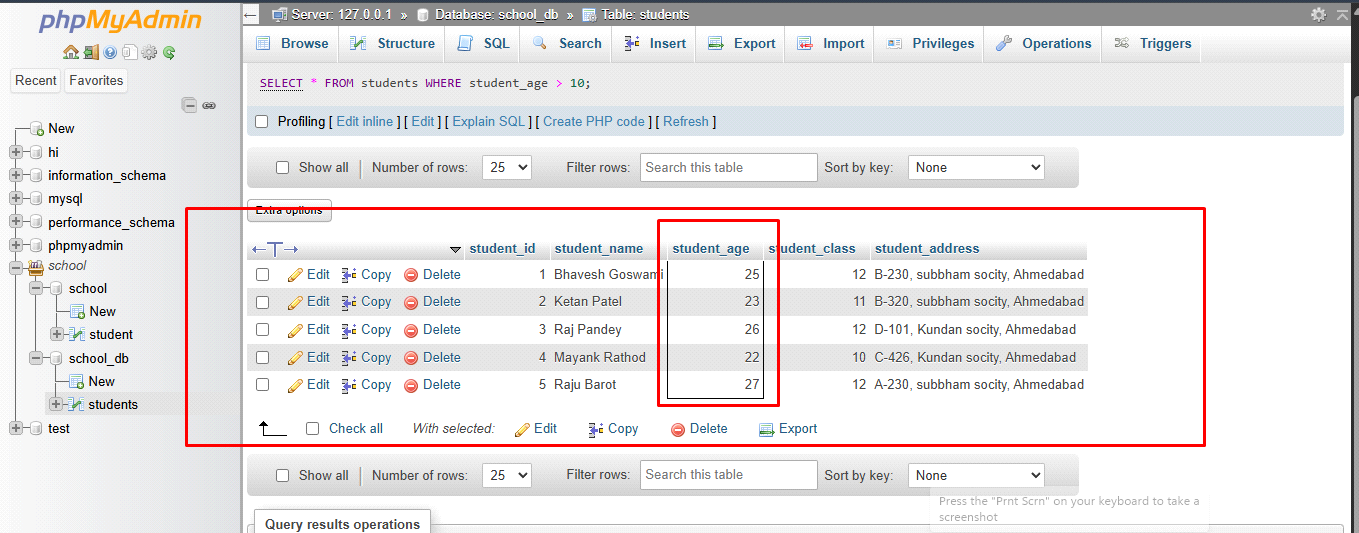
* SELECT student\_name, student\_age FROM students;



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

* SELECT \* FROM students

WHERE student\_age > 10;



* **SQL Constraints**

**Lab 1: Create a table teachers with the following columns: teacher\_id (Primary Key), teacher\_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).**

* CREATE TABLE teachers

**(**

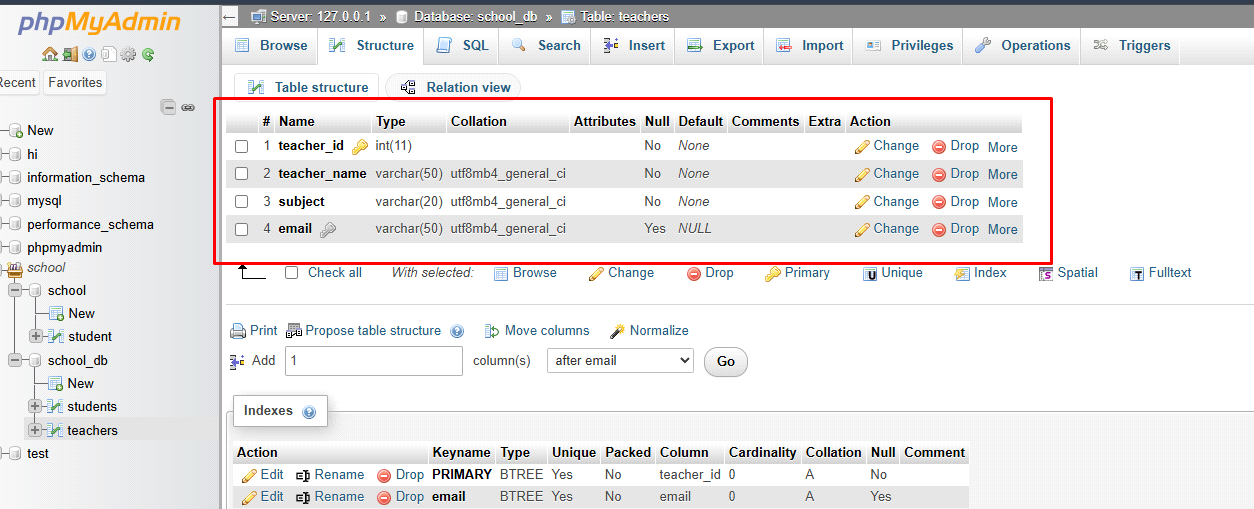
teacher\_id INT PRIMARY KEY,

teacher\_name varchar(50) NOT NULL,

subject varchar(20) NOT NULL,

email varchar(50) UNIQUE

**);**



**Lab 2: Implement a FOREIGN KEY constraint to relate the teacher\_id from the teachers table with the students table.**

* **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

(

Courses\_id int PRIMARY KEY,

courses\_name varchar(20),

courses\_credit INT

)



**Lab 2: Use the CREATE command to create a database university\_db.**

* CREATE DATABASE university\_db;

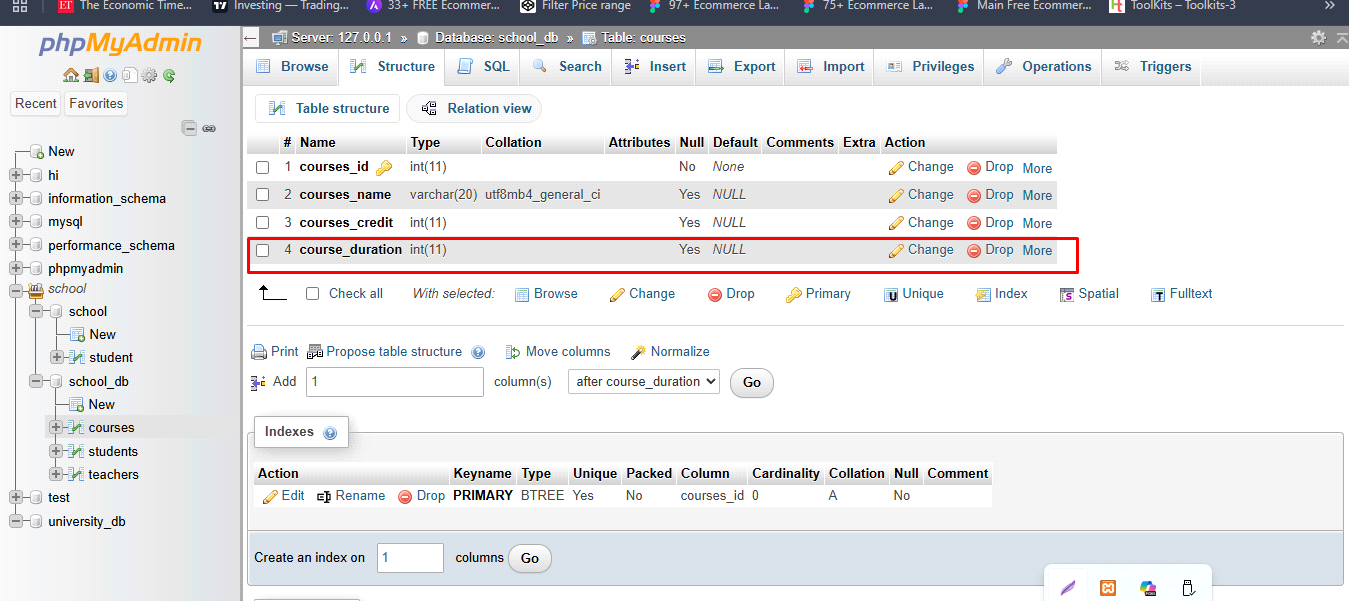


* **ALTER Command**

**Lab 1: Modify the courses table by adding a column course\_duration using the ALTER command.**

* ALTER TABLE courses

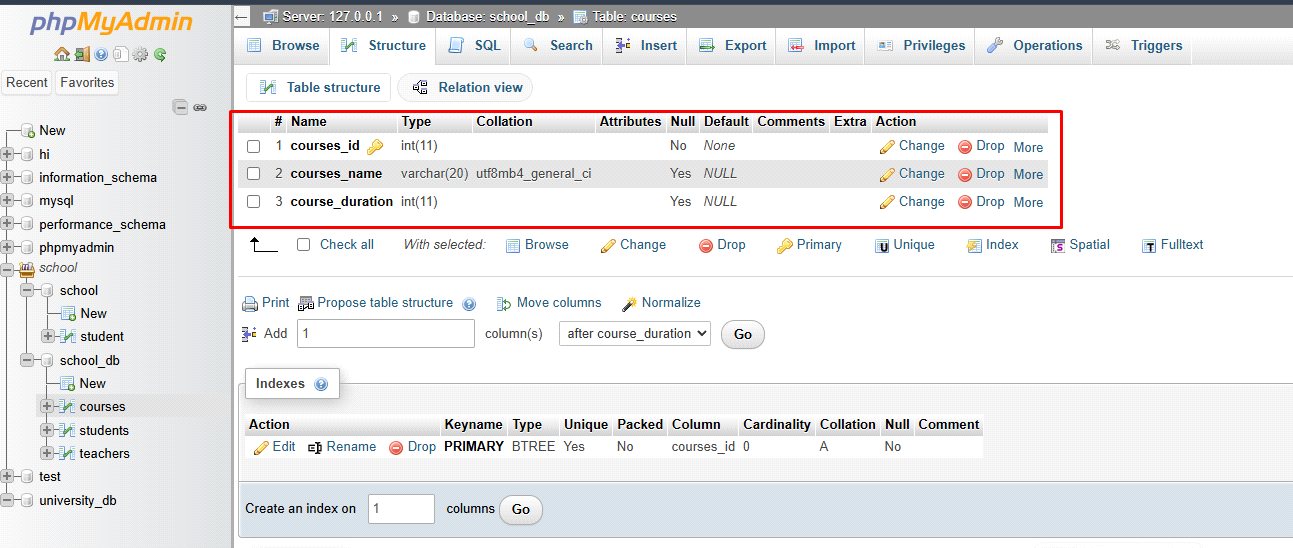
ADD COLUMN course\_duration INT;



**Lab 2: Drop the course\_credits column from the courses table.**

* ALTER TABLE courses

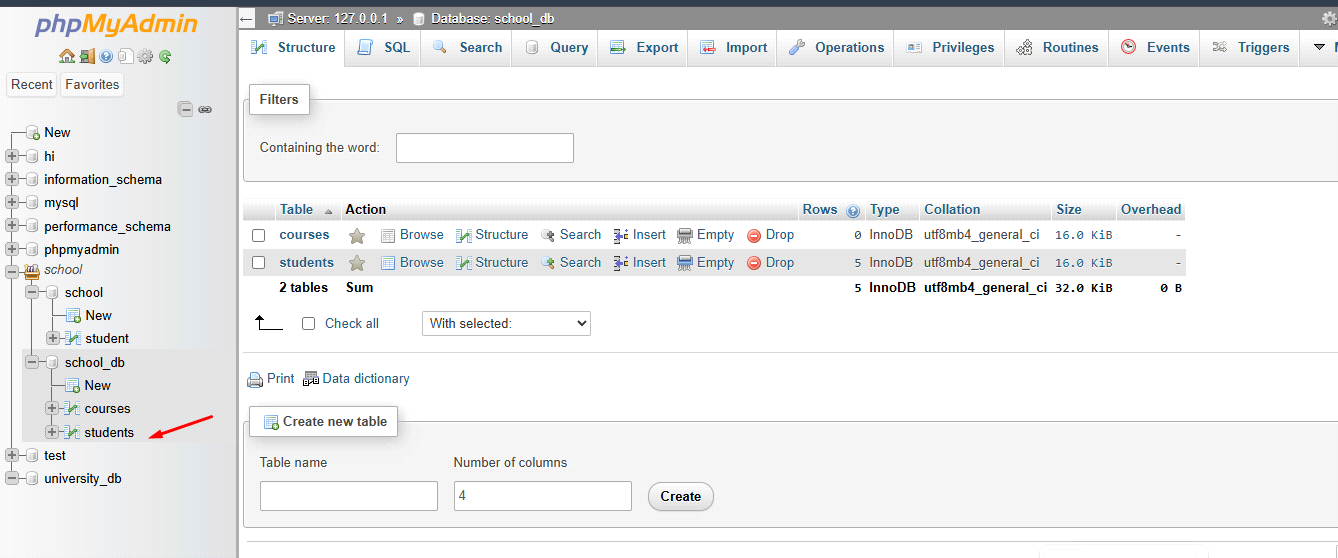
DROP courses\_credit;



* **DROP Command**

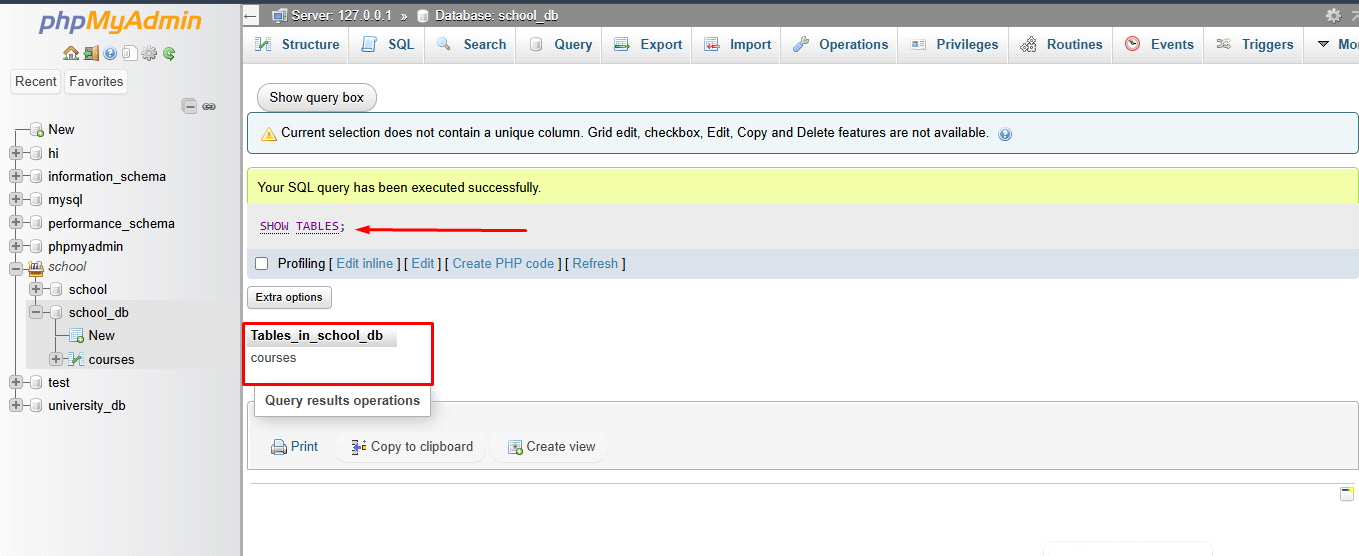
**Lab 1: Drop the teachers table from the school\_db database.**

* DROP TABLE teachers;



**Lab 2: Drop the students table from the school\_db database and verify that the table has been removed.**

* DROP TABLE students;



* **Data Manipulation Language (DML)**

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

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

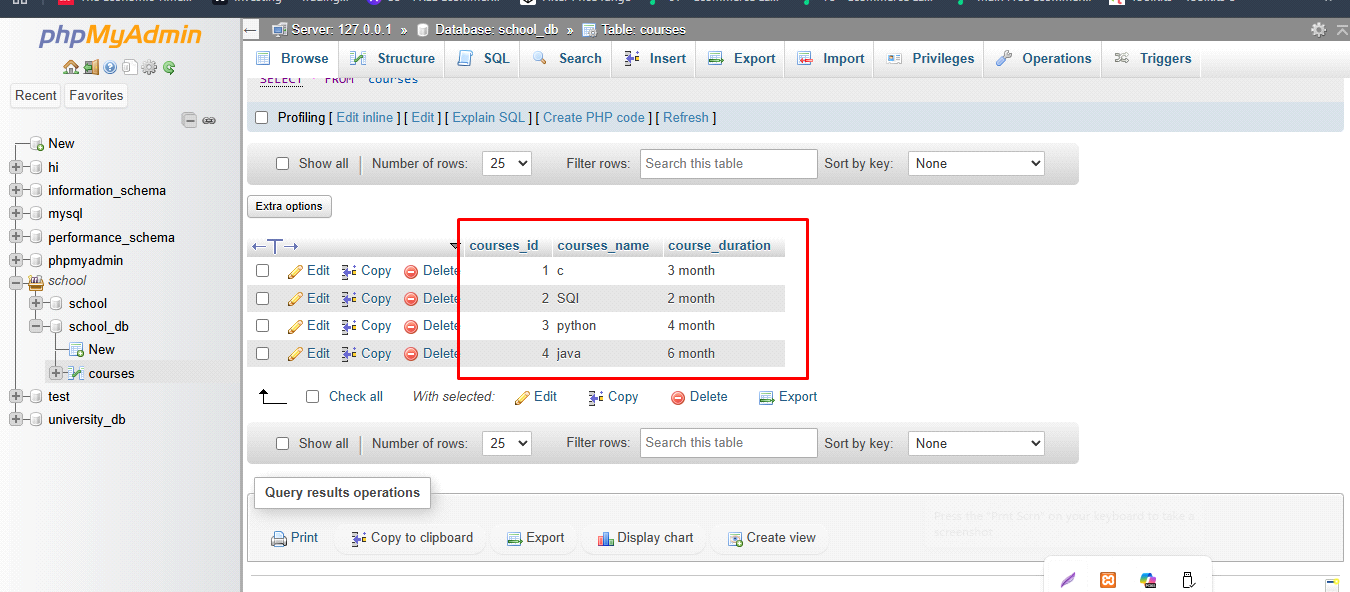
VALUES

('c', '3 month'),

('SQL', '2 month'),

('python', '4 month'),

('java', '6 month');



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

* UPDATE courses

SET course\_duration = '6 month'

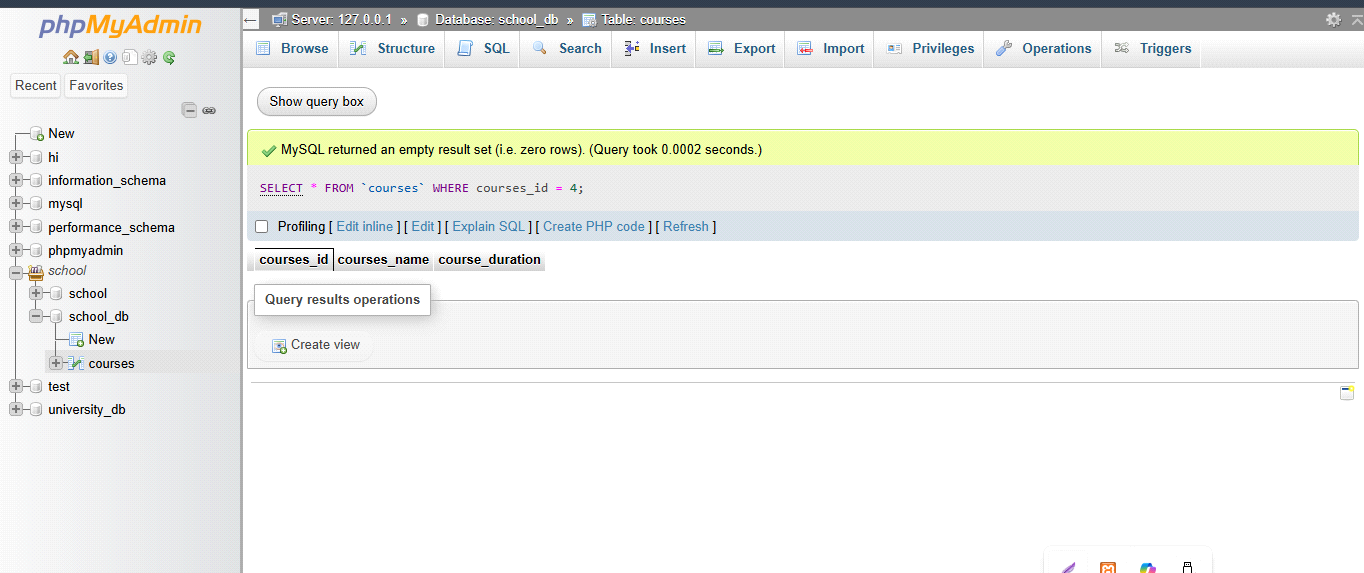
WHERE courses\_id = 3;



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

* DELETE FROM courses

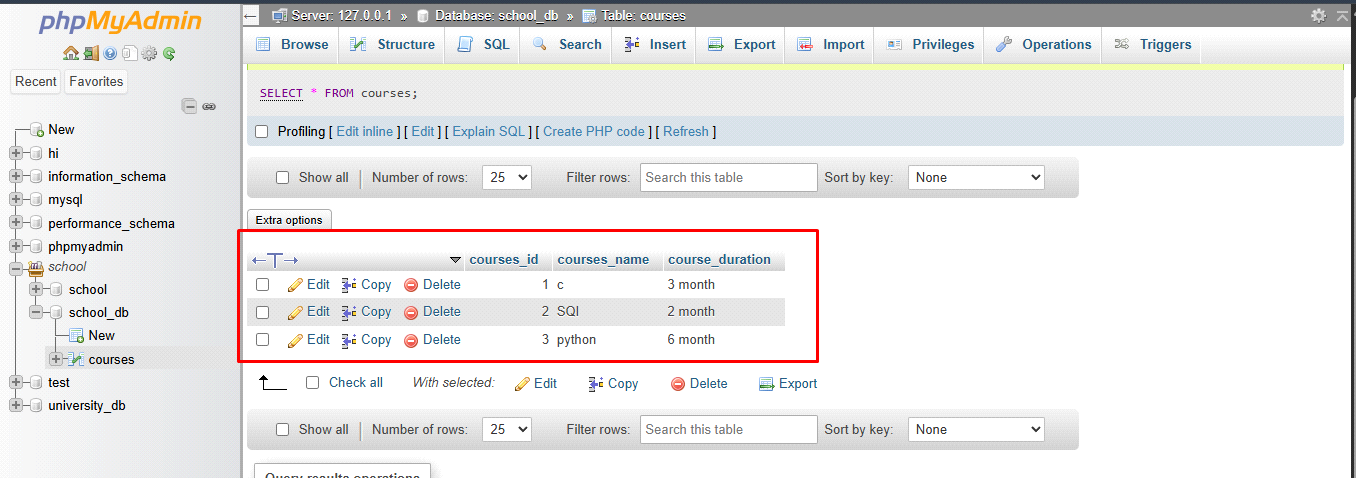
WHERE courses\_id = 4;



* **Data Query Language (DQL)**

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

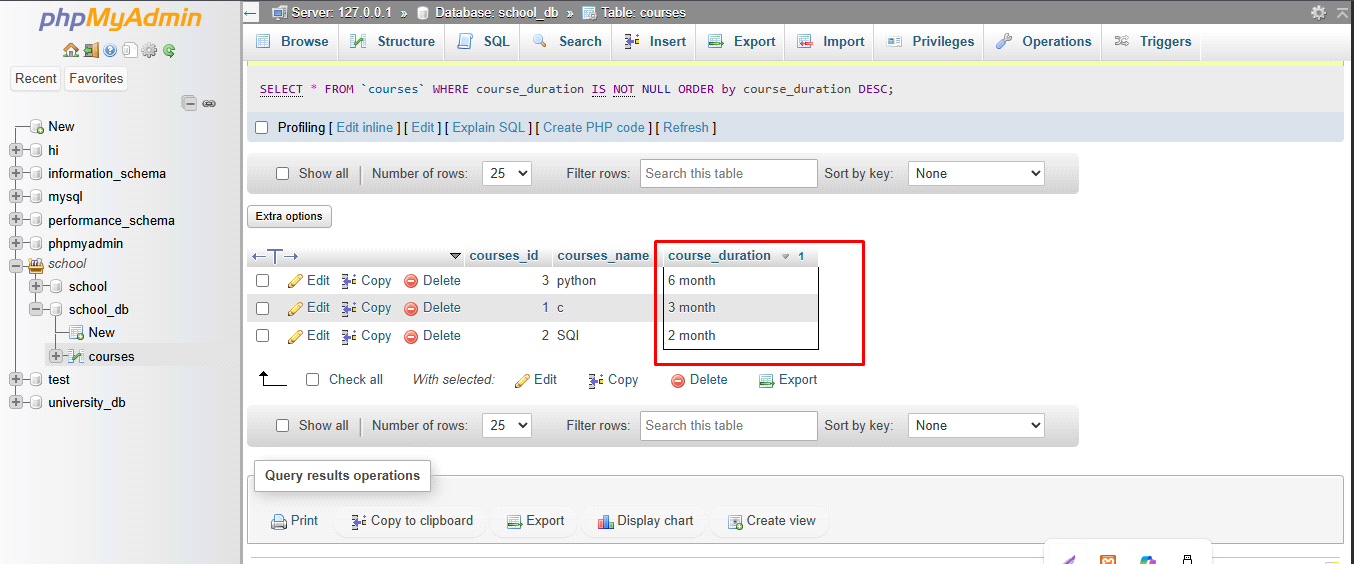
* SELECT \* FROM courses;



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

* SELECT \* FROM `courses`

ORDER by course\_duration DESC;

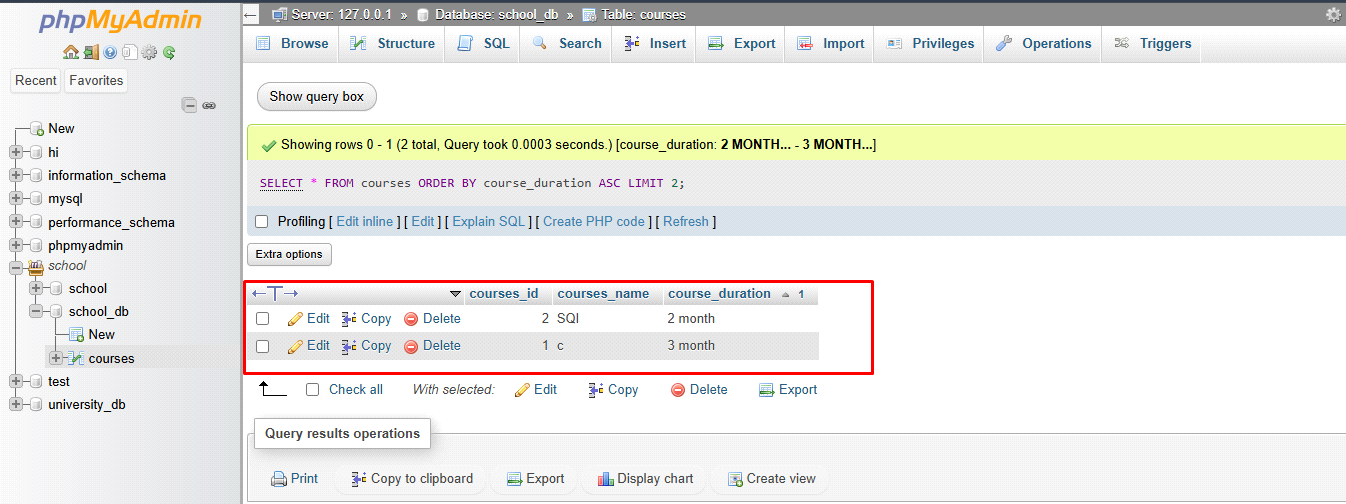


**Lab 3: Limit the results of the SELECT query to show only the top two courses using LIMIT.**

* SELECT \* FROM courses

ORDER BY course\_duration ASC

LIMIT 2;



* **Data Control Language (DCL)**

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

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

* **Transaction Control Language (TCL)**

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

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

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

* **SQL Joins**

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

- CREATE DATABASE office;

- CREATE TABLE departments

(

d\_id int AUTO\_INCREMENT PRIMARY KEY,

d\_name varchar(20)

);

- INSERT INTO departments(d\_name)

VALUES

('HR'),

('Manager'),

('sales'),

('front-end'),

('Back-end');

('Full stack'),

('Tester');

- CREATE TABLE employees

(

e\_id int PRIMARY key AUTO\_INCREMENT,

e\_name varchar(20),

e\_salary decimal(10, 2),

d\_id int,

FOREIGN key (d\_id) REFERENCES departments(d\_id)

);

- INSERT INTO employees(e\_name, e\_salary, d\_id)

VALUES

('Bhavesh', '25000', 1),

('Rohan', '30000', 2),

('Kishan', '28000', 3),

('Keyur', '22000', 4),

('Tushar', '23000', 5);

SELECT

employees.e\_id,

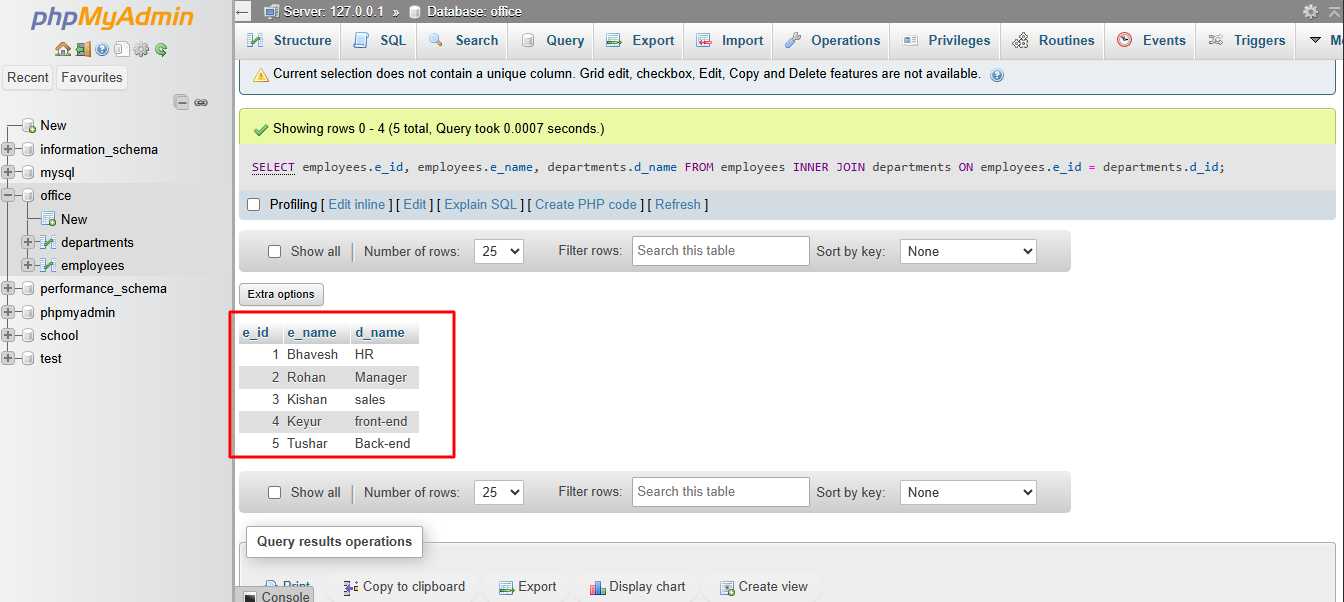
employees.e\_name,

departments.d\_name

FROM employees

INNER JOIN departments

ON employees.e\_id = departments.d\_id;



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

**-** SELECT

employees.e\_id,

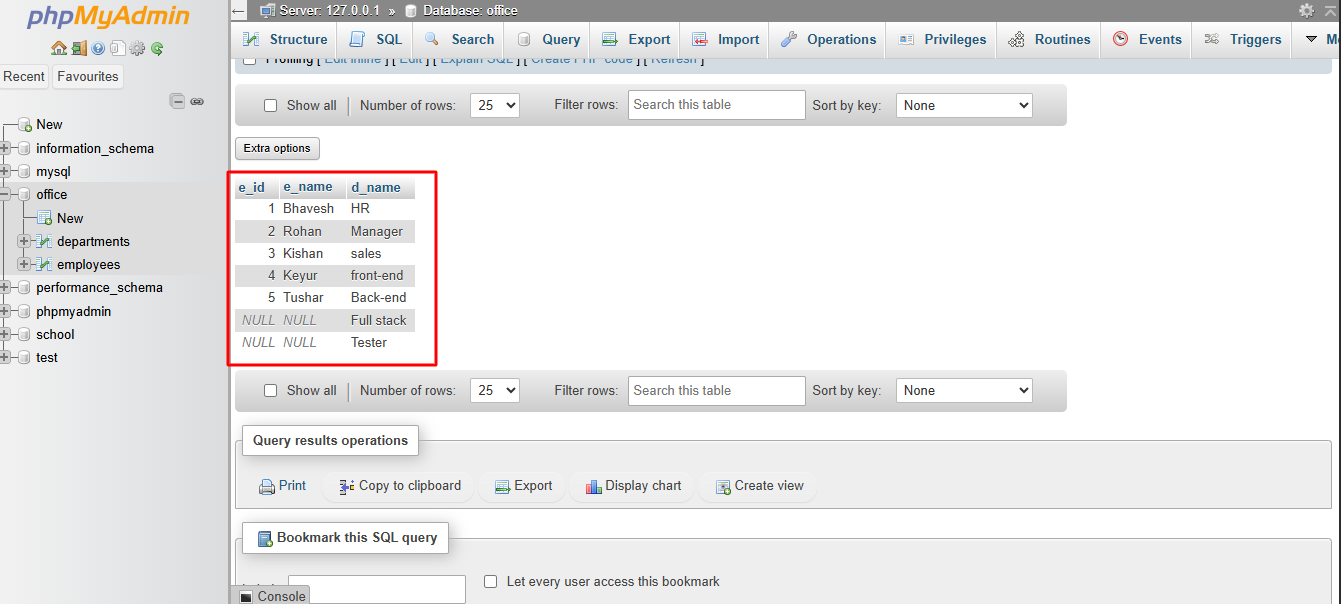
employees.e\_name,

departments.d\_name

FROM departments

LEFT JOIN employees

ON departments.d\_id = employees.e\_id;



* **SQL Group By**

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

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

* **QL Stored Procedure**

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

**Lab 2: Write a stored procedure that accepts course\_id as input and returns the course details**

* **SQL View**

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

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

* **SQL Triggers**

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

**Lab 2: Create a trigger to update the last\_modified timestamp whenever an employee record is updated.**

* **Introduction to PL/SQL**

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

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

* **PL/SQL Control Structures**

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

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

* **SQL Cursors**

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

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

* **Rollback and Commit Savepoint**

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

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

**EXTRA LAB PRACTISE FOR DATABASE CONCEPTS**

**1. Introduction to SQL**

**Lab 3: Create a database called library\_db and a table books with columns: book\_id, title, author, publisher, year\_of\_publication, and price. Insert five records into the table.**

- CREAT DATABASE library\_db;

- CREATE TABLE books

(

book\_id int PRIMARY KEY AUTO\_INCREMENT,

b\_title varchar(50),

b\_author varchar(20),

b\_publisher varchar(20),

b\_year\_of\_publication int,

b\_price int

);

- INSERT INTO books

(b\_title, b\_author, b\_publisher, b\_year\_of\_publication, b\_price)

VALUES

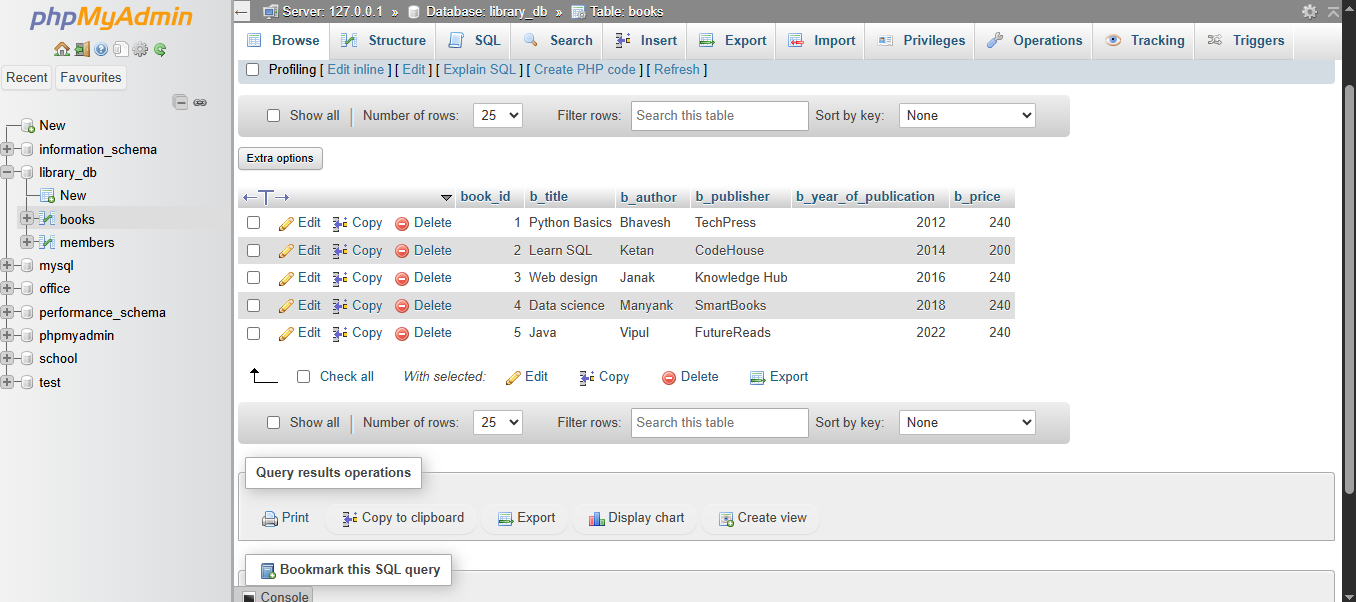
('Python Basics', 'Bhavesh', 'TechPress', 2012, 240),

('Learn SQL', 'Ketan', 'CodeHouse', 2014, 200),

('Web design', 'Janak', 'Knowledge Hub', 2016, 240),

('Data science', 'Manyank', 'SmartBooks', 2018, 240),

('Java', 'Vipul', 'FutureReads', 2022, 240);



**Lab 4: Create a table members in library\_db with columns: member\_id, member\_name, date\_of\_membership, and email. Insert five records into this table.**

**-** CREATE DATABASE library\_db;

- CREATE TABLE members (

member\_id INT PRIMARY KEY,

member\_name VARCHAR(50) NOT NULL,

date\_of\_membership DATE NOT NULL,

email VARCHAR(100) NOT NULL

);

- INSERT INTO members (member\_id, member\_name, date\_of\_membership, email)

VALUES

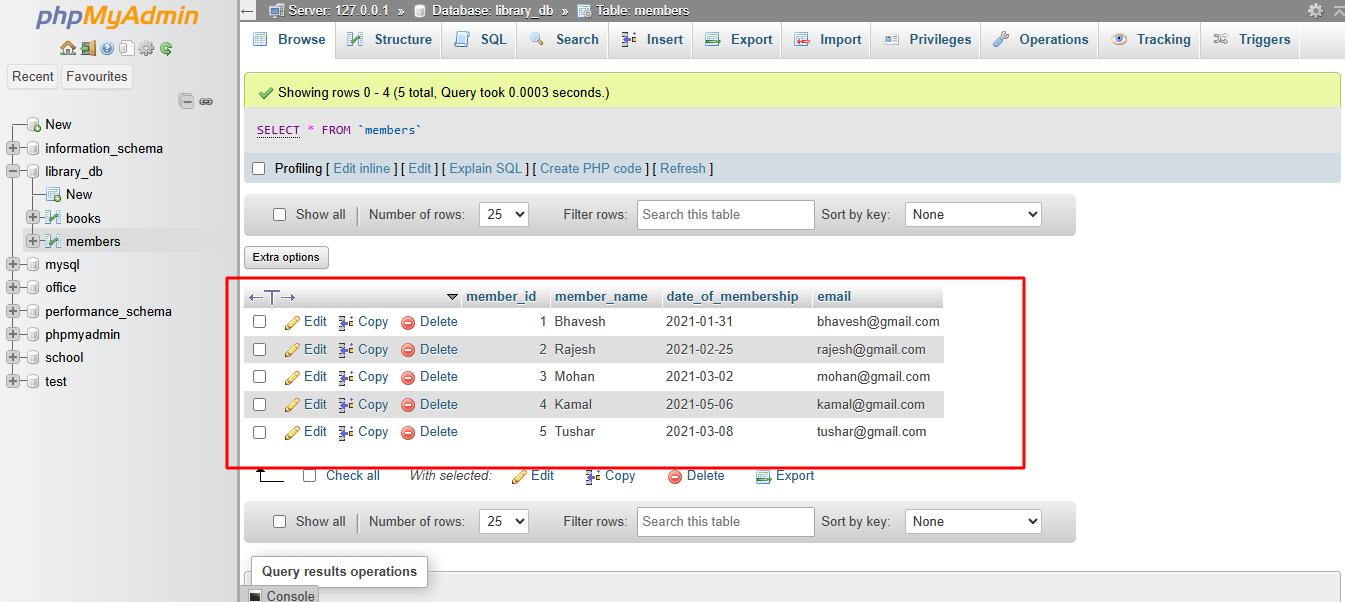
(1, 'Bhavesh', '2021-01-31', 'bhavesh@gmail.com'),

(2, 'Rajesh', '2021-02-25', 'rajesh@gmail.com'),

(3, 'Mohan', '2021-03-02', 'mohan@gmail.com'),

(4, 'Kamal', '2021-05-06', 'kamal@gmail.com'),

(5, 'Tushar', '2021-03-08', 'tushar@gmail.com');



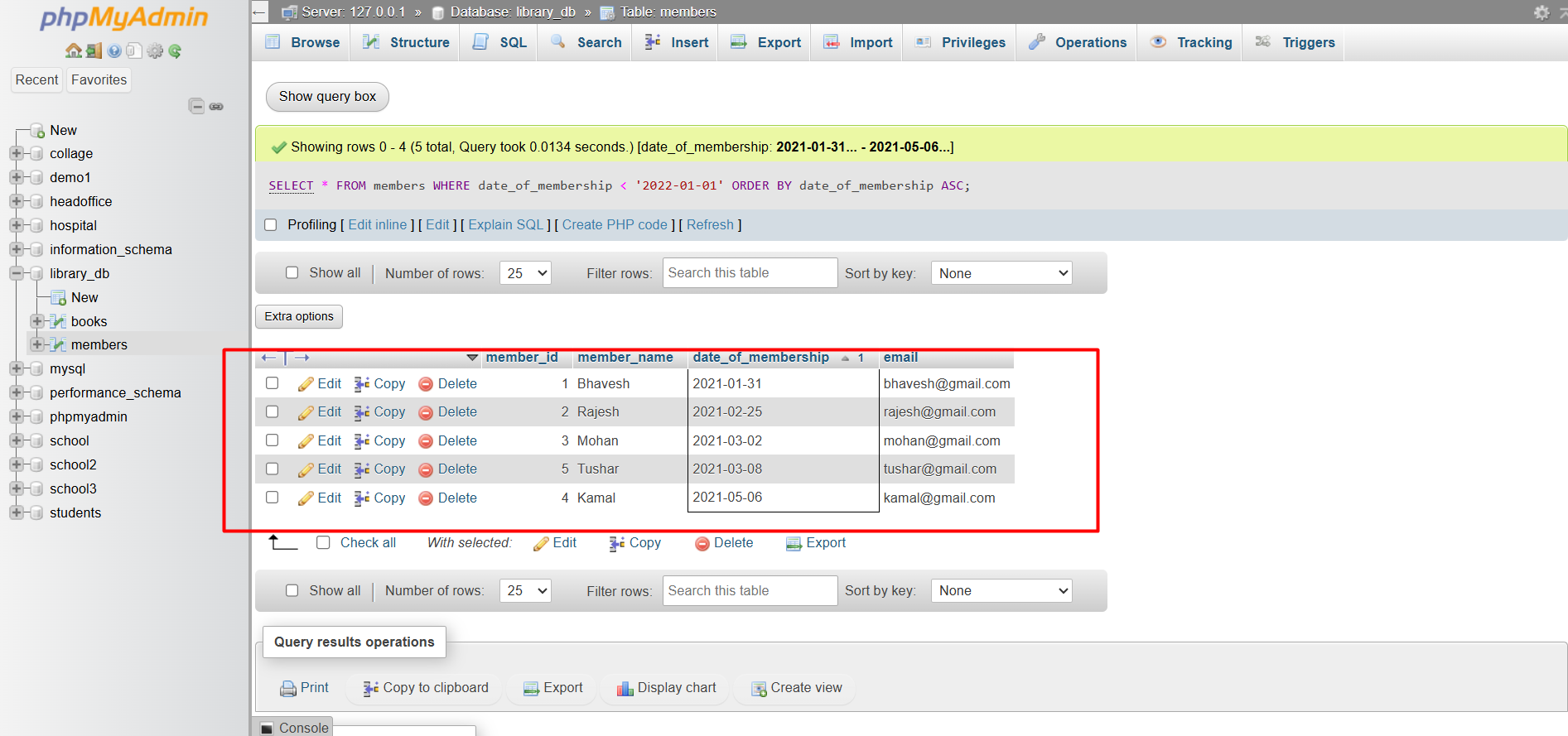
**2. SQL Syntax**

**Lab 3: Retrieve all members who joined the library before 2022. Use appropriate SQL syntax with WHERE and ORDER BY.**

**Ans.** SELECT \* FROM members

WHERE date\_of\_membership < '2022-01-01'

ORDER BY date\_of\_membership ASC;



**Lab 4: Write SQL queries to display the titles of books published by a specific author. Sort the**

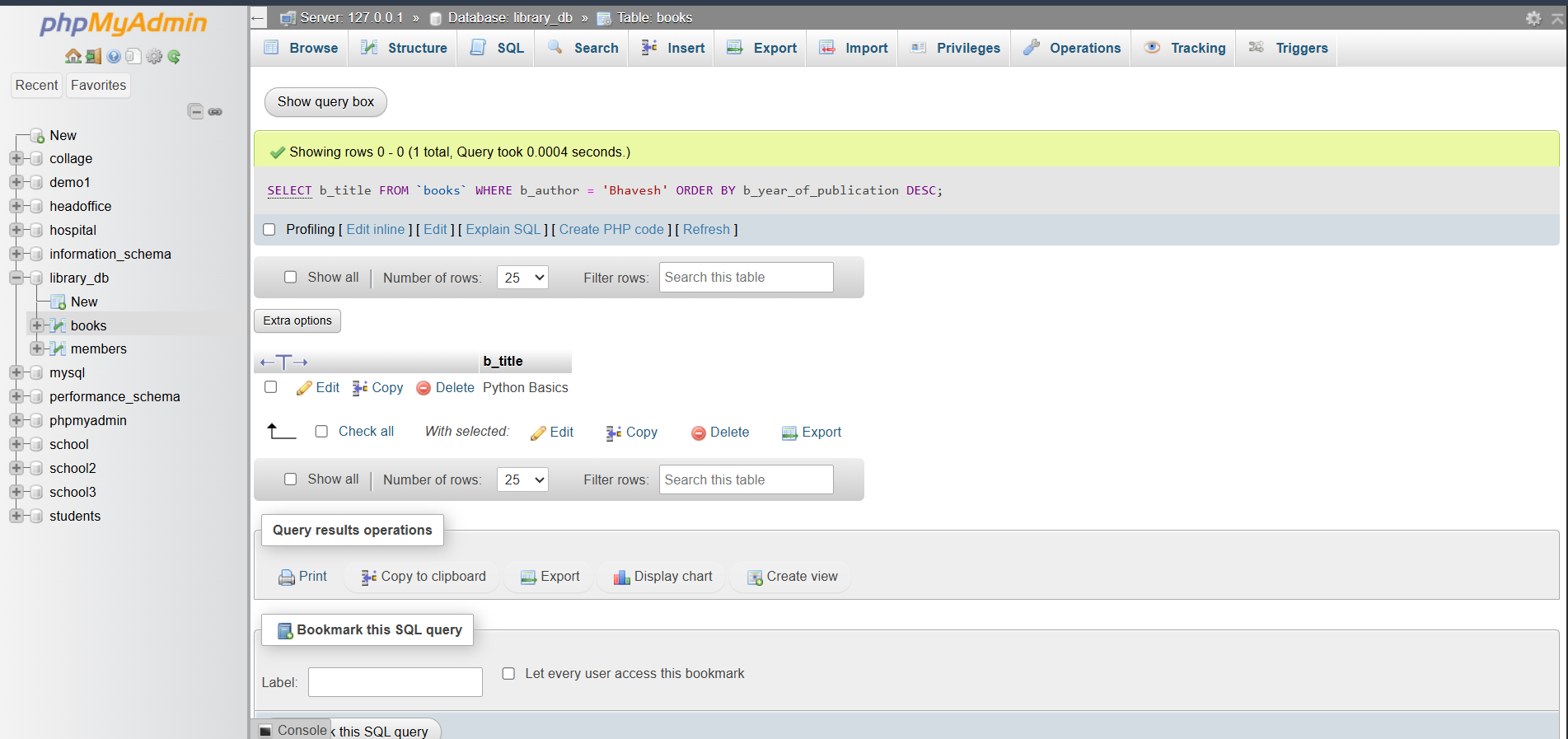
**results by year\_of\_publication in descending order.**

**Ans.**

SELECT b\_title FROM `books`

WHERE b\_author = 'Bhavesh'

ORDER BY b\_year\_of\_publication DESC;



**3. SQL Constraints**

**Lab 3: Add a CHECK constraint to ensure that the price of books in the books table is**

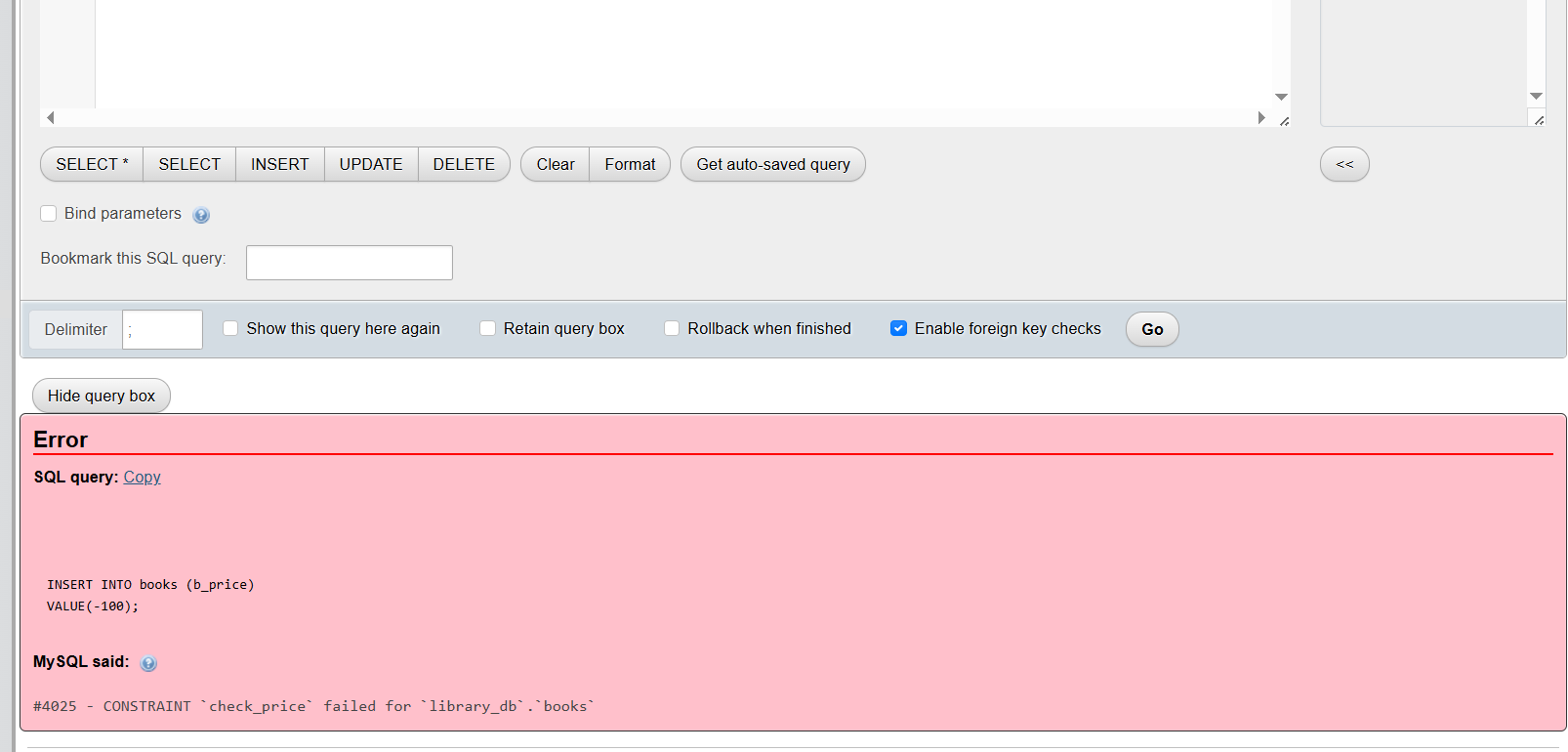
**greater than 0.**

**Ans.**

ALTER TABLE books

ADD CONSTRAINT Check\_price

CHECK (b\_price > 0);



**Lab 4: Modify the members table to add a UNIQUE constraint on the email column,**

**ensuring that each member has a unique email address.**

**Ans.**

ALTER TABLE members

ADD CONSTRAINT UNIQUE\_EMAIL

 UNIQUE (email);

